SIAM J. COMPUT.
Vol. 28, No. 1, pp. 1–9

OPTIMAL BIWEIGHTED BINARY TREES AND THE COMPLEXITY OF MAINTAINING PARTIAL SUMS*

HARIPRIYAN HAMPAPURAM† AND MICHAEL L. FREDMAN‡

Abstract. Let A be an array. The *partial sum problem* concerns the design of a data structure for implementing the following operations. The operation update(j, x) has the effect $A[j] \leftarrow A[j] + x$, and the query operation sum(j) returns the partial sum $\sum_{i=1}^{j} A[i]$. Our interest centers upon the optimal efficiency with which sequences of such operations can be performed, and we derive new upper and lower bounds in the semigroup model of computation. Our analysis relates the optimal complexity of the partial sum problem to optimal binary trees relative to a type of weighting scheme that defines the notion of *biweighted* binary tree.

Key words. data structures, partial sums, lower bounds

AMS subject classifications. 68P05, 68Q25

PII. S0097539795291598

1. Introduction.

Let A be an array of length N. The *partial sum problem* concerns the design of a data structure for implementing the following operations. The operation update(j, x) has the effect $A[j] \leftarrow A[j] + x$ (where $1 \leq j \leq N$), and the query operation sum(j) returns the partial sum $\sum_{i=1}^{j} A[i]$. We refer to N as the *size* of the partial sum problem. Our interest centers upon the optimal efficiency with which sequences of such operations can be performed, and we derive new upper and lower bounds.

In this paper we investigate complexity relative to the *semigroup* model of computation: the array A can store values from an arbitrary commutative semigroup, and the implementations of the update and sum operations must perform correctly irrespective of the particular choice of the semigroup. In particular, the implementations are not permitted to utilize the subtraction operation. The model assumes the availability of memory registers $z_1, z_2, \ldots$ which store semigroup values and permits operations of the form $z_i \leftarrow z_j + z_k$. In this setting an $O(\log N)$ upper bound is readily established, and Yao [8] has shown that $\Omega(\log N / \log \log N)$ is an inherent lower bound for the amortized complexity of the operations (worst case). Here we consider average case complexity relative to a large class of probability distributions, and as a by-product of this approach, we are able to improve upon Yao's result, deriving an $\Omega(\log N)$ lower bound for the amortized complexity, even when the operations are performed off-line. We remark that with a slight change in the definition of the partial sum problem, namely, by redefining the operation update(j, x) to have the effect $A[j] \leftarrow x$, we find that matching $\theta(\log N)$ upper and lower bounds have been previously established for this problem [3]. This latter definition of the update operation is more readily exploited in a lower bound argument. In particular, in the semigroup

*Received by the editors September 6, 1995; accepted for publication (in revised form) October 15, 1996; published electronically May 15, 1998. A preliminary version of this paper appeared under the same title in *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, 1993, pp. 480–485.

http://www.siam.org/journals/sicomp/28-1/29159.html

†Intrinsa Corporation, 444 Castro St, Suite 130, Mountain View, CA 94041. This research was supported by NSF grant NSF-STC-91-19999.

‡Department of Computer Science, Rutgers University, Hill Center, New Brunswick, NJ 08903 (fredman@cs.rutgers.edu). This research was supported by NSF grant CCR-9008072.

setting all structural quantities that depend on the old value of $A[j]$ are subsequently unusable after the current update is performed.

Our average case analysis concerns sequences of independently selected operations in which the operation update(j, x) is selected with probability p_j to be the next operation in the sequence, and the operation sum(j) is selected with probability q_j to be the next operation in the sequence, where $\sum_{j=1}^{N} p_j + \sum_{j=1}^{N} q_j = 1$. We are able to approximately characterize the optimal expected cost of implementing such sequences in terms of optimal binary trees relative to a type of weighting scheme, defining the notion of *biweighted* binary tree.

In the following section we define the notion of biweighted binary tree and demonstrate the connection between biweighted binary trees and data structures for the partial sum problem. We also present a heuristic for constructing approximately optimal biweighted binary trees. This heuristic serves to motivate the lower bound theorem presented in section 3.

In what remains of this section we describe a class of data structures which is natural with respect to the partial sum problem and which has provided a framework for some previous work. These structures also provide a starting point for the current analysis. Consider the obvious $O(\log N)$ data structure for the partial sum problem which consists of a balanced binary tree with N leaves labeled from 1 to N in order from left to right. The value in $A[i]$ is stored in leaf i, and stored in each internal node η is the sum of the values that are stored in the leaves of the subtree of η. The operation update(j, x) is performed by incrementing by x the values stored in the nodes along the path from leaf j to the root. The operation sum(j) is performed by summing the values in a minimal collection of nodes whose subtrees exactly cover the first j leaves. This data structure can be viewed as a particular instance of (and motivates) the following class of structures. Let $z_1, z_2, \ldots$ designate memory registers that store values from the space pertinent to the array A. A data structure belonging to our class consists of two sequences $U_1, \ldots, U_N$ and $R_1, \ldots, R_N$, where the U_i and the R_i are individual subsets of the memory registers $z_1, z_2, \ldots$. The task update(j, x) is performed by adding x to each register in the set U_j, and sum(j) is performed by summing the registers in the set R_j. The sizes of the U_j and R_j sets reflect the complexity of our operations, and it is readily demonstrated [4] that to correctly implement the partial sum problem, it is both necessary and sufficient that the following condition be satisfied:

$$|U_i \cap R_j| = \begin{cases} 1 & \text{if } i \leq j, \\ 0 & \text{otherwise.} \end{cases} \tag{1.1}$$

We refer to data structures for the partial sum problem that fall within this framework as *U-R systems*.

By imposing size restrictions on the U_i subsets (or the R_i subsets) it is possible to investigate update time versus query time tradeoffs within this class of data structures, as investigated in [1]. The investigation of such tradeoffs is motivated by circumstances in which, say, update operations are more frequently performed than query operations. In this paper we address directly those circumstances under which the operations are known to be requested with varying frequencies. Thus, if update(j, x) is performed with probability p_j and sum(j) is performed with probability q_j, then the expected cost of an operation is given by

$$C(\bar{p}, \bar{q}) = \sum_{j=1}^{N} (p_j |U_j| + q_j |R_j|), \tag{1.2}$$

where $\bar{p}$ and $\bar{q}$ represent the p_j and q_j probability sequences, respectively. We assume throughout this paper that all of the p_j and q_j probabilities are strictly positive; otherwise we can effectively reduce the size N of the array A. In the sequel we solve the problem of minimizing the quantity $C(\bar{p}, \bar{q})$ subject to the condition (1.1). Let $C_{min}(\bar{p}, \bar{q})$ denote this minimum value for $C(\bar{p}, \bar{q})$. It is shown in section 3 that $C_{min}(\bar{p}, \bar{q})$ approximates (to within a constant factor) the optimal cost of doing the operations in the general semigroup model of computation, extending beyond data structures that fall within the framework of U-R systems. Moreover, in terms of expected cost, this lower bound extends to off-line processing.

2. Biweighted trees. We proceed to define a family of valid data structures for the partial sum problem that fall within the U-R system framework. Let T be a binary tree with N nodes. We assume that the nodes of T have zero, one, or two children, and that T is ordered, so that each child of a given node is designated as a left child or right child. We also number the nodes of T from 1 to N in order from left to right. Now in terms of T we define a U-R system for the partial sum problem of size N as follows.

Let η be the jth node of T, and let $\eta_0 (= \eta), \eta_1, \ldots, \eta_h$, $h \geq 0$, be the nodes along the path from η to the root η_h of T. Then U_j consists of η and for $i > 0$ those η_i such that η_{i-1} is the left child of η_i; and R_j consists of η and for $i > 0$ those η_i such that η_{i-1} is the right child of η_i. Now let η_{ij} denote the lowest common ancestor of the ith and jth nodes of T. It is easily checked that

$$U_i \cap R_j = \begin{cases} \eta_{ij} & \text{if } i \leq j\,, \\ \phi & \text{otherwise.} \end{cases} \tag{2.1}$$

The condition (1.1) directly follows from (2.1) and therefore our construction yields a valid data structure for the partial sum problem. We let $\Delta(T)$ denote this data structure.

Next, we interpret the quantity $C(\bar{p}, \bar{q})$, the expected cost of an operation under the distribution $(\bar{p}, \bar{q})$, when implemented using $\Delta(T)$. Associate with the ith node of T the pair of probabilities (p_i, q_i), $1 \leq i \leq N$. Relative to these associated probabilities we then define the *biweighted* path length of T to be

$$W(T, \bar{p}, \bar{q}) = \sum_{i=1}^{N} (p_i \ell_i + q_i r_i)\,,$$

where ℓ_i is the number of left links on the path from the root of T to the ith node and r_i is the number of right links on this same path. Because $\ell_i = |U_i| - 1$ and $r_i = |R_i| - 1$ it follows immediately that

$$C(\bar{p}, \bar{q}) = W(T, \bar{p}, \bar{q}) + 1\,. \tag{2.2}$$

Note also that $W(T, \bar{p}, \bar{q})$ defines (one-half of) the usual weighted path length of T when $\bar{p} = \bar{q}$.

Equation (2.2) suggests the possibility that a good data structure for the partial sum problem, relative to the distribution $(\bar{p}, \bar{q})$, is given by $\Delta(T_{\bar{p},\bar{q}})$, where $T_{\bar{p},\bar{q}}$ is chosen to have minimum biweighted path length $W(T, \bar{p}, \bar{q})$. The following theorem addresses this possibility relative to U-R systems.

THEOREM 2.1. *Relative to the distribution $(\bar{p}, \bar{q})$ with strictly positive probabilities, an optimal U-R system for the partial sum problem is given by $\Delta(T_{\bar{p},\bar{q}})$. In other*

words,

$$C_{min}(\bar{p},\bar{q}) \;=\; W(T_{\bar{p},\bar{q}}\,,\,\bar{p}\,,\,\bar{q}) \;+\; 1\,.$$

Proof of Theorem 2.1. We relax the requirement that the weights $(\bar{p},\bar{q})$ sum to 1 and establish that

$$C_{min}(\bar{p},\bar{q}) \;=\; W(T_{\bar{p},\bar{q}}\,,\,\bar{p}\,,\,\bar{q}) \;+\; \sum_{j=1}^{N}(p_j+q_j)$$

from which Theorem 2.1 follows as an immediate consequence. (We use equation (1.2) to define $C(\bar{p},\bar{q})$ for arbitrary positive weights $(\bar{p},\bar{q})$.) The above equality is established once we show that we can construct, given an arbitrary U-R system, a tree T such that

$$C(\bar{p},\bar{q}) \;\geq\; W(T,\bar{p},\bar{q}) \;+\; \sum_{j=1}^{N}(p_j+q_j)\,. \tag{2.3}$$

Now given a U-R system $(\bar{U},\bar{R})$ consisting of sequences $\bar{U} \;=\; U_1\,,\ldots,\,U_N$ and $\bar{R} \;=\; R_1\,,\ldots,\,R_N$, for some integer $k\in[1,N]$ our construction below yields two U-R systems $(\bar{U}^\ell,\bar{R}^\ell)$ and $(\bar{U}^r,\bar{R}^r)$ for arrays of lengths $k-1$ and $N-k$, respectively, such that

$$|U_i^\ell| \;\leq\; |U_i|-1 \text{ and } |R_i^\ell| \;=\; |R_i| \tag{2.4}$$

for $1\leq i\leq k-1$; and

$$|U_i^r| \;=\; |U_{i+k}| \text{ and } |R_i^r| \;\leq\; |R_{i+k}|-1 \tag{2.5}$$

for $1\leq i\leq N-k$.

Assume for the moment that an integer k and U-R systems $(\bar{U}^\ell,\bar{R}^\ell)$ and $(\bar{U}^r,\bar{R}^r)$ satisfying (2.4) and (2.5) can be constructed. Let $(\bar{p}_\ell,\bar{q}_\ell)$ designate the first $k-1$ pairs (p_i,q_i) of the weights $(\bar{p},\bar{q})$ and let $(\bar{p}_r,\bar{q}_r)$ designate the last $N-k$ pairs of these weights. By assigning the weights $(\bar{p}_\ell,\bar{q}_\ell)$ to the system $(\bar{U}^\ell,\bar{R}^\ell)$, assigning the weights $(\bar{p}_r,\bar{q}_r)$ to the system $(\bar{U}^r,\bar{R}^r)$, and noting that $|U_k|,|R_k| \;\geq\; 1$, we conclude from (2.4) and (2.5) that

$$\begin{aligned} &C(\bar{p},\bar{q}) \\ &\quad\geq C(\bar{p}_\ell,\bar{q}_\ell)+C(\bar{p}_r,\bar{q}_r) \\ &\qquad+\sum_{j=1}^{k-1}p_j+\sum_{j=k+1}^{N}q_j \;+\; (p_k+q_k)\,. \end{aligned} \tag{2.6}$$

We construct the tree T satisfying (2.3) recursively as follows. The root of T is the kth node, where k defines the partition of $(\bar{U},\bar{R})$ as described above. The left subtree T_{left} of T is obtained recursively from $(\bar{U}^\ell,\bar{R}^\ell)$, and the right subtree T_{right} is obtained recursively from $(\bar{U}^r,\bar{R}^r)$. Upon assigning the weights $(\bar{p},\bar{q})$ to the nodes to T, so that $(\bar{p}_\ell,\bar{q}_\ell)$ designates the weights assigned to T_{left} and $(\bar{p}_r,\bar{q}_r)$ designates the weights assigned to T_{right}, we find that

$$\begin{aligned} W(T,\bar{p},\bar{q}) = W(T_\ell,\bar{p}_\ell,\bar{q}_\ell) \;+\; W(T_r,\bar{p}_r,\bar{q}_r) \\ +\sum_{j=1}^{k-1}p_j \;+\; \sum_{j=k+1}^{N}q_j\,. \end{aligned} \tag{2.7}$$

It follows by induction, using (2.7) and (2.6), that (2.3) is satisfied.

We complete the proof of Theorem 2.1 by giving the construction satisfying (2.4) and (2.5). Let

$$\Gamma = U_1 \cup \cdots \cup U_N \cup R_1 \cup \cdots \cup R_N .$$

For each element x in Γ, let

$$\text{u-extent}(x) = \max\{j | x \in U_j\} .$$

Observe that for any set R_j condition (1.1) implies that

$$\text{u-extent}(x) \leq j \text{ for each } x \text{ in } R_j . \tag{2.8}$$

For each set R_j let

$$\text{u-extent}(R_j) = \min\{\text{u-extent}(x) | x \in R_j\} .$$

Finally, let

$$k = \max\{\text{u-extent}(R_j) | 1 \leq j \leq N\} ,$$

and choose k' so that $k = \text{u-extent}(R_{k'})$. From (2.8) we have that $k \leq k'$. Also, for each $x \in R_{k'}$, $\text{u-extent}(x) \geq k$. Because $U_j \cap R_{k'} \neq \phi$ for $j < k'$, it follows that each set U_j with $j < k'$ contains an element x_j such that $\text{u-extent}(x_j) \geq k$. On the other hand (2.8) asserts that for each $j < k$ the set R_j contains no element x with $\text{u-extent}(x) \geq k$. Thus we can find in each set U_j with $j < k$ an element x_j (with $\text{u-extent}(x_j) \geq k$) not contained in any set R_h with $h < k$.

Accordingly, we construct the system $(\bar{U}^\ell, \bar{R}^\ell)$ by setting $U_j^\ell = U_j - \{x_j\}$, where x_j is some element in U_j with $\text{u-extent}(x_j) \geq k$, and setting $R_j^\ell = R_j$, $1 \leq j \leq k-1$. Certainly (2.4) is satisfied and moreover the system $(\bar{U}^\ell, \bar{R}^\ell)$ inherits condition (1.1) from the system $(\bar{U}, \bar{R})$ since the deleted elements x_j do not appear in any of the intersections $U_h \cap R_i$ when $h, i \leq k-1$.

Now every set R_j contains an element x_j with $\text{u-extent}(x_j) \leq k$. For $h > k$, none of these elements x_j appear within the sets U_h by definition of $\text{u-extent}(x)$. Thus, if we set $U_j^r = U_{j+k}$ and $R_j^r = R_{j+k} - \{x_j\}$, for $1 \leq j \leq N-k$, then the resulting system $(\bar{U}^r, \bar{R}^r)$ satisfies (2.5), and, as before, condition (1.1) is satisfied. □

Equation (2.7) suggests that the following balancing heuristic is reasonable to consider in constructing a tree T with near optimal biweighted path length: choose the root to be the kth node where k is chosen to equalize the sums $\sum_{j=1}^{k-1} p_j$ and $\sum_{j=k+1}^{N} q_j$ and construct the subtrees recursively. More precisely we define T_{BAL} so that its root is the kth node, where k is the least integer satisfying $\sum_{j=1}^{k-1} p_j \geq \sum_{j=k+1}^{N} q_j$, and whose subtrees are recursively constructed.

THEOREM 2.2. *Let T_{BAL} be defined as above. Then $W(T_{BAL}, \bar{p}, \bar{q}) \leq 2 \cdot W(T_{\bar{p},\bar{q}}, \bar{p}, \bar{q}) + 1$.*

Comment. Theorem 2.2 justifies using a balancing heuristic for constructing near optimal U-R systems. However, in view of Theorem 2.1 it is easy to construct truly optimal U-R systems by using dynamic programming to construct a tree having optimal biweighted path length. (The recurrence that forms the basis for the dynamic programming algorithm is obtained by selecting k in equation (2.7) to minimize the right-hand side.) But the true significance of Theorem 2.2 is revealed in the next

section when we consider the full class data structures within the semigroup model of computation. As a means for estimating complexity, T_{BAL} is a considerably more convenient object to reason about than the optimal tree, $T_{\bar{p},\bar{q}}$.

Proof of Theorem 2.2. Let η be any node in T_{BAL}, say, the jth node, let $T_{\eta,left}$ be the left subtree descending from η, and let $T_{\eta,right}$ be the right subtree descending from η. Let P_{sum} denote the sum of the p_i values associated with the nodes in $T_{\eta,left}$, and let Q_{sum} be the sum of the q_i values associated with the nodes in $T_{\eta,right}$. Now let γ be the lowest common ancestor in $T_{\bar{p},\bar{q}}$ of those nodes which comprise the subtree descending from η within T_{BAL}. Then either

(i) the left subtree $T_{\eta,left}$ descending from η lies to the left of γ in $T_{\bar{p},\bar{q}}$, or

(ii) the right subtree $T_{\eta,right}$ descending from η lies to the right of γ in $T_{\bar{p},\bar{q}}$.

For the purpose of estimating $W(T_{\bar{p},\bar{q}}, \bar{p}, \bar{q})$, in the case (i) we can charge the quantity P_{sum} to the left link of γ and in the case (ii) we can charge the quantity Q_{sum} to the right link of γ. We will demonstrate below that no p_i or q_i value gets charged to the same link of $T_{\bar{p},\bar{q}}$ twice. Assuming this, it follows that $W(T_{\bar{p},\bar{q}}, \bar{p}, \bar{q}) \geq$ the sum over all nodes in T_{BAL} of the quantities $\min(P_{sum}, Q_{sum}) = S_{BAL}$ (say). By definition of T_{BAL} we have that $P_{sum} \geq Q_{sum}$ and $Q_{sum} > P_{sum} - q_j - p_{j-1}$. It follows that

$$\min(P_{sum}, Q_{sum}) = Q_{sum} > \frac{1}{2}(P_{sum} + Q_{sum} - q_j - p_{j-1})\,. \tag{2.9}$$

Applying equation (2.7) to T_{BAL}, we note that $W(T_{BAL}, \bar{p}, \bar{q})$ equals the sum over all nodes in T_{BAL} of the quantities $P_{sum} + Q_{sum}$. Using the inquality (2.9) to compare the sums over the nodes of T_{BAL} that give, respectively, S_{BAL} and $W(T_{BAL}, \bar{p}, \bar{q})$, we conclude that $S_{BAL} \geq \frac{1}{2}W(T_{BAL}, \bar{p}, \bar{q}) - \frac{1}{2}$. Combining these inequalities for S_{BAL} yields the inequality in the statement of our theorem.

To complete the proof, we proceed to show that no p_i or q_i value gets charged to the same link of $T_{\bar{p},\bar{q}}$ twice. Suppose to the contrary that p_i (say) gets charged twice to the left link of γ, a node in $T_{\bar{p},\bar{q}}$. Now the first charge is through some node η_1 of T_{BAL} and the second charge is through some node η_2 of T_{BAL}. With respect to T_{BAL} both η_1 and η_2 are ancestors of the node τ with which p_i is associated and τ lies in the left subtrees of both η_1 and η_2. Therefore, the nearer ancestor η_1 (say) lies in the left subtree of η_2. Also in $T_{\bar{p},\bar{q}}$ we have that the nodes belonging (in T_{BAL}) to the left subtree of η_2 all lie to the left of γ. Consequently, the lowest common ancestor γ' (in $T_{\bar{p},\bar{q}}$) of the nodes in the subtree of η_1 (with respect to T_{BAL}) must lie to the left of γ. But the charge through η_1 must be assessed to a link of γ', contrary to the assumption. □

3. Unrestricted semigroup lower bound. In this section we generalize the lower bounds established in the preceding section for U-R systems to all computations that fall within the semigroup model of computation. Our bounds also apply to off-line computations. Our method of attack is to combine the biweighted tree perspective with the method of Wilber [7] used to analyze the off-line complexity of performing search tree operations intermixed with appropriately timed node rotations. Whereas the two problems, search tree operations versus partial sum operations, are very different, similar treatments succeed with both problems.

In the spirit of Wilber [7] we define a *lower bound tree* Υ that will be used to estimate the expected amount of work required to perform a random sequence of update and sum operations. For the partial sum problem of size N, with operations being requested randomly and independently in accordance with the distribution $(\bar{p}, \bar{q})$, we

choose Υ to be the tree T_{BAL} from the preceding section. Our argument proceeds in accordance with the following outline.

Part A. Given a specific sequence s of update and sum operations, we define the *score* of Υ with respect to the sequence s.

Part B. We show that the score defined in Part A provides a lower bound for the number of semigroup operations required to implement the sequence s off-line.

Part C. We estimate the expected value of the score defined in Part A for a random sequence s of operations.

Part A. Let η be the kth node in Υ, and let a and b be the smallest and largest nodes in the subtree descending from η. Now remove from the sequence s all update(j, x) and sum(j) operations with j outside of the interval $[a, b]$, remove all operations update(j, x) with $j > k$, and remove all operations sum(j) with $j < k$. Let s' be the resulting sequence. Then we define $\lambda(\eta, s)$ to be the number of pairs of consecutive terms in s' such that first term of the pair is an update operation and the second term is a sum operation. (Observe that $\lambda(\eta, s) = \lambda(\eta, s')$.) Finally, the score of Υ with respect to s is given by

$$\Lambda(\Upsilon, s) = \sum_{\eta \in \Upsilon} \lambda(\eta, s).$$

Part B. We choose as our semigroup the set of linear expressions over a set consisting of an infinite number of indeterminates. Assume that initially the quantity stored in $A[j]$ is given by x_j, where the x_j are distinct indeterminates. Also, assume that whenever an update(j, x) operation is performed, the quantity x is a newly introduced indeterminate. We associate with each indeterminate x the index j designating the array position $A[j]$ to which x contributes, and we refer to j as the *index of* x.

Our proof that $\Lambda(\Upsilon, s)$ provides a lower bound for the required number of semigroup operations proceeds by induction. Assume that the root η of Υ is the kth node in the tree. Consider an implementation σ of the sequence s of update and sum operations. If we eliminate from σ all semigroup operations involving indeterminates with index $\geq k$, then the remaining operations, σ_{left}, implement the subsequence s'' of update and sum operations in s whose array indices correspond to nodes in the left subtree Υ_{left} of Υ. Similarly, if we set to 0 all indeterminates with index $\leq k$ and then eliminate from σ any semigroup operations involving the addition of 0, then the remaining operations, σ_{right}, implement the subsequence s''' of update and sum operations whose array indices correspond to nodes in the right subtree Υ_{right} of Υ, treating the array A as though its index ranges over the interval $[k+1, N]$.

Now observe that

$$\begin{aligned} &\Lambda(\Upsilon, s) \\ &= \Lambda(\Upsilon_{left}, s'') + \Lambda(\Upsilon_{right}, s''') + \lambda(\eta, s'), \end{aligned} \tag{3.1}$$

where s' is defined in Part A. We easily see that the two sequences of semigroup operations σ_{left} and σ_{right} reflect disjoint semigroup operations in σ, and by the induction hypothesis, the number of operations in σ_{left} and σ_{right} dominate the first two terms on the right-hand side of equation (3.1). Part B is completed once we show that σ includes $\lambda(\eta, s')$ further semigroup operations.

Consider a pair of consecutive operations update(j', x), sum(j) in s'. The indeterminate x must be combined with x_j, the indeterminate originally stored in $A[j]$, by executing a semigroup operation. We may conclude, therefore, that there exists in

σ an operation $\alpha: z_p \leftarrow z_q + z_r$ contributing to the computation of sum(j), in which x gets combined with some indeterminate whose index is $\geq k$, and where x is not already found to be so combined among the operands on the right. This operation does not contribute to σ_{left} by definition. Likewise, it does not contribute to σ_{right} since the index j' of x does not exceed k (so that the operand in α containing x contains only indeterminates with index $\leq k$ and thus gets set to 0 in the process defining σ_{right}). Last, we may assume that the operation α excludes any indeterminate x' that follows x in s', since x must be produced in the sum(j) operation without the presence of x'. (In other words, if we remove from σ all semigroup operations that involve x', then the remaining operations would suffice to generate the required value for sum(j).) Thus, we can account for $\lambda(\eta, s')$ such operations α.

Part C. Let η be a node in $T_{BAL} = \Upsilon$, and assume that η is the kth node of Υ. We compute the expected value of $\lambda(\eta, s)$. Assume that the sequence s is of length m. Then the expected number of terms in s' is given by $m(P_{sum}+Q_{sum}+p_k+q_k) = mR$ (say). The conditional probability y_1 that a given term in s' is an update operation is given by $(P_{sum}+p_k)/R$, and the conditional probability y_2 that this term is a sum operation is given by $(Q_{sum}+q_k)/R$. By our definition of T_{BAL} we have $P_{sum} \geq Q_{sum}$ and $Q_{sum}+q_k > P_{sum}-p_{k-1}$. Therefore, $P_{sum}+p_k \geq \frac{1}{2}(R-q_k)$, and $Q_{sum}+q_k \geq \frac{1}{2}(R-p_{k-1}-p_k)$. We conclude that the expected number of pairs contributing to $\lambda(\eta, s)$ is given by $(mR-1)\cdot y_1 \cdot y_2 \geq \frac{mR}{4}(1-q_k/R)(1-(p_{k-1}+p_k)/R) - 1 \geq \frac{m}{4}(R-p_{k-1}-p_k-q_k) - 1$. Summing over all nodes η we conclude that $\Lambda(\Upsilon, s) = \Omega(m(W(T_{BAL}, \bar{p}, \bar{q}) - 1)) = \Omega(m(W(T_{\bar{p},\bar{q}}, \bar{p}, \bar{q}) - 1))$.

Summarizing the above discussion we have established the following theorem.

THEOREM 3.1. *For a random sequence s of* update(j, x) *and* sum(j) *operations, the optimal expected cost of executing the operations in s is given by $\theta(W(T_{\bar{p},\bar{q}}, \bar{p}, \bar{q}))$ per operation. In other words, U-R systems derived from optimal biweighted trees are near optimal within the semigroup model of computation. Moreover, the lower bound applies to off-line computations.*

Remark. Consider the special case $p_i = q_i = \frac{1}{2N}$ for $1 \leq i \leq N$. It is clear in this case that we may choose the optimal tree $T_{\bar{p},\bar{q}}$ to be the fully balanced tree with N nodes. Since $W(T_{\bar{p},\bar{q}}, \bar{p}, \bar{q}) = \Omega(\log N)$ in this instance, we conclude that the optimal off-line complexity of the partial sum problem is $\Omega(\log N)$ per operation. This improves upon the $\Omega(\log N/\log\log N)$ lower bound that was established for on-line computations [8].

4. Discussion. The problem of maintaining partial sums is perhaps the least complicated example of a nontrivial range or geometric query problem, and this problem is not yet fully understood. The partial sum problem is now well understood in the semigroup model of computation, but much remains to be resolved for less restrictive computational models. For example, in the group model of computation, wherein the space of array values is given by a group so that subtraction is available, a gap remains that separates the best upper and lower bounds for the worst-case amortized complexities. The best upper bound is $O(\log N)$ and the best lower bound is $\Omega(\log N/\log\log N)$ [5]. It seems reasonable to conjecture, however, that the truth will favor the upper bound. In particular, the notion of U-R systems generalizes for the group model; the sets U_i remain as before, and the sets R_i are replaced with linear expressions over the memory registers with integer coefficients. In this setting $\log N$ bounds are known [4] (both upper and lower), including even a determination of the constant factor. The biweighted tree treatment, however, does not seem to easily extend to cover the group case. In particular, lower bounds seem difficult.

Similarly, in the cell-probe model of computation with polylog(N) word size and with the space of array values given by Z_N, the integers mod N, the same gap exists between the best upper and lower bounds [5]. On the other hand, if the space of array values is given by Z_2, then matching upper and lower bounds of size $\theta(\log N/\log\log N)$ exist in the cell probe model [5] (see also [2]).

The nature of the possible tradeoffs between query versus update time in these less restrictive models is likewise not well understood. There is also an absence of off-line lower bounds in these other models.

As an object of independent interest, the biweighted trees pose some open questions. In particular, there seems to be no easy analogue to the entropy function, which is used to estimate the weighted path length of optimal trees in the usual setting. Complicating the situation is the fact that the function $C_{min}(\bar{p},\bar{q})$ highly depends on the ordering of the probabilities, even when the p_i and q_i values remain coupled. Simple examples exist which demonstrate the phenomenon that permuting the order of the probabilities can cause $C_{min}(\bar{p},\bar{q})$ to vary over the range from $O(1)$ and $O(\log N)$.

Finally, it is interesting to note that given any probability distribution with strictly positive probabilities, it is possible to explicitly exhibit arbitrarily large sequences of update and sum operations that have the corresponding proportions of the update and sum operations and satisfy the bounds of Theorem 3.1. The sequence is a generalization of the bit reversal sequence [7, 8]. The details of the construction can be found in [6].

Acknowledgment. The first author wishes to thank Vivek Gore for many helpful discussions.

REFERENCES

[1] W. A. Burkhard, M. L. Fredman, and D. J. Kleitman, *Inherent complexity trade–offs for range query problems*, Theoret. Comput. Sci., 16 (1981), pp. 279–290.

[2] P. Dietz, *Optimal algorithms for list indexing and subset rank*, Algorithms and Data Structures: Workshop WADS '89, Ottawa, Canada, 1989.

[3] M. L. Fredman, *A lower bound on the complexity of orthogonal range queries*, J. Assoc. Comput. Mach., 28 (1981), pp. 696–705.

[4] M. L. Fredman, *The complexity of maintaining an array and computing its partial sums*, J. Assoc. Comput. Mach., 29 (1982), pp. 250–260.

[5] M. L. Fredman and M. E. Saks, *The cell probe complexity of dynamic data structures*, in Proceedings of 21st ACM Symposium on Theory of Computing, Seattle, WA, 1989, pp. 345–354.

[6] H. Hampapuram, *The Partial Sum Problem: Tight Upper and Lower Bounds*, Ph.D. thesis, Department of Computer Science, Rutgers—The State University of New Jersey, New Brunswick, 1994.

[7] R. Wilber, *Lower bounds for accessing binary trees with rotations*, SIAM J. Comput., 18 (1989), pp. 56–67.

[8] A. C. Yao, *On the complexity of maintaining partial sums*, SIAM J. Comput., 14 (1985), pp. 277–288.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 10–26

DYNAMIC 2-CONNECTIVITY WITH BACKTRACKING*

JOHANNES A. LA POUTRÉ† AND JEFFERY WESTBROOK‡

Abstract. We give algorithms and data structures that maintain the 2-edge and 2-vertex-connected components of a graph under insertions and deletions of edges and vertices, where deletions occur in a backtracking fashion (i.e., deletions undo the insertions in the reverse order). Our algorithms run in $\Theta(\log n)$ worst-case time per operation and use $\Theta(n)$ space, where n is the number of vertices. Using our data structure we can answer queries, which ask whether vertices u and v belong to the same 2-connected component, in $\Theta(\log n)$ worst-case time.

Key words. dynamic graph algorithms, backtracking

AMS subject classifications. 68Q20, 68Q25

PII. S0097539794272582

1. Introduction.

Dynamic graph problems have been studied extensively in the last several years. Roughly speaking, the research has concentrated on two categories of dynamic graphs, viz., *partially dynamic* or *incremental* graphs, which are graphs that grow on line by the insertion of vertices and edges, and *fully dynamic* graphs, which are subject to arbitrary insertion and deletion of edges and vertices.

A number of different problems on incremental and fully dynamic graphs have been studied, including 2- and 3-edge connectivity, 2- and 3-vertex connectivity, spanning trees, and planarity testing [1], [5], [7], [8], [9], [10], [12], [13], [22], [18], [19], [20], [21], [26], [27], [30], [31], [33]. Deterministic algorithms for incremental 2-edge and 2-vertex connectivity running in $\Theta(\alpha(m,n))$ amortized time per operation, where m is the maximum number of edges and n the maximum number of vertices, are described in [18], [33]. Those algorithms require $\Theta(n)$ time per operation in the worst case. A fully dynamic deterministic algorithm for 2-edge connectivity running in $O(\sqrt{n})$ time is given by Eppstein et al. [7], and an $O(\sqrt{n}\log n)$ time algorithm for fully dynamic 2-vertex connectivity is given by Rauch [27]. (Again, m is the maximum number of edges and n the maximum number of vertices.)

Thus, there is a substantial gap in deterministic time complexity between the incremental and fully dynamic problems.[1] A tantalizing question is whether we can obtain much better time bounds than those for the fully dynamic problems by putting restrictions on the deletions of edges. A natural and useful restriction is to limit

*Received by the editors August 8, 1994; accepted for publication (in revised form) October 28, 1996; published electronically June 15, 1998.
http://www.siam.org/journals/sicomp/28-1/27258.html

†Department of Computer Science, Princeton University, Princeton NJ 08540 and Department of Computer Science, Utrecht University, 3508 TB Utrecht, The Netherlands. At Princeton University, the research was supported by a NATO Science Fellowship awarded by NWO (the Netherlands Organization for Scientific Research) and DIMACS (Center for Discrete Mathematics and Theoretical Computer Science - NSF-STC88-09648). At Utrecht University, the research of the author has been made possible by a fellowship of the Royal Netherlands Academy of Sciences and Arts (KNAW). Current address: Department of Computer Science, Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands (han@wi.leidenuniv.nl).

‡AT&T Labs-Research, Florham Park, NJ 07932 (westbrook@att.com). This research was done while the author was at the Department of Computer Science, Yale University, and was partially supported by National Science Foundation grant CCR-9009753.

[1]Recently randomization has been used to derive polylogarithmic time algorithms for several fully dynamic graph problems [17].

deletions to a backtracking *Undo*, which removes the most recently added edge not yet removed.

Dynamic backtracking problems appear to be an important research area for several reasons. The backtracking operation *Undo* is a common feature of interactive software systems. Also, backtracking search is a common search strategy in many logic and artificial intelligence applications. For example, maintaining 2-vertex-connected components (with backtracking) has been proposed as a way to improve search in Prolog [25]. Furthermore, dynamic graphs with backtracking suffice for many interactive system applications like, e.g., CAD/CAM systems and VLSI layout. Maintaining 2-vertex-connected components could potentially be used for problems in reliable network design, or for designing VLSI layouts.

Previous work on backtracking addressed the *Union-Find* problem, in which the standard disjoint set operations of *Union* and *Find* are augmented by the backtracking operation *Deunion*. (A *Deunion* undoes the most recent *Union* that has not been undone.) *Union-Find* with backtracking is a central problem in the implementation of unification and backtracking search in the logic programming language Prolog. Mannila and Ukkonen [23], [24] first formalized and studied this problem and proposed several algorithms which Westbrook and Tarjan [32] subsequently analyzed: each operation can be performed in $\Theta(\log n/\log\log n)$ amortized time. Blum [3] gave a data structure for *Union-Find* without backtracking that runs in $\Theta(\log n/\log\log n)$ worst-case time per operation. As observed in [32], Blum's data structure can be adapted to handle *Deunions* in the same time bound. Variants and extensions of this problem are studied in [11], [14].

In [30] it is observed that backtracking graph connectivity could be solved in $\Theta(\log n/\log\log n)$ time by a straightforward application of the backtracking *Union-Find* algorithm (as incremental graph connectivity can be solved by straightforwardly applying standard *Union-Find*). Tamassia [29] gave an algorithm for a hierarchical embedding problem related to VLSI design that is essentially an algorithm with *Undo* operations for maintaining an embedded planar *st*-orientable graph (such a graph is 2-connected if one edge (s,t) is added) under a restricted set of modifications; it thus gives a better time complexity than its fully dynamic counterpart for general, unrestricted embedded planar graphs [16] but only by a factor $\log n$.

In this paper, we consider maintaining the 2-edge and 2-vertex connectivity relations in dynamic graphs with backtracking, i.e., graphs subject to the modifications *Insert Vertex*(), which adds a new, isolated vertex to the graph; *Insert Edge*(u,v), which inserts a new edge between vertices u and v; and *Undo*, which undoes the effects of the most recent insertion not yet undone. We give algorithms and data structures that maintain a decomposition of a dynamic backtracking graph into its 2-edge and 2-vertex-connected components throughout any sequence of backtracking operations. Using our data structure we can answer *Test*(u,v) queries, which ask whether vertices u and v belong to the same 2-connected component, in $\Theta(\log n)$ worst-case time. Our algorithms run in worst-case time $\Theta(\log n)$ per operation and use $\Theta(n)$ space, where n is the current number of vertices ("existing") in the graph.

To our knowledge, the algorithms in this paper are the first nontrivial results for dynamic backtracking graph problems that yield a substantial improvement in time complexity over their fully dynamic counterparts. Our algorithms also solve the corresponding incremental problems in logarithmic worst-case time, improving the $\Theta(n)$ worst-case bounds on the algorithms given in [18], [33]. In fact, we present our results by first describing new incremental algorithms and then augmenting them to

support backtracking.

For comparison, we mention several alternate approaches to solving the backtracking problem. The simplest one is to push each edge on a stack as it is added, popping the stack for each *Undo*. A test query is answered by copying the edges on the stack, constructing the graph, and running a standard biconnectivity algorithm [4]. Updates require $\Theta(1)$ time in the worst case, queries take $\Theta(m)$ time, and the space required is $\Theta(m)$, where m is the maximum number of edges ever in the graph.

Another possibility is to use the techniques of *persistence* [6]. A normal data structure is *ephemeral* in the sense that after an update the old version is destroyed and replaced by the updated version. Using the techniques of Driscoll et al. [6], a pointer-based data structure in which all nodes have constant bounded in-degree and out-degree can be made *fully persistent*. In a fully persistent data structure all versions of the data structure can be accessed, and any old version can be updated to yield a new version. The amortized time per operation of the persistent data structure is equal to the worst-case time per operation of the underlying ephemeral data structure, and the space requirement of the persistent data structure is equal to the total number of pointer changes made in all versions of the ephemeral data structure. Applying persistence to a data structure for the incremental 2 connectivity problem gives a solution to the backtracking problem. The data structures of [19], [33] do not have constant bounded in-degree, and the worst-case time per operation is $\Theta(n)$. When we replace nodes with high in-degree with balanced binary trees, persistent versions of these data structures give a backtracking algorithm that runs in $\Theta(n \log n)$ amortized time per operation and requires $\Theta(Mn)$ space, where M is the total number of edge insertion operations. Applying persistence to the incremental data structures developed in this paper gives less dismal results: $\Theta(\log n)^2$ amortized time per operation and $\Theta(n + M \log n)$ space. (The additional factor of $\log n$ again arises from replacing nodes with high in-degree by balanced binary trees.) In contrast, our direct solution to the backtracking problem gives $\Theta(\log n)$ time in the worst-case and only $\Theta(n)$ space.

This paper is organized as follows. In section 3, we present a solution for 2-edge connectivity that runs in $\Theta(\log n)$ time per operation and $\Theta(n)$ space. Although obtaining these bounds for 2-edge connectivity is relatively simple, obtaining $O(\log n)$ bounds for 2-vertex connectivity requires rather more sophisticated data structuring and accounting. These are presented in section 4. There we first give an intermediate and simpler solution that runs in $\Theta(\log^2 n / \log \log n)$ time per operation and then present the $\Theta(\log n)$ solution.

2. Preliminaries.

2.1. Terminology. In this paper, we use the standard graph terminology in Harary [15].

Let $G = (V, E)$ be a graph. A *path* is a sequence of vertices $v_0, v_1, \ldots, v_k$ such that $\{v_i, v_{i+1}\} \in E, 0 \leq i < k$. Vertices v_0 and v_k are *endpoints* of the path; the remaining vertices are *internal*. Two vertices in V are *connected* if there exists a path between them. The connected components of G are the maximal subgraphs of mutually connected vertices.

Let $\{u, v\}$ be an edge of graph G whose removal disconnects the graph. Such an edge is called a *bridge*. The 2-*edge-connected components* of G are the connected components that remain after all bridges are removed. Two vertices are 2-*edge connected* if they belong to the same 2-edge-connected component. If u and v are 2-edge connected, then there are at least two edge disjoint paths between them.

Let u be a vertex whose removal disconnects G. Such a vertex is called a *cutpoint*. Two edges are called 2-*vertex connected* if they lie on a common simple cycle. Thus, 2-vertex connectivity is an equivalence relation on the edge set. The 2-*vertex-connected components* or *blocks* of G are the subgraphs of G induced by the edges in an equivalence class plus their end nodes. Two vertices are 2-*vertex connected* if they belong to the same 2-vertex-connected component. Thus, two vertices are 2-vertex connected if and only if there are at least two vertex-disjoint paths between them, and any two 2-vertex-connected components intersect at most at one cutpoint.

Recall from [33] that for the 2-edge and the 2-vertex connectivity relation, $\Theta(n)$ 2-connected components may be merged in case of an edge insertion, and, similarly, $\Theta(n)$ new components may arise in case of an *Undo* operation. Thus, it is possible to construct a sequence of operations in which each operation changes the number of 2-connected components by $\Theta(n)$. This is in contrast with connected components, where only two components may be joined or separated by an insertion or *Undo* operation.

2.2. Dynamic trees. In [28], Sleator and Tarjan presented their dynamic tree data structure. For later reference, we briefly describe the main features of this data structure.

The data structure maintains a rooted tree with costs on each edge or, alternatively, on each vertex. It performs the following operations (among others) in $\Theta(\log n)$ worst-case time:

find root(u), which returns the root of the tree in which node u is contained;

parent(u), which returns the parent of node u in the tree (if any);

find min(u), which returns the edge of minimum cost on the path from u to the root;

add cost(u, x), which adds value x to the cost of all edges on the path from u to the root;

cut(u), which creates two trees from one by cutting the edge from u to its parent;

link(u, v, x), which combines two trees into one by making u a child of v (it presumes that u is the root of a tree distinct from the tree containing v), where edge (u, v) has cost x;

and *evert*(u), which reroots the tree containing u at u.

If costs are associated with nodes, *add cost*(u, x) adds value x to the cost of all nodes on the path from u to the root, and *find min*(u) returns the node of minimal cost on the path from u to the root.

As observed in [30], maintaining the connectivity relation with backtracking can be performed in $\Theta(\log / \log \log n)$ time per operation and $\Theta(n)$ space by using a *Union-Find* structure with backtracking. We remark that the connectivity relation can also be maintained using dynamic trees in $\Theta(\log n)$ time per operation. For each component we maintain a spanning tree in the obvious way and test whether u and v are in the same component by *find root* operations for u and v. We will not make the maintenance of the connectivity relation explicit in our algorithms for 2-edge and 2-vertex connectivity.

3. 2-edge connectivity. In this section, we describe algorithms and data structures for maintaining the 2-edge connectivity relation in dynamic graphs with backtracking. We first describe a data structure that performs 2-edge connectivity queries and edge insertions in $\Theta(\log n)$ worst-case time, and we then extend this algorithm to handle backtracking.

Let T be a spanning tree of a graph G. An edge $e \in T$ is *covered* if it lies on the fundamental cycle (with respect to T) of some nontree edge $f \in G$.

LEMMA 3.1 (see [10]). *Two vertices are in the same 2-edge-connected component of G if and only if all edges on the path in T between u and v are covered.*

Using this lemma, it is easy to solve the incremental 2-edge connectivity problem in worst-case $\Theta(\log n)$ time per operation, using the variant of the dynamic tree data structure in which cost is attached to the tree edges. A spanning tree is maintained for each component of G and used for testing 2-edge connectivity. Each edge of G is classified as either a spanning tree edge, an *essential* nontree edge, or a *nonessential* nontree edge. The edge is a spanning tree edge if at the time of insertion it connected two previously unconnected components. The edge is an essential edge if at the time of insertion it reduced the number of 2-edge-connected components. Otherwise it is nonessential. At any time, let G' be the subgraph of G consisting of all vertices of G, all spanning tree edges, and all essential nontree edges. Thus, G and G' have the same 2-edge-connected components. The size of G' is $2(n-1)$ because each edge in G' is either a spanning tree edge or an edge that reduced the number of 2-edge-connected components by 1, which can happen at most $n-1$ times. The cost of a tree edge e will be the number of nonspanning tree edges in G' covering tree edge e. The operations on G defined in the introduction are implemented as follows.

Insert Vertex(): Create a new single-node tree and return a pointer to the new node.

Test(u, v): If u and v are in different components, return "no." Otherwise perform *evert*(v) followed by *find min*(u). If the minimum value is zero, return "no," otherwise "yes."

Insert Edge(u, v): If u and v are in different components, do *evert*(u) and perform *link*$(u, v, 0)$, creating a new tree edge e with cost 0. Otherwise, compute *Test*(u, v). If the result is "no," then *evert*(v) and *add cost*$(u, 1)$. Otherwise do nothing.

The correctness of these routines is easily seen by induction on the number of requests. The crucial observation is that the cost of an edge is exactly equal to the number of covering edges in the graph G', and that if there is an edge in G covering edge e, then there is an edge in G' covering e. Each operation runs in worst-case time $\Theta(\log n)$, since each performs a constant number of dynamic tree operations.

So far, we have a data structure for the incremental problem. To support *Undo*, we utilize a *backtrack stack*. If an *Insert Edge*(u, v) or *Insert Vertex*() operation changes the number of components or 2-edge-connected components, then it is essential and G' is augmented accordingly; a new record is pushed on the backtrack stack. The record contains the type of operation performed, the endpoints of the new edge in the case of an edge insertion, the name of the new vertex in the case of a vertex insertion, and a counter initialized to zero. This counter contains the number of nonessential, not yet undone insertions performed after the one described in the record (which is an essential one), and before the essential insertion described in the next record (if any). Thus, if an *Insert Edge*(u, v) operation adds an edge between two vertices that are already in the same 2-edge-connected component, the counter in the top stack-record is simply incremented.

To perform *Undo*, proceed as follows. Examine the counter in the top record on the stack. If it is greater than zero, decrement the counter and terminate. Otherwise pop the top record. If the operation stored in this record is an *Insert Vertex*(), then

delete the appropriate vertex. If the operation is *Insert Edge*(u, v), then do *evert*(v) and *add cost*$(u, -1)$. Perform *find min*(u). If it returns an edge e of cost -1 (meaning e is an edge that, when inserted, connected two previously unconnected components), then *cut*(u) is performed, while e is deleted. We obtain the following theorem.

THEOREM 3.2. *A sequence of* Test(u, v), Insert Vertex(), Insert Edge(u, v), *and* Undo *operations can be performed in* $\Theta(\log n)$ *worst-case time per operation and in* $\Theta(n)$ *space, where* n *is the current number of nodes.*

Proof. It is readily seen that the relation between G and G' is maintained. To show the correctness of the backtracking procedure, it suffices to confirm that an *Undo* performed immediately after an insertion restores the data structure to its condition prior to the insertion. The time bound follows from [28]. The space complexity follows since the size of the stack (i.e., the number of records in it) is bounded by the size of G'. □

4. 2-vertex connectivity. In this section, we describe algorithms and data structures for maintaining the 2-vertex connectivity relation in dynamic graphs with backtracking. As in the previous section, we begin by describing a data structure that performs only 2-vertex test queries and edge insertions in $O((\log n)^2/\log\log n)$ worst-case time. This then serves as a basis for a backtracking algorithm with this time complexity. Subsequently, we present data structures and algorithms to achieve $O(\log n)$ time per operation.

As in the case of 2-edge connectivity, our approach is to maintain a spanning tree of the graph and store information with the vertices and edges of the spanning tree that can be used efficiently to answer test queries. In the case of 2-vertex connectivity, however, there is no simple covering lemma, and our algorithms and data structures are consequently more complex. Our approach to 2-vertex connectivity is based on the following lemma.

LEMMA 4.1. *Let* T *be a spanning tree of graph* G. *Two nodes are in the same block of* G *if and only if all tree edges on the path* P *between* u *and* v *are in the same block.*

Proof. If all edges on P are in the same block, then u and v are in the same block, since if an edge is in block B so are its endpoints. Conversely, assume u and v are in the same block B. Any simple path between u and v (such as P) must be entirely contained inside B. Otherwise, it must pass out of B through some cutpoint, and by definition of a cutpoint it cannot return into B without going through the same cutpoint, contradicting the assumption that P is simple. □

To use the lemma we must find an efficient way to test 2-vertex connectivity along tree paths. We will use the dynamic tree data structure of Sleator and Tarjan with cost related to nodes as a basis. By itself, however, this data structure is insufficient for our needs. We augment the data structure to solve 2-vertex connectivity with backtracking.

4.1. The dynamic tree data structure of Sleator and Tarjan. The fundamental principle behind the data structure is a partitioning of tree edges into vertex-disjoint paths, called a *path decomposition*. An edge within a path is called *solid*, while a nonpath edge is called *dashed*. Dynamic tree operations are performed by manipulating the path partition so as to place relevant vertices in the same path. Each path p has two endnodes *head*(p) and *tail*(p), which are the nodes on p that are farthest from and nearest to the root, respectively.

Let T be a tree rooted at r. There is a unique path decomposition of T defined by its *heavy edges*. Denote by $s(v)$ the number of descendants of node v, and denote

by $p(v)$ the parent of v. Let $\langle u, v\rangle$ denote a tree edge with $v = p(u)$. Call edge $\langle u, v\rangle$ *heavy* if $2s(u) > s(v)$, otherwise *light*. Removal of light edges leaves a collection of disjoint paths. There are $O(\log n)$ light edges on any path to the root.

The solid path decomposition maintained by the Sleator–Tarjan dynamic tree data structure is exactly the heavy path decomposition, except possibly during the execution of one of the dynamic tree operations. At the conclusion of each operation, however, the correspondence between solid paths and the unique heavy path decomposition is restored.

The *costs* of nodes are examined and changed using only two functions: *find-min-on-path* and *add-cost-to-path*. The former operation finds the minimum cost node on a solid path, and the latter increases the cost of all nodes on a single solid path. To perform a *find min*(u) operation, for example, the path from u to the root r must be turned into a single solid path to which *find-min-on-path* is applied. After the minimum is found, the heavy path decomposition is restored. Each solid path p is stored in a binary search tree D_p, where the leaves of the tree correspond to the nodes on the path so that in order on the tree corresponds to path order from head to tail. Each internal node of D_p has a "partial" cost. The cost of the solid path node v stored at leaf l of D_p is the sum of all the partial costs stored with the internal nodes on the path from the root of D_p down to the leaf node l. Thus, the cost of all nodes on the solid path p can be changed by Δ by adding Δ to the partial cost of the root of D_p. By maintaining minima of subtrees in D_p, the minimum on path p can be found in time linear in the depth of D_p. Using the binary tree data structure, a path p can be split at node v to give three new paths, p_1, v, and p_2, with p_1 containing the former head of p and p_2 containing the former tail. After the split, the cost of v can be determined in $O(1)$ time. The inverse of splitting is a concatenation, which produces a single path p consisting of p_1, followed by v, followed by p_2.

The path decomposition is manipulated by means of three functions: *expose*, *conceal*, and *reverse* [28]. An *expose* creates a solid path starting at a specified node v and ending at the root. The new path may not necessarily contain only heavy edges. A *conceal* takes a solid path p containing the root and possibly some light edges and modifies the collection of solid paths so that every edge incident with a node of p is solid if and only if it is heavy. A *reverse* operation reverses the direction of tree edges in a solid path ending at the root. Thus, for example, an evert at v is implemented by an expose of v, a reverse of the resulting path, and a conceal of the path now rooted at v.

Let P be the path from node v to the root of T. The *expose*(v) operation turns all the dashed edges on P into solid edges and simultaneously turns all the solid edges incident to but not on P into dashed edges. Let $\langle x, y\rangle$ be a dashed edge on P ($y = p(x)$), and let $\langle z, y\rangle$ be the solid edge containing a sibling z of x. If y has no heavy child, there is no such edge. The process of making $\langle x, y\rangle$ into a solid edge and $\langle z, y\rangle$ into a dashed edge is called a *splice*, denoted *splice*(x). At the time of the *splice*, let p_x be the solid path containing x as its tail. A *splice* involves splitting the solid path p containing y into p_1, y, p_2 (both p_1 and p_2 may be empty) and concatenating p_x, y, p_2. While y is a singleton, its cost can be obtained or updated in $O(1)$ time. The expose operation performs the necessary *splices* in order from v to the root. The solid path initially containing v is split as necessary so that v has no descendant solid edge.

The *conceal* operation is the inverse of *expose*. Given a solid path P containing the root, with head v, a *conceal* turns all the light edges on P into dashed edges, and

all the heavy edges incident to but not on P into solid edges. It thus restores the heavy path decomposition. *Conceal* processes P from its tail down. Let $\langle x, y\rangle$ be a solid light edge on P and let $\langle z, y\rangle$ be the dashed heavy edge incident on y. If y has no heavy child z, there is no such edge. The process of making $\langle x, y\rangle$ into a dashed edge and $\langle z, y\rangle$ into a solid edge is called a *slice*, denoted *slice*(x). Obviously, *slice* is the inverse of *splice*. At the time of the *slice*, let p_z be the solid path containing z as its tail. A *slice* involves splitting the solid path P into p_1, y, p_2 (both p_1 and p_2 may be empty) and concatenating p_z, y, p_2. The *conceal* then continues down path p_1, which has tail x, unless p_1 is empty. The method by which *conceal* determines which edges are light is quite clever and intricate. It involves keeping track of the number of descendants of dashed edges in the data structure.

4.2. A dynamic tree data structure for 2-vertex connectivity. We use the Sleator–Tarjan data structure as a basis for our data structure.

Let T be a tree rooted at r. Given a path decomposition for T, we categorize the tree edges incident on a node v in three ways. Edge $\langle v, w\rangle$ is a *parent* edge if $w = p(v)$ with respect to the current tree root. (The parent edge may be either solid or dashed.) Edge $\langle u, v\rangle$ is a *solid child* edge if it belongs to a solid path and $v = p(u)$. All other edges are *dashed child* edges. A solid edge can change to a parent edge, and vice versa, via a reverse. A solid child edge can change to a dashed child edge, or vice versa, via an expose or conceal. No single operation, however, can change an edge incident on v from being a parent to being a dashed child.

With each node v we associate an integer *counter value* $c(v)$ that, roughly speaking, takes the place of the cost value maintained by the basic dynamic tree structure. The counter value differs from the cost value, however, in that it depends on both the current 2-connected components of G and the current path decomposition of the spanning tree T. The counter values satisfy the following:

(i) if v has both a solid child edge and a parent edge, then $c(v)$ is zero if these two edges are in different blocks of G, and positive (including ∞) otherwise;

(ii) if v has no solid child edge or no parent edge, then $c(v) = +\infty$.

The counter condition implies that two edges belonging to the same solid path are in the same block if and only if all intervening path vertices have value greater than zero. Solid paths are implemented in the same manner as in the standard Sleator–Tarjan data structure, and counter values can be examined or set using the *find-min-on-path* and *add-cost-to-path* functions. The symbol $+\infty$ indicates a positive number that cannot be changed by *add-cost-to-path*. (Below, we explain this further.)

The counter value of node v says nothing about the relationship between dashed edges incident on v nor between dashed edges and solid edges incident upon v. An additional data structure handles these relationships. For each vertex v we maintain a *block partition*, $\mathcal{B}_v$, of the edges $\{e_1, e_2, \ldots, e_k\}$ adjacent to v. Let $\mathcal{B}_v(e_i)$ denote the set of $\mathcal{B}_v$ containing edge e_i. Let y be the solid child edge, if any, of v, and let z be the parent edge, if any, of v. At all times, edges e_i and e_j belong to the same 2-vertex-connected component (block) of G if and only if at least one of the following holds:

1. $\mathcal{B}_v(e_i) = \mathcal{B}_v(e_j)$.
2. $\mathcal{B}_v(e_i) = \mathcal{B}_v(y)$, $\mathcal{B}_v(e_j) = \mathcal{B}_v(z)$, and $c(v) > 0$.
3. As in 2, but with e_i and e_j exchanged.

The block partition is subject to *Unions*, *Finds*, and eventually *Deunions*. For each tree edge $e = \langle u, v\rangle$ there is a record containing the names of its endpoints and pointers to two representatives, one for $\mathcal{B}_u$ and one for $\mathcal{B}_v$. We denote these

representatives by $e_{v,u}(u)$ and $e_{u,v}(v)$, respectively. The two representatives of e contain back pointers to the record for e. This edge data structure is created and initialized when a new tree edge connecting two previously unconnected components is added.

To implement edge insertions, we use the standard dynamic tree operations such as *evert* and *link*. Each of these operations is in turn implemented with $O(1)$ invocations of the primitives *reverse*, *expose*, and *conceal*. We will also call these primitives directly in our implementations. As these primitives are executed, the block partitions and counter values must be modified to preserve the needed invariants.

If the counter values and block partitions are valid prior to a *reverse* operation, they remain valid after a *reverse*. The other two primitives change the solid path decomposition, however, and hence may cause changes in our counter values and block partitions. The *expose* primitive uses the function *splice*, which makes a light dashed edge solid and a heavy solid edge dashed. The *conceal* primitive uses the function *slice*, which makes a light solid edge dashed and a heavy dashed edge solid. For our purposes, whether the edges are heavy or light does not matter. The block partition and counter values are modified as follows.

Let $y = \langle u, v\rangle$ be a solid child edge of v which must be made dashed. Let z be the parent edge, if any, of v. If $\mathcal{B}_v(y) \neq \mathcal{B}_v(z)$ and $c(v) > 0$, then unite $\mathcal{B}_v(y)$ and $\mathcal{B}_v(z)$. In any case, set $c(v) = +\infty$.

Let $y = \langle u, v\rangle$ be a dashed child edge of v which must be made solid (there is no solid child edge of v at this point). Let z be the parent edge, if any, of v. If $\mathcal{B}_v(y) \neq \mathcal{B}_v(z)$, then set $c(v) = 0$; otherwise set $c(v) = +\infty$.

It is straightforward to verify by case analysis that these algorithms correctly maintain the counter values and block partitions through any sequence of queries and edge insertions. To facilitate access to the representatives for edges y and z in the above algorithms, each tree node v is augmented with pointers to the representatives in $\mathcal{B}_v$ of the solid edges incident to v, or the dashed edge from v to its parent, as appropriate. These pointers can be updated as part of the extended splice and slice in $O(1)$ time.

4.3. Implementation of test and insertion operations. In this subsection we present the implementation of *Test*(u, v) and *Insert Edge*(u, v). (The implementation of *Insert Vertex*() is straightforward.)

Test(u, v):

1. If u and v are in different components, return "no" and terminate.
2. Save the tree root r.
3. Evert the tree at v, making v the root. (Expose v and reverse the path from v to r.)
4. Expose u.
5. Perform *find-min-on-path* on the resulting solid path. Return "no" if the minimum value is 0; otherwise return "yes."
6. Conceal u and evert the tree at r.

Insert Edge(u, v):

1. If u and v are in different components, and hence different trees, evert at u, and perform Link(u, v). Add two singleton sets to the block partitions $\mathcal{B}_u$ and $\mathcal{B}_v$, each representing edge {u,v}. Then terminate.
2. If u and v are in the same tree, compute *Test*(u, v).

3. If the result is "no," then evert at v; expose u; increment counters on the resultant path with *add-cost-to-path*; conceal u.

LEMMA 4.2. *The above implementations of* Test(u, v) *and* Insert Edge(u, v) *are correct.*

Proof. Each operation uses $O(1)$ calls to the primitives *add-cost-to-path*, *find-min-on-path*, *expose*, *conceal*, and *reverse*. By the discussion from the previous section, these operations are correct.

Consider the *Test*(u, v) operation. Step 1 is trivially correct. Steps 2–4 construct a new path decomposition in which u and v are in the same solid path and u is the root of the tree. The primitives *expose*, *conceal*, and *reverse* construct correct counter values and block partitions, as defined in the previous section, for the new path decomposition. The endpoints of the path to which *find-min-on-path* is applied always have counter value $+\infty$. Hence the *find-min-on-path* operation determines whether there is a zero on a node internal to the path, which in turn determines whether u and v are in the same 2-connected component. Step 6 restores the original path decomposition.

The correctness of *Insert Edge*(u, v) follows from a similar argument. Step 3 increases all counter values along the path, so there is no longer a node of zero cost separating u and v. The final *conceal* will possibly change the path partition but will correctly construct new counter values and block partitions. □

LEMMA 4.3. *The above implementations of* Test(u, v) *and* Insert Edge(u, v) *run in* $\Theta((\log n)^2/\log\log n)$ *worst-case time per operation.*

Proof. The implementations of *add-cost-to-path*, *find-min-on-path*, *expose*, *conceal*, and *reverse* given by Sleator and Tarjan run in $\Theta(\log n)$ time in the worst case. In addition, the number of splices and slices performed per expose or conceal is $\Theta(\log n)$ in the worst case. Node counter values can be obtained or set to a particular value in $O(1)$ time when paths are split and concatenated during splices and slices. (We defer explaining how to implement $+\infty$ for the moment.) There are $O(1)$ *Union* and *Find* operations on block partitions during a splice or slice. *Unions* and *Finds* can be done in $\Theta(\log n/\log\log n)$ worst-case time per operation using Blum's data structure [3]. □

Hence, an incremental algorithm that runs in $\Theta((\log n)^2/\log\log n)$ worst-case time per operation is the result of this section.

4.4. The *Undo* operation. The algorithms for 2-vertex connectivity are more complicated than those for 2-edge connectivity, and the *Undo* operation is correspondingly more complex. Correct backtracking can be guaranteed by logging every change to a pointer or data field done in the course of an operation. By unwinding the log, each change can be exactly undone and the exact previous state of the data structure restored. This approach is space intensive, however. Our goal is to store a minimal amount of backtracking information. This means that the *Undo* will not restore the exact state of the data structure prior to the operation being undone. In particular, an *Undo* will restore the exact previous path decomposition, block partitions, counter values, and backtracking stack, but it will not necessarily restore the previous states of the data structures used to implement the solid paths and block partitions. There is no conceptual problem, since several different data structure states may represent the same solid path or block partition.

To prove the correctness of our implementation of backtracking, it suffices to show that by using the information stored on the backtracking stack during an update

operation, the update operation can be immediately undone. That is, no matter what the states of the data structures implementing the solid paths and block partitions, the previous path decomposition, block partitions, counter values, and backtracking stack can be restored. The correctness of the whole algorithm then follows by induction on the number of operations, since the answers to biconnectivity queries are determined only by these attributes.

The implementation of *Test*(u, v) given in subsection 4.3 may change the block partition, since it executes *expose* and *conceal* operations. It is most convenient to undo these changes immediately after the test operation is completed. This can be done in a brute-force fashion by logging each change to any pointer or data field and by storing the location of the field and the previous value. The old values can be restored by going backward through the log. Since *Test*(u, v) runs in $O((\log n)^2/\log\log n)$ time in the worst-case, the total size of the log and the time to restore the previous values is $O((\log n)^2/\log\log n)$. A better approach is to observe that the implementation of *Test*(u, v) is almost the same as the implementation of *Insert Edge*(u, v). Hence we may use the *Undo* algorithm for *Insert Edge*(u, v) developed below with only minor modification.

Each time an essential edge is inserted, a record is pushed onto the backtracking stack. As in the algorithm for 2-edge connectivity, an edge is essential only if its insertion reduces the number of components or 2-vertex components. Each record contains a counter that indicates the number of not-yet-undone nonessential edge insertions performed after the essential one described in the record. Upon an insertion of a nonessential edge, the only change to the data structure is to increment the counter in the top record. Upon an *Undo*, the counter in the top record is examined. If greater than zero, it is simply decremented. This suffices to restore the state of the data structure prior to the most recent insertion.

If the insertion is essential, a new record with a zero counter is pushed on the stack. The record describes the operation performed and the effect: either a decrease in the number of components or a decrease in the number of blocks. The record also describes each of the $O(1)$ "suboperations," *expose*, *reverse*, *conceal*, and *add-cost-to-path*, that were done.

A *reverse* can be undone by another *reverse* on the same path. The effect of incrementing counters by *add-cost-to-path* can be undone by using *add-cost-to-path* to add -1 to the same path.

Undoing the effects of an *expose* or *conceal* is more complicated. For each such operation, the main backtracking record contains a substack of subrecords. This substack will record changes to the block partitions that occur during the operation.

Suppose we expose v and we want to immediately undo the *expose*. The heavy path decomposition that existed prior to the expose can be restored by an immediate *conceal*. Since the heavy path decomposition is unique, each *splice* occurring in the *expose* will be exactly undone by a *slice* in the *conceal*. The *conceal* algorithm processes each sliced edge in the reverse order that it was spliced by the *expose*. Similarly, a *conceal* operation, which travels down a solid path turning light edges on the path into dashed edges, can be immediately undone by an *expose* starting at the last vertex on the solid path traversed by the *conceal*. Each *slice* done by the *conceal* is exactly undone by a *splice* in the *expose*.

To restore the counter values and block partitions, it suffices to show how to modify *splice* to undo the effects of an immediately preceding *slice* and vice versa. The function *splice* makes a heavy solid child dashed, if there is one, and then makes

a light dashed child solid. The function *slice* makes a light solid child dashed, and then makes a heavy dashed child solid, if there is one. With respect to the counter and block partition, it does not matter whether the edges are heavy or light. Hence it suffices to show how to undo the effect of turning a dashed edge solid and how to undo the effect of turning a solid edge dashed.

The block partitions are managed by an algorithm for set *Union* with backtracking that supports the operations *Find*, *Union*, and *Deunion*. As described in section 1, these operations can be implemented in $\Theta(\log n / \log\log n)$ time, either worst case or amortized.

Let $y = \langle u, v\rangle$ be solid child edge of v which must be made dashed as part of a normal edge insertion. Let z be the parent edge, if any, of v. If $\mathcal{B}_v(y) \neq \mathcal{B}_v(z)$ and $c(v) > 0$, then unite $\mathcal{B}_v(y)$ and $\mathcal{B}_v(z)$; push a new subrecord on the substack, labeled with the name "v"; store the current value of $c(v)$ into the subrecord; and set $c(v) = +\infty$. In all other cases, simply set $c(v) = +\infty$. If x is being made dashed during an *Undo*, simply set $c(v) = +\infty$.

Let $y = \langle u, v\rangle$ be a dashed child edge of v which must be made solid. Let z be the parent edge, if any, of v. If $\mathcal{B}_v(y) \neq \mathcal{B}_v(z)$, then set $c(v) = 0$; otherwise set $c(v) = +\infty$. If y is being made dashed as part of an *Undo*, then examine the top subrecord on the substack. If it is labeled "v," then pop the subrecord, execute a *Deunion* in the block partition for v, and set $c(v)$ equal to the value stored in the subrecord.

By inspecting these routines, one may verify that after a normal solid-to-dashed operation, an immediate dashed-to-solid operation in *Undo* mode will correctly restore the previous block partition and counter value. Similarly, after a normal dashed-to-solid operation, an immediate solid-to-dashed operation in *Undo* mode correctly restores the previous state. Recall that to make a dashed child solid, v can have no other solid child, and hence $c(v) = +\infty$.

THEOREM 4.4. *A sequence of* m Test(u, v), Insert Vertex(), Insert Edge(u, v), *and* Undo *operations can be performed in* $\Theta((\log n)^2 / \log\log n)$ *worst-case time per operation and in* $\Theta(n)$ *space.*

Proof. As discussed above, the correctness of the *Undo* algorithm follows by induction, since sufficient information is stored on the backtrack stack to allow each function that changes the path decomposition or counter values to be immediately undone. The running time of an *Undo* is order of the running time of the operation being undone, which is $\Theta((\log n)^2 / \log\log n)$ in both cases.

Next we consider the space utilization. After $O(n)$ essential edge insertions, all edges are in the same 2-vertex component. Since $O(1)$ records are only pushed on the backtrack stack if components or blocks are joined, and $O(1)$ subrecords are only pushed on a substack if sets in the block partitions are united, the total space required on the backtrack stack is $O(n)$ main records plus $O(n)$ total subrecords. Obviously, $\Omega(n)$ space is required just to store the vertices. □

We use $+\infty$ to ease designing and analyzing the algorithm. It ensures that the endpoints of a path always have positive counter value and so cannot interfere with the operation *find-min-on-path*. If v is internal to a solid path, and both incident solid edges belong to the same set of the block partition, then $c(v)$ is also $+\infty$. This is an easy way to ensure that no amount of counter decrements performed during *Undo*s will accidentally reduce $c(v)$ to zero, thereby violating the counter condition. For theoretical purposes, there is no difficulty in assuming that the arithmetic operations of the target machine are augmented to handle $+\infty$. For actual implementations,

we can dispose of $+\infty$ as follows. Say that the value of $c(v)$ is *equivalent* to $+\infty$ if it is greater than the number of not-yet-undone counter increments that have been applied to v. Since this number is at least zero, a value equivalent to $+\infty$ is always positive. Modify the algorithms so that wherever a counter was previously set to $+\infty$ it is now set to one more than the current size of the backtrack stack. One may show the correctness of this modified algorithm by imagining that the original and modified algorithms are run side by side on the same input and verifying by induction that a counter in the modified algorithm is equivalent to $+\infty$ exactly when the corresponding counter in the original algorithm is equal to $+\infty$.

4.5. An $\Theta(\log n)$ algorithm for 2-vertex connectivity. We improve the worst-case running time to $\Theta(\log n)$ per operation by using *globally biased binary search trees* [2], [28] to implement the block partitions. In the Sleator–Tarjan data structure, the solid paths are already implemented by globally biased trees.

In a biased binary tree, A, each node has a specified *weight* $w(x)$. Let A_v be the subtree of A rooted at v; the *size* of a biased-tree node, $s(v)$, is defined as $s(v) = \sum_{u \in A_v} w(u)$. Define the *rank* of node v, $r(v)$, as $\log s(v)$. We use $r(A)$ to denote the rank of the root of biased binary tree A.

The biased binary tree data structure has the following properties.

1. The depth of node $v \in A$ is $O(r(A) - r(v))$.
2. For $v \in A$, the operation *split*(v) produces trees A_1, v, A_2 and requires time $O(r(A) - r(v))$.
3. The operation *concat*(A_1, A_2) concatenates two trees A_1 and A_2 in time

$$O(\max\{r(A_1), r(A_2)\} - \max\{r(\mathit{right}(A_1)), r(\mathit{left}(A_2))\}),$$

where *left*(A) and *right*(A) denote the leftmost and rightmost nodes, respectively, in tree A.

Note that concatenating trees A_1, v, and A_2 to give tree A requires time $O(r(A) - r(v))$. For this can be done by *concat*(*concat*$(A_1, v), A_2)$, requiring time $O(r(A_1) - r(v)) + O(\max(\max(r(A_1), r(v)) + 1, r(A_2)) - r(v))$ which is $O(r(A) - r(v))$.

Thus, splitting a tree A into A_1, v, A_2 by *split*(v) can be undone by concatenating A_1, v, and A_2 both within the same time bound $O(r(A) - r(v))$ and vice versa.

Recall that each tree edge $\langle u, v\rangle$ has two representatives, one each in the block partitions for u and v. Denote these by $e_{v,u}(u)$ and $e_{u,v}(v)$, respectively. If $\langle u, v\rangle$ is a dashed child edge of v, define $w(e_{u,v}(v))$ to be the number of descendants of u in the spanning tree T. If $\langle u, v\rangle$ is a parent or solid edge of v, define $w(e_{u,v}(v))$ to be zero. These weights are already explicitly maintained in the standard Sleator–Tarjan data structure and can be accessed in $O(1)$ time when needed by our modified algorithms.

Each block partition set $B \in \mathcal{B}_v$ is implemented by a header that contains pointers to one or two biased binary trees. The binary trees contain the dashed child edges of v, using the weights given above. Each tree root contains a back pointer to the set header. Set B may be stored in two trees if it contains both the parent and solid child edges of v, and it is stored in one tree otherwise. In case of two trees, one of them precedes the other as indicated by the pointers (distinguished as a left and a right pointer). We usually indicate this order by using indices with the trees, e.g., X_1,X_2. Furthermore, each parent or solid child edge e contains a pointer to the set header of $\mathcal{B}_v(e)$. Hence testing whether $\mathcal{B}_v(e_1) = \mathcal{B}_v(e_2)$ can be done by obtaining and comparing the corresponding set header for e_1 and e_2, either by using a direct pointer to a set header in case of a parent or solid child or by traversing the root path and using the pointer from the root to the set header in the case of a dashed child.

For biased binary tree A, we denote by $h(A)$ the node in A with maximum weight. In the case of a tie, the node with largest index is taken. (We assume that nodes have unique names $1, \ldots, n$). The following invariant is maintained.

> If $B \in \mathcal{B}_v$ contains neither a parent nor a solid edge, then the (single) biased tree A for set B satisfies $right(A) = h(A)$.

If the counter values and block partitions are valid prior to a *reverse* operation, they remain valid after the path reversal. This is because the block partition and counter values are modified only when a dashed child turns into a solid child. The primitives *expose* and *conceal*, however, may cause changes to counter values and block partitions via the functions *splice* and *slice*. Recall from sections 4.2 and 4.4 that it suffices to show how to turn a solid edge dashed, how to turn a dashed edge solid, and how to undo each of these actions.

We first examine how the block partition and counter values change when a solid edge $y = \langle u, v\rangle$ is made dashed. We assume that v is not the root, and that the parent edge of v is z. If v is the root, execute the following algorithm as if z exists but forms a singleton block, i.e., $\mathcal{B}_v(z) = \{z\}$. We will use x, y, and z to denote both edges incident on v and their representatives in the block partition trees of v.

1. If y is to be made dashed as part of a normal edge insertion, then test if $\mathcal{B}_v(y) \neq \mathcal{B}_v(z)$ and $c(v) > 0$. If so, then let Y be the tree of $\mathcal{B}_v(y)$ and Z the tree of $\mathcal{B}_v(z)$. Unite sets $\mathcal{B}_v(y)$ and $\mathcal{B}_v(z)$ by creating a new set header with two pointers to Y, Z, in that order. Push a new subrecord on the substack, labeled with the name "v," and copy $c(v)$ into the subrecord. Continue with the rest of this routine.

2. If $\mathcal{B}_v(y) = \mathcal{B}_v(z)$ then let Y_1, Y_2 be the two trees of set $\mathcal{B}_v(y)$. Concatenate (Y_1, y, Y_2).

3. If $\mathcal{B}_v(y) \neq \mathcal{B}_v(z)$ ($c(v) = 0$) then let Y be the tree for $\mathcal{B}_v(y)$. Let $Y' = concat(Y, y)$. Find $y' = h(Y')$. Perform $split(y')$, giving Y_1, y', Y_2, followed by concatenate (Y_2, Y_1, y'). The result is the new biased tree for $\mathcal{B}_v(y)$.

4. Set $c(v) = +\infty$.

We need to augment the biased tree data structure to allow us to search for the maximum weighted node $h(Y')$ in step 3. This can be done in a standard fashion, storing at each internal node of the biased tree the maximum weight/index pair of any descendant. Using this information, node $h(Y')$ can be found in time proportional to its depth, which is $O(r(Y') - r(h(Y')))$.

Next we examine how the block partition and counter values change when a dashed edge $x = \langle u, v\rangle$ is made solid. As before, if v is the root, execute the following algorithm as if z exists but forms a singleton block, i.e., $\mathcal{B}_v(z) = \{z\}$.

1. Let X be the biased tree of $\mathcal{B}_v(x)$. Perform $split(x)$, giving X_1, x, X_2. (Possibly $X_2 = \emptyset$.)

2. If $\mathcal{B}_v(x) \neq \mathcal{B}_v(z)$, then set $c(v) = 0$, execute $concat(X_2, X_1)$, and store a pointer to the result in the header for $\mathcal{B}_v(x)$.

3. If $\mathcal{B}_v(x) = \mathcal{B}_v(z)$, then set $c(v) = +\infty$ and store pointers to X_1, X_2 in that order.

4. If x is being made dashed as part of an *Undo*, then examine the top subrecord on the substack. If it is labeled "v," then pop the subrecord. Set $c(v)$ equal to the value stored in it. *Undo* the *Union* indicated by the subrecord creating two headers for $\mathcal{B}_v(x)$ and $\mathcal{B}_v(z)$, respectively. Make X_1 the tree for $\mathcal{B}_v(x)$ and X_2 the tree for $\mathcal{B}_v(z)$.

One may easily verify that these routines maintain valid counters and block partitions if no *Undo* operations are performed. We defer for the moment a discussion of

the *Undo* operation, and turn to an analysis of the running time.

LEMMA 4.5. *The functions* splice(u) *and* slice(u) *run in time* $O(1 + \log s(v) - \log s(u))$, *where* $v = p(u)$, $s(u)$, *and* $s(v)$ *are the number of descendants of* u *and* v, *respectively, given the current root of the spanning tree* T.

Proof. Consider the function that makes a solid child edge y dashed. Since the set headers for y and z can be accessed in $O(1)$ time, all set equivalence tests take $O(1)$ time. Step 1 requires $O(1)$ time. Step 2 requires time $O(r(Y') - r(y))$ for the concatenation, where Y' is the resulting tree. Step 3 requires time $O(r(Y') - r(y))$ for all the concatenation and split operations, since $r(y') \geq r(y)$ by definition, and in the final concatenation y is rightmost in Y_2.

Consider the function that makes a dashed child edge x solid. The split in step 1 takes time $O(r(X) - r(x))$. After the split, we have $O(1)$ time access to the set header for $\mathcal{B}_v(x)$. The concatenation in step 2 requires time $O(r(X) - r(h(X))) = O(r(X) - r(x))$. This follows because $h(X)$ was rightmost in X by the invariant and hence in X_2 (if $X_2 \neq \emptyset$), and $r(h(X)) \geq r(x)$, by definition. Steps 3 and 4 take time $O(1)$.

Splice and *slice* both perform one each of these operations. In both cases, the maximum rank of any tree in the block partition is $\log s(v)$. Hence the cost of a *splice* or *slice* is $O(1 + \log s(v) - r(x) - r(y))$.

In the case of a *splice*, $r(x) = \log s(u)$ by definition, and since y is a heavy child, $w(y) \geq s(v)/2$. It follows that a *splice* takes $O(1 + \log s(v) - \log s(u))$ time.

In the case of a *slice*, $r(y) = \log s(u)$ and $w(x) \geq s(v)/2$ for the analogous reason. This implies that *slice* takes $O(1 + \log s(v) - \log s(u))$ time. □

LEMMA 4.6. *Using globally biased trees, an* expose *or* conceal *operation requires time* $\Theta(\log n)$.

Proof. An *expose*(u) operation consists of a sequence of *splices* along the tree path from u to the root. Let $u_1, u_2, \ldots, u_k$ be the sequence of nodes at which a *splice* occurs, and let $v_i = p(u_i)$ for all i. By Lemma 4.5 the cost of the *expose* is $\sum_{i=1}^{k} O(1 + \log s(v_i) - \log s(u_i))$. Since $s(v_i) \leq s(u_{i+1})$ for $1 \leq i \leq k-1$, the sum telescopes to $O(k + \log n)$. The heavy path decomposition guarantees that $k = O(\log n)$, although $k = \Omega(\log n)$ in the worst case. A *conceal* operation is implemented by a sequence of *slices* going down the tree. Again the costs telescope for a total of $\Theta(\log n)$. □

Now consider the *Undo* operation. As in section 4.4, it suffices to verify that after a normal solid-to-dashed operation, an immediate dashed-to-solid operation in *Undo* mode will correctly restore the previous block partition and counter value. Similarly, after a normal dashed-to-solid operation, an immediate solid-to-dashed operation in *Undo* mode correctly restores the previous state.

Suppose a solid edge is made dashed. Either steps 2 and 4, steps 3 and 4, or steps 1, 2, and 4 of the solid-to-dashed routine are executed. Steps 2 and 4 of solid-to-dashed will be undone by steps 1 and 3 of dashed-to-solid. Steps 3 and 4 of solid-to-dashed will be undone by steps 1 and 2 of dashed-to-solid. Finally, steps 1, 2, and 4 of solid-to-dashed are undone by steps 1, 3, and 4 of dashed-to-solid.

Similarly, suppose a dashed edge is made solid. Either steps 1 and 2 or steps 1 and 3 of the dashed-to-solid routine are executed. Steps 1 and 2 of dashed-to-solid are undone by steps 3 and 4 of solid-to-dashed. Steps 1 and 3 of dashed-to-solid are undone by steps 2 and 4 of solid-to-dashed.

THEOREM 4.7. *A sequence of* m Test(u, v), Insert Vertex(), Insert Edge(u, v), *and* Undo *operations can be performed in* $\Theta(\log n)$ *worst-case time per operation and*

in $\Theta(n)$ *space.*

Proof. The running time follows from Lemma 4.6 and the previously established running time for the Sleator–Tarjan data structure. The space bound follows from the proof of Theorem 4.4. □

5. Remarks. The only lower bound known for backtracking problems is one of $\Omega(\log n / \log\log n)$ for a disjoint set union with backtracking [32]. This bound can be applied to any class of algorithms for backtracking graph problems that keep disjoint data structures for distinct components of the graph. Our algorithms fall into that class. Hence there remains a gap of $\Theta(\log\log n)$ between the known upper and lower bounds for 2-connectivity with backtracking. Note that in one operation, $\Omega(n)$ 2-connected components may be joined (or unjoined), which makes the problem apparently harder disjoint set union, where only two sets can be joined by a single step.

Other interesting open problems are backtracking algorithms for 3-connectivity and general planarity testing. The latter has as an application, e.g., VLSI design, amongst others. We anticipate that our algorithms herein provide a basis for efficient solutions of these problems.

Acknowledgments. We thank the anonymous referees for helpful comments.

REFERENCES

[1] G. D. Battista and R. Tamassia, *On-line maintenance of triconnected components with spqr-trees*, Algorithmica, 15 (1996), pp. 302–318.
[2] S. Bent, D. D. Sleator, and R. E. Tarjan, *Biased search trees*, SIAM J. Comput., 14 (1985), pp. 545–568.
[3] N. Blum, *On the single-operation worst-case time complexity of the disjoint set union problem*, SIAM J. Comput., 15 (1986), pp. 1021–1024.
[4] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, McGraw-Hill, New York, 1990.
[5] G. Di Battista and R. Tamassia, *On-line planarity testing*, SIAM J. Comput., 25 (1996), pp. 956–997.
[6] J. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan, *Making data structures persistent*, J. Comput. Sys. Sci., 38 (1989), pp. 86–124.
[7] D. Eppstein, Z. Galil, G. Italiano, and A. Nissenzweig, *Sparsification: A general technique for dynamic graph algorithms*, in Proc. 33rd Symp. of Foundations of Computer Science, 1992.
[8] D. Eppstein, Z. Galil, G. Italiano, and T. Spencer, *Separator based sparsification* I. *Planarity testing and minimum spanning trees*, J. Comput. Syst. Sci., 52 (1996), pp. 3–27.
[9] G. N. Frederickson, *Data structures for on-line updating of minimum spanning trees, with applications*, SIAM J. Comput., 14 (1985), pp. 781–798.
[10] G. N. Frederickson, *Ambivalent data structures for dynamic* 2*-edge-connectivity and k smallest spanning trees*, SIAM J. Comput., 26 (1997), pp. 484–538.
[11] Z. Galil and G. F. Italiano, *Data structures and algorithms for disjoint set union problems*, Computing Surveys, 23 (1991), 319–344.
[12] Z. Galil and G. F. Italiano, *Fully dynamic algorithms for* 2*-edge connectivity*, SIAM J. Comput., 21 (1992), pp. 1047–1069.
[13] Z. Galil and G. F. Italiano, *Maintaining the* 3*-edge connected components of a graph on-line*, SIAM J. Comput., 22 (1993), pp. 11–28.
[14] A. Apostolico, G. F. Italiano, G. Gambosi, and M. Talamo, *The set union problem with unlimited backtracking*, SIAM J. Comput., 23 (1994), pp. 50–70.
[15] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1972.
[16] G. F. Italiano, J. A. La Poutré, and M. H. Rauch, *Fully dynamic planarity testing in planar embedded graphs*, in Algorithms - ESA '93, Lecture Notes in Computer Science 726, T. Lengauer, ed., Springer-Verlag, Berlin, 1993, pp. 212–223.
[17] V. King and M. Rauch-Henzinger, *Randomized dynamic algorithms with polylogarithmic time per update*, in Proc. 27th ACM Symp. on Theory of Computing, 1995, pp. 519–527.

[18] J. A. La Poutré, *Maintenance of* 2- *and* 3-*Connected Components of Graphs, part* ii: 2- *and* 3-*Edge-Connected Components and* 2-*Vertex-Connected Components*, Technical Report RUU-CS-90-27, Utrecht University, 1990.
[19] J. A. La Poutré, *Dynamic Graph Algorithms and Data Structures*, Ph.D. thesis, University of Utrecht, Netherlands, 1991.
[20] J. A. La Poutré, *Maintenance of triconnected components of graphs*, in Proc. Int. Colloquium on Automata, Languages, and Programming (ICALP '92), Lecture Notes in Computer Science 623, Springer-Verlag, New York, 1992, pp. 354–365.
[21] J. A. La Poutré, *Alpha algorithms for incremental planarity testing*, in Proc. 26th ACM Symp. on Theory of Computing, 1994, pp. 706–715.
[22] J. A. La Poutré, J. van Leeuwen, and M. H. Overmars, *Maintenance of* 2- *and* 3-*edge-connected components of graphs*, Discrete Math., 114 (1993), pp. 329–359.
[23] H. Mannila and E. Ukkonen, *On the complexity of unification sequences*, in Third International Conference on Logic Programming, Lecture Notes in Computer Science 225, Springer-Verlag, New York, 1986, pp. 122–133.
[24] H. Mannila and E. Ukkonen, *The set union problem with backtracking*, in Proc. 13th International Colloquium on Automata, Languages, and Programming (ICALP 86), Lecture Notes in Computer Science 226, Springer-Verlag, New York, 1986, pp. 236–243.
[25] G. Port, Private communication, 1988.
[26] M. Rauch Henzinger, *Fully dynamic biconnectivity in graphs*, Algorithmica, 13 (1995), pp. 503–538.
[27] M. Rauch, *Improved data structures for fully dynamic biconnectivity*, in Proc. 26th ACM Symp. on Theory of Computing, 1994, pp. 686–695.
[28] D. D. Sleator and R. E. Tarjan, *A data structure for dynamic trees*, J. Comput. System Sci., 26 (1983), pp. 362–391.
[29] R. Tamassia, *On-line planar graph embedding*, J. Algorithms, 21 (1996), pp. 201–239.
[30] J. Westbrook, *Algorithms and Data Structures for Dynamic Graph Problems*, Ph.D. thesis, Department of Computer Science, Princeton University, Princeton, NJ, October 1989.
[31] J. Westbrook, *Fast incremental planarity testing*, in Proc. Int. Symp. on Automata, Languages and Programming (ICALP '92), Lecture Notes in Computer Science, Springer-Verlag, New York, 1992.
[32] J. Westbrook and R. E. Tarjan, *Amortized analysis of algorithms for set union with backtracking*, SIAM J. Comput., 18 (1989), pp. 1–11.
[33] J. Westbrook and R. E. Tarjan, *Maintaining bridge-connected and biconnected components on-line*, Algorithmica, 7 (1992), pp. 433–464.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 27–35

ON THE STRUCTURE OF $\mathcal{NP}_{\mathbb{C}}$*

GREGORIO MALAJOVICH† AND KLAUS MEER‡

Abstract. This paper deals with complexity classes $\mathcal{P}_{\mathbb{C}}$ and $\mathcal{NP}_{\mathbb{C}}$ as they were introduced over the complex numbers by Blum, Shub, and Smale [*Bull. Amer. Math. Soc.*, 21 (1989), p. 1]. Under the assumption $\mathcal{P}_{\mathbb{C}} \neq \mathcal{NP}_{\mathbb{C}}$ the existence of noncomplete problems in $\mathcal{NP}_{\mathbb{C}}$ not belonging to $\mathcal{P}_{\mathbb{C}}$ is established.

Key words. complexity, NP-completeness, BSS machines over $\mathbb{C}$, Hilbert Nullstellensatz

AMS subject classifications. Primary, 68Q15; Secondary, 68Q05, 03D15

PII. S0097539795294980

1. Introduction.

In 1989 Blum, Shub, and Smale introduced a computational model over arbitrary ring structures [3]. It especially results in a complexity theory over the complex numbers $\mathbb{C}$ as well as a complex analogue "$\mathcal{P}_{\mathbb{C}} \neq \mathcal{NP}_{\mathbb{C}}$?" of the famous $\mathcal{P} \neq \mathcal{NP}$? problem over the integers. One of the main parts in [3] is devoted to asking for the existence of $\mathcal{NP}_{\mathbb{C}}$-complete problems. The main example is Hilbert's Nullstellensatz.

DEFINITION 1 (Hilbert's Nullstellensatz). *Let $\mathbb{K}$ be an algebraically closed field. The* Hilbert–Nullstellensatz *decision problem over $\mathbb{K}$, denoted by $HN_{\mathbb{K}}$, is defined as follows.*

Let $f_1, \ldots, f_s \in \mathbb{K}[x_1, \ldots, x_n]$ be given polynomials, $s, n \in \mathbb{N}$, $\deg f_i = 2$.

Decide if there is a common zero $x \in \mathbb{K}^n$ for all f_i.

For example, following the terminology of [3] the polynomial systems over $\mathbb{K}$ build the "input set" of decision problem $\mathrm{HN}_{\mathbb{K}}$, whereas the solvable ones build the "yes set.")

Assume that $\mathbb{K}$ is the field $\mathbb{C}$ of complex numbers or the algebraic closure $\overline{\mathbb{Q}}$ of the rationals over $\mathbb{C}$.

In the Blum, Shub, Smale (BSS) model over $\mathbb{K}$, the computational cost of every algebraic operation as well as the size of any element in $\mathbb{K}$ is supposed to be 1. Under those assumptions, it was proven in [3] that $\mathrm{HN}_{\mathbb{K}}$ is $\mathcal{NP}_{\mathbb{K}}$-complete.

Thus $\mathrm{HN}_{\mathbb{C}}$ represents the entire difficulty of class $\mathcal{NP}_{\mathbb{C}}$; it is solvable in polynomial time if and only if $\mathcal{P}_{\mathbb{C}} = \mathcal{NP}_{\mathbb{C}}$. In fact, most mathematicians assume $\mathcal{P}_{\mathbb{C}} \neq \mathcal{NP}_{\mathbb{C}}$, implying that $\mathrm{HN}_{\mathbb{C}}$ does not allow efficient (i.e., polynomial-time) algorithms. (For a more intensive treatment of related results we refer the interested reader to the forthcoming book [2] as well as the survey paper [10].)

However, except for the above-mentioned result only a few things are known about the intrinsic structure of $\mathcal{NP}_{\mathbb{C}}$ if $\mathcal{P}_{\mathbb{C}} \neq \mathcal{NP}_{\mathbb{C}}$ is assumed.

In the present paper we want to go a step in this direction and will show the following theorem.

* Received by the editors November 20, 1995; accepted for publication (in revised form) October 28, 1996; published electronically June 15, 1998.
http://www.siam.org/journals/sicomp/28-1/29498.html

† Departamento de Matematica Aplicada, Instituto de Matematica da UFRJ, Caixa Postal 68530, LEP 21945, Rio, Brazil (gregorio@lyric.labma.ufrj.br). This research was partially supported by CNPq (Brazil) grant 520305/94-9.

‡ Lehrstuhl C für Mathematik, RWTH Aachen, Templergraben 55, 52062 Aachen, Germany (meer@rwth-aachen.de). This paper was written while Klaus Meer was visiting Gregorio Malajovich. The visit was sponsored by CNPq (452267/95-1) and by UFRJ.

MAIN THEOREM. *Assume $\mathcal{P}_{\mathbb{C}} \neq \mathcal{NP}_{\mathbb{C}}$. Then there is a decision problem in $\mathcal{NP}_{\mathbb{C}} \setminus \mathcal{P}_{\mathbb{C}}$ that is not $\mathcal{NP}_{\mathbb{C}}$-complete.*

The theorem is not surprising in stating that, once leaving the class of $\mathcal{NP}_{\mathbb{C}}$-complete problems, one will not directly jump into $\mathcal{P}_{\mathbb{C}}$.

The proof is constructive in the sense that we will describe exactly how to build a noncomplete decision problem in $\mathcal{NP}_{\mathbb{C}} \setminus \mathcal{P}_{\mathbb{C}}$. This problem is not a natural one. Finding more natural noncomplete problems may be an interesting task. We will discuss this question at the end of the paper.

Let us first briefly sketch the proof. The basic idea is to produce a noncomplete problem not in $\mathcal{P}_{\mathbb{C}}$, starting from a complete one. That idea was given by Ladner in [8]. There the Turing model analogue of the main theorem was established. Also see [12] for a more general approach to obtaining such results over the integers.

Given a complete problem one will switch stepwise to an easier one by changing the problem "dimensionwise." Here we will start with $\mathrm{HN}_{\mathbb{C}}$ and then carefully construct another problem. On some input dimensions it will still represent the $\mathrm{HN}_{\mathbb{C}}$ problem, whereas on the other dimensions it represents a trivial one.

However, Ladner's approach heavily relies on the fact that in the Turing setting the sets of $\mathcal{P}$- and $\mathcal{NP}$-machines are effectively countable. This is not at all the case for BSS machines over $\mathbb{C}$. We will circumvent the latter problem by using the following transfer principle given in [2, Chapter 6].

THEOREM 1 (Blum–Cucker–Shub–Smale). *Let (Y, Y_0) be a decision problem solved by a BSS machine M over $\mathbb{C}$. Moreover, let $(Y_{\overline{\mathbb{Q}}}, Y_{0\overline{\mathbb{Q}}})$ be its restriction to $\overline{\mathbb{Q}}$, i.e., $Y_{\overline{\mathbb{Q}}} := Y \cap \overline{\mathbb{Q}}^\infty$ and $Y_{0\overline{\mathbb{Q}}} := Y_0 \cap \overline{\mathbb{Q}}^\infty$.*

Then there exists a constant $c \in \mathbb{N}$ as well as a machine M' over $\overline{\mathbb{Q}}$ solving $(Y_{\overline{\mathbb{Q}}}, Y_{0\overline{\mathbb{Q}}})$ s.t. the running time $T_{M'}(y)$ of M' for all $y \in Y_{\overline{\mathbb{Q}}}$ is bounded by $T_M(y)^c$ (where $T_M(y)$ is the running time of M on y).

The transfer principle should be considered as follows. We first show the main theorem for the BSS-model over $\overline{\mathbb{Q}}$. This will be carried out in a similar way as Ladner's proof, but attention must be paid to several details such as the enumeration of all $\mathcal{P}_{\overline{\mathbb{Q}}}$-machines. To do this, symbolic computations for dealing with algebraic numbers will be necessary. In that way we built up two subproblems of $\mathrm{HN}_{\overline{\mathbb{Q}}}$ belonging to $\mathcal{NP}_{\overline{\mathbb{Q}}} \setminus \mathcal{P}_{\overline{\mathbb{Q}}}$. Then both subproblems are shown to be noncomplete. This part also differs from Ladner's approach in that we do not need effective countability of $\mathcal{NP}_{\overline{\mathbb{Q}}}$ machines.

We intend for this paper to be self-contained. Therefore, we will present the whole construction over $\overline{\mathbb{Q}}$, even though those readers that are aware of [8] would be able to fill in the remaining gaps by getting fewer details.

Finally, the above-mentioned subproblems are extended to decision problems over $\mathcal{NP}_{\mathbb{C}}$. The transfer principle guarantees that they do not to belong to $\mathcal{P}_{\mathbb{C}}$, too. Furthermore, its special structure will ensure that both of them will not be complete for $\mathcal{NP}_{\mathbb{C}}$.

2. Sketch of the proof for $\overline{\mathbb{Q}}$. In this section we are going to outline the proof of the main theorem where $\mathbb{C}$ is replaced by $\overline{\mathbb{Q}}$. The proof is given in section 3.

Note that according to [3] $\mathrm{HN}_{\overline{\mathbb{Q}}}$ is $\mathcal{NP}_{\overline{\mathbb{Q}}}$ - complete. We will need Theorem 2.

THEOREM 2. *There is a decision procedure for $\mathcal{NP}_{\overline{\mathbb{Q}}}$ working in time $s^{O(n)}$.*

For more details, see [6], [4], [5], or [11].

We chose dense representation for the system f that appears in the Hilbert Nullstellensatz. Therefore, its size S is greater than $sn \geq n \log_2(s)$. Hence, the Hilbert

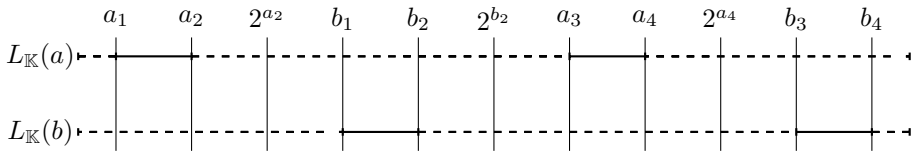


FIG. 1. *Problems $L_{\mathbb{K}}(a)$ and $L_{\mathbb{K}}(b)$.*

Nullstellensatz can be solved in exponential time $2^{O(S)}$ with respect to the input size.

The idea is to start with $\mathrm{HN}_{\overline{\mathbb{Q}}}$ and turn it into a sparser and sparser problem. This will be done by changing the problem to look like the empty set over $\overline{\mathbb{Q}}^n$ for certain "input dimensions" n.

Assume a fixed coding is given that represents an instance of $\mathrm{HN}_{\mathbb{C}}$, respectively, $\mathrm{HN}_{\overline{\mathbb{Q}}}$ as an element of some space $\mathbb{C}^n$ or $\overline{\mathbb{Q}}^n$ (where n only depends on the number of polynomials and the number of variables). The coding is assumed to be *dense*. This means that polynomials are represented by the coefficients of all possible monomials that may appear. For instance, the size of the representation of one quadratic polynomial in two variables should be at least 6, regardless of zero coefficients.

The following definitions will be crucial for understanding the structure of the decision problems we have to build.

DEFINITION 2. *Let $\mathbb{K} \in \{\overline{\mathbb{Q}}, \mathbb{C}\}$; for any strictly increasing sequence $a := (a_1, a_2, \ldots)$ of natural numbers let $L_{\mathbb{K}}(a)$ be the following decision problem over $\mathbb{K}$:*

for any input dimension $n \in \mathbb{N}$,

$$L_{\mathbb{K}}(a) \cap (\mathbb{K}^n, \mathbb{K}^n) := \begin{cases} HN_{\mathbb{K}} \cap (\mathbb{K}^n, \mathbb{K}^n) & \text{if } \exists i \in \mathbb{N} \text{ s.t. } a_{2i-1} \leq n < a_{2i}, \\ (\emptyset, \emptyset) & \text{otherwise.} \end{cases}$$

We call each of the subsets $\{a_j, \ldots, a_{j+1}\}$ a cluster *of the associated problem.*

DEFINITION 3. *Two sequences (a) and (b) are said to have an* exponential gap *if and only if they are strictly increasing and*

$$\begin{aligned} a_{2i+1} &> 2^{b_{2i}} \quad (i \geq 1), \\ b_{2i+1} &> 2^{a_{2i+2}} \quad (i \geq 0). \end{aligned}$$

Or, again,

$$a_1 < a_2 < 2^{a_2} < b_1 < b_2 < 2^{b_2} < a_3 < a_4 < 2^{a_4} < b_3 < \cdots.$$

Let's make the preceding definition a little bit more clear. Assume (a) and (b) have an exponential gap. Figure 1 shows the corresponding problems $L_{\mathbb{K}}(a)$ and $L_{\mathbb{K}}(b)$. A dotted line corresponds to those dimensions where a problem looks like the empty set, whereas a straight line represents those dimensions where it corresponds to the $\mathrm{HN}_{\mathbb{K}}$ problem.

LEMMA 1. *Let $\mathbb{K}$ be one of $\overline{\mathbb{Q}}$ or $\mathbb{C}$. Assume $\mathcal{P}_{\mathbb{K}} \neq \mathcal{NP}_{\mathbb{K}}$. Moreover let (a) and (b) be two sequences of natural numbers having an exponential gap. If both $L_{\mathbb{K}}(a)$ and $L_{\mathbb{K}}(b)$ belong to $\mathcal{NP}_{\mathbb{K}} \setminus \mathcal{P}_{\mathbb{K}}$, then both of them are not $\mathcal{NP}_{\mathbb{K}}$-complete.*

Proof. Assume, for example, that $L_{\mathbb{K}}(b)$ is $\mathcal{NP}_{\mathbb{K}}$-complete. Let M be a polynomial time machine performing a reduction from $L_{\mathbb{K}}(a)$ to $L_{\mathbb{K}}(b)$. Aside from the fact that (a) and (b) have an exponential gap, any instance y for problem $L_{\mathbb{K}}(a)$ of dimension n, $a^{2i+1} \leq n \leq a^{2i+2}$ for some $i \in \mathbb{N}$ big enough, must be mapped by M to an instance $M(y)$ of $L_{\mathbb{K}}(b)$ of dimension less than $b_{2i} \leq \log(n)$. This is due to the fact that between dimensions b_{2i} and b_{2i+1} the input- as well as the "yes"-set of $L_{\mathbb{K}}(b)$ by definition equal the empty set.

Applying a single exponential decision algorithm for $\mathrm{HN}_{\overline{\mathbb{Q}}}$ to $M(y)$ (cf. Theorem 2) will yield the right answer for $M(y)$ and hence for y in a polynomial number of steps with respect to $size(y)$. Thus it would follow that problem $L_{\mathbb{K}}(a)$ belongs to $\mathcal{P}_{\mathbb{K}}$, which contradicts the assumption. □

The rest of section 2 is devoted to explaining informally how to produce two sequences (v) and (w) such that the conditions listed below hold true:

(i) (v) and (w) have an exponential gap,
(ii) $L_{\overline{\mathbb{Q}}}(v)$ and $L_{\overline{\mathbb{Q}}}(w)$ both belong to class $\mathcal{NP}_{\overline{\mathbb{Q}}}$,
(iii) $L_{\overline{\mathbb{Q}}}(v)$ and $L_{\overline{\mathbb{Q}}}(w)$ both don't belong to $\mathcal{P}_{\overline{\mathbb{Q}}}$.

Clearly, if all three conditions are satisfied simultaneously, then according to Lemma 1 the main theorem over $\overline{\mathbb{Q}}$ will follow.

Let's explain more explicitely how to reach this goal.

The elements of sequences (v) and (w) are produced two at a time, through alternated steps. Each time a component v_{2j+2}, respectively, w_{2i} has been defined, condition (i) is enforced just by demanding the next component w_{2j+1}, respectively, v_{2i+1} to be at least exponentially far away. Of course this must be done by simultaneously respecting (ii) and (iii).

In order to guarantee (ii) and (iii) we take an effective enumeration $p_1, p_2, \ldots$ of all polynomial time machines over $\overline{\mathbb{Q}}$ together with one of $\overline{\mathbb{Q}}^{\infty}$. "Effective" here means that the enumeration can be produced by a BSS machine over $\overline{\mathbb{Q}}$. To do this we will use symbolic computations over $\overline{\mathbb{Q}}$ by representing algebraic numbers via minimal polynomials with rational coefficients. (In fact an ordinary Turing machine would suffice to perform this enumeration.)

The idea now is to define (v) and (w) such that for all $i \in \mathbb{N}$ machine p_i computes a false answer on at least one input of dimension n where $v_{2i-1} \leq n \leq v_{2i}$, respectively, $w_{2i-1} \leq n \leq w_{2i}$. Note that if v_{2i-1} or w_{2i-1} are already defined there will always exist a v_{2i}, respectively, w_{2i} with that property: on the clusters $\{v_{2i-1}, \ldots, v_{2i}\}$ and $\{w_{2i-1}, \ldots, w_{2i}\}$ the problems $L_{\overline{\mathbb{Q}}}(v)$ and $L_{\overline{\mathbb{Q}}}(w)$ look like $\mathrm{HN}_{\overline{\mathbb{Q}}}$. Thus if v_{2i} and w_{2i} are chosen large enough the polynomial time machine p_i will "believe" the above problems equal $\mathrm{HN}_{\overline{\mathbb{Q}}}$ except on a finite-dimensional space; consequently it must make a mistake.

The final task is to find the numbers v_{2i} and w_{2i} such that the resulting decision problems are members of $\mathcal{NP}_{\overline{\mathbb{Q}}}$. This can be gained in the following way: assume we already have defined $v_1, \ldots, v_{2i+1}$ as well as $w_1, \ldots, w_{2i}$ satisfying the following:

(∗) the machines $p_1, \ldots, p_i$ all fail on problems $L_{\overline{\mathbb{Q}}}(v)$ and $L_{\overline{\mathbb{Q}}}(w)$ for at least one input of dimension $\leq v_{2i+1}$.

In order to fool the next machine p_{i+1} on a cluster $\{v_{2i+1}, \ldots, v_{2i+2}\}$ choose v_{2i+2} large enough such that within v_{2i+2} many steps one can perform the following program:

- check condition (∗) to be true (this can be done inductively within v_{2i+1} steps);
- enumerate all possible inputs for $\mathrm{HN}_{\overline{\mathbb{Q}}}$ of dimension at least v_{2i+1};

- simulate p_{i+1} on these inputs;
- decide the solvability of the given system and compare with the result of p_{i+1};
- as soon as an input system is found on which p_{i+1} fails, the program stops.

Note that even though the decision procedure in between may use exponential time with respect to the size of the given systems, we circumvent this problem by enlarging the cluster until v_{2i+2} steps are sufficient to perform all the demanded operations. In a similar way the next cluster $\{w_{2i+1}, \ldots, w_{2i+2}\}$ is defined (ensuring $w_{2i+1} > 2^{v_{2i+2}}$).

This construction especially yields condition ii. The resulting $\mathcal{NP}_{\overline{\mathbb{Q}}}$ algorithms read as follows: given any polynomial system by a code in some $\overline{\mathbb{Q}}^n$, one has to check whether n belongs to one of the clusters. This can be done in polynomial time with respect to n. If "yes," the according $L_{\overline{\mathbb{Q}}}$-problem on $\overline{\mathbb{Q}}^n$ corresponds to $\mathrm{HN}_{\overline{\mathbb{Q}}}$, and a solution of the given system is guessed. If "no," reject the input.

Let's now present the formal construction of (v) and (w).

3. The main theorem over $\overline{\mathbb{Q}}$. The main result of this section is the following theorem.

THEOREM 3 (main theorem over $\overline{\mathbb{Q}}$). *Assume that $\mathcal{P}_{\overline{\mathbb{Q}}} \neq \mathcal{NP}_{\overline{\mathbb{Q}}}$. Then there are (we can construct) sequences (v) and (w) such that $L_{\overline{\mathbb{Q}}}(v)$ and $L_{\overline{\mathbb{Q}}}(w)$ are in $\mathcal{NP}_{\overline{\mathbb{Q}}} \backslash \mathcal{P}_{\overline{\mathbb{Q}}}$, and both of them are not $\mathcal{NP}_{\overline{\mathbb{Q}}}$-complete.*

3.1. Background: Computing over algebraic extensions. An algebraic extension of $\mathbb{Q}$ may be defined by a primitive element ζ or by the corresponding minimal polynomial

$$a(z) = z^d + a_{d-1}z^{d-1} + \cdots + a_0.$$

Elements of $\mathbb{Q}[\zeta]$ are equivalence classes of $\mathbb{Q}[z]$ modulo $a(z)$ and can be represented by a polynomial of degree $d-1$ with coefficients in $\mathbb{Q}$.

A general theory of computing over algebraic extensions can be found in the review paper by Loos [7].

Addition and subtraction can be performed as in $\mathbb{Q}[z]$. The product of elements represented by $b(z)$ and $c(z)$ is $b(z)c(z) \bmod a(z)$. It can be performed by multiplying $b(z)$ and $c(z)$ in the usual way and then performing the Euclidian algorithm.

The extended Euclidian algorithm can be used to compute division: if $b(z)c(z) + a(z)d(z) = 1$, then $c(\zeta) = d(\zeta)^{-1}$ in $\mathbb{Q}[\zeta]$.

Checking if $b(\zeta)$ is equal to zero (or not) is trivial.

Therefore, a machine over $\mathbb{Q}$, *without order*, can simulate a given machine p over an extension given by some $a(z)$. It is also possible to check that $a(z)$ is irreducible, for example, using the Lenstra–Lenstra–Lovász algorithm (cf. [9]).

In this paper, we will need to perform computations in $\mathbb{Q}[\zeta_1, \zeta_2]$. Given minimal polynomials of ζ_1 and ζ_2 over $\mathbb{Q}$, one may find (algorithmically) a primitive element and a minimal polynomial of $\mathbb{Q}[\zeta_1, \zeta_2]$. See [7, Theorem 6].

Computation with inputs in $\overline{\mathbb{Q}}$ can be simulated using the very same techniques.

3.2. Timing a machine. One important ingredient to prove Theorem 3 is the simulation of a machine for a given number of steps. Thus we have to deal with BSS machines which additionally are able to count the number of steps they perform. The construction of such timed machines is straightforward and can be performed as in the discrete setting. Therefore, without loss of generality we will assume machines to be timed whenever necessary.

3.3. Enumeration. Let $A = \{A_0, \ldots\}$ and $B = \{B_0, \ldots\}$ be countable sets. The triangular enumeration of $A \times B$ is

$$(A_0, B_0), (A_1, B_0), (A_1, B_1), (A_2, B_0) \ldots .$$

We will enumerate $\mathbb{Q}$, $\mathbb{Q}^2$, $\mathbb{Q}^3 = \mathbb{Q}^2 \times \mathbb{Q}$ this way. We may enumerate $\mathbb{Q}^\infty$ in the same way:

$$(0, \ldots ...), (q_1, 0, \ldots), (q_1, q_1, 0, \ldots), (q_2, 0 \ldots),$$

where $0, q_1, q_2, \ldots$ enumerates $\mathbb{Q}$.

The algebraic closure $\overline{\mathbb{Q}}$ of the rationals may be enumerated by enumerating all extensions of $\mathbb{Q}$ through the minimal polynomials, together with the elements of the extension. One has to pay attention to the fact that the polynomial defining an extension should be irreducible.

One can enumerate $\overline{\mathbb{Q}}^\infty$ in the same way. Finally, later on we need to enumerate BSS machines over $\overline{\mathbb{Q}}$ together with polynomial time bounds $n^k, k \in \mathbb{N}$. This will be done as indicated above by taking A to be the set of all BSS machines over $\overline{\mathbb{Q}}$ and $B := \mathbb{N}$.

3.4. A machine to fool a polynomial time machine. Under our general hypothesis $\mathcal{P}_{\overline{\mathbb{Q}}} \neq \mathcal{NP}_{\overline{\mathbb{Q}}}$, we will construct a machine M over $\overline{\mathbb{Q}}$ that will take as input

- an integer $m \in \mathbb{N}$,
- a list $p \in (\mathbb{Q}^{\deg a})^\infty$ representing (possibly) a machine $\tilde{p}$ over an extension $\mathbb{Q}[\zeta]$;
- an integer $r \in \mathbb{N}$ representing a polynomial time bound $\text{size}(x)^r$ for $\tilde{p}$;
- an integer $s \geq m$ representing a total running time bound for M.

Machine M is used for the construction of the two decision problems we are looking for. The purpose of this machine is the following.

Since $\text{HN}_{\overline{\mathbb{Q}}}$ is $\mathcal{NP}_{\overline{\mathbb{Q}}}$-complete, it remains complete when restricted to inputs exceeding the fixed dimension m. The list p is supposed to represent a machine that is a candidate for deciding this restricted $\text{HN}_{\overline{\mathbb{Q}}}$ in polynomial time bound $\text{size}(x)^r$, where x is an input of size $\geq m$.

Thus there must exist an input $\bar{x}, size(\bar{x}) \geq m$ such that $p(\bar{x})$ does not provide the right answer to the $\text{HN}_{\overline{\mathbb{Q}}}$ problem for input $\bar{x}$. Machine M will check whether both such $\bar{x}$ exists up to dimension s and can be found within s steps.

Let us explain more precisely the way M works: consider an input (m, p, r, s) and $s \geq m$. We assume M is timed. As soon as the procedure described below has performed more than s steps, machine M stops. In that case it returns the answer TIMEOUT indicating that up to dimension s no input $\bar{x}$ satisfying the above conditions exists.

As long as fewer than s steps have been executed M behaves as follows.

1. Enumerate all possible inputs belonging to $\overline{\mathbb{Q}}^\infty$ of size at least m; let x be the actually enumerated element.
2. Simulate machine p for at most $size(x)^r$ many steps on input x.
3. Simulate a decision procedure for $\text{HN}_{\overline{\mathbb{Q}}}$ on input x.
4. If 2 and 3 yield the same answer, goto 1 and take the next element of the enumeration; if 2 and 3 yield different answers, output c; here c denotes the maximum among $size(x)$ and the number of steps already performed by M.

Note that $M(m, p, r, s) =$TIMEOUT indicates that it is impossible to fool (p, r) in the above sense on $\text{HN}_{\overline{\mathbb{Q}}}$ for an input of size at least m and at most s. On the

other hand, if M returns $c \leq s$ there exists an $\bar{x} \in \overline{\mathbb{Q}}^c$ of size at least m such that $p(\bar{x})$ differs from the correct answer of $\mathrm{HN}_{\overline{\mathbb{Q}}}(\bar{x})$, and this difference can be shown by algorithm M in $\leq c$ steps.

We finally remark that in step 4 different answers comparing the simulation under 2 and 3 can be obtained either if the computation of $p(x)$ cannot be finished within $size(x)^r$ many steps or both simulations are completed but with different results.

Remark. Because of our assumption, $\mathcal{P}_{\overline{\mathbb{Q}}} \neq \mathcal{NP}_{\overline{\mathbb{Q}}}$ machine M will always end up with a suitable dimension $c \leq s$ if s is large enough.

3.5. A machine to produce (v) and (w) up to s. Next we will use machine M to construct another machine N that will produce sequences (v) and (w) in the following way.

Given an input $s \in \mathbb{N}$, all $v_j < s$ and $w_j < s$ will be computed. Sequences (v) and (w) do not depend on s. Furthermore, the following statements hold.

- Sequences (v) and (w) have an exponential gap.
- Machines $p_1, \ldots, p_i$ are fooled (in the sense of the previous section) for inputs of size in

$$\{v_1, \ldots, v_2\} \cup \{v_3, \ldots, v_4\} \cup \cdots \cup \{v_{2i-1}, \ldots, v_{2i}\}$$

  and also for inputs of size in

$$\{w_1, \ldots, w_2\} \cup \{w_3, \ldots, w_4\} \cup \cdots \cup \{w_{2i-1}, \ldots, w_{2i}\}.$$

- The running time of N is a polynomial in s.

Machine N is defined as explained now.

As an input it gets a natural number s; for the first cluster of sequence (v) we set $v_1 := 1$. As long as those values v_{2i} and w_{2i} already computed do not exceed s, the following algorithm is performed for $i = 0, 1, \ldots$.

1. Enumerate all BSS machines over $\overline{\mathbb{Q}}$ together with a natural number representing a polynomial time bound; let (p, r) be the actually enumerated element.
2. Let v_{2i+2} be the result of $M(v_{2i+1}, p, r, s)$.
3. If $v_{2i+2} = \mathrm{TIMEOUT}$ then algorithm N terminates.
4. Otherwise output v_{2i+2} as the next element of those clusters defining (v); compute the starting point w_{2i+1} of the next cluster for sequence (w) as $w_{2i+1} := 2^{v_{2i+2}} + 1$.
5. If $w_{2i+1} > s$ then algorithm N terminates.
6. Otherwise output w_{2i+1}; let w_{2i+2} be the result of $M(w_{2i+1}, p, r, s)$.
7. If $w_{2i+2} = \mathrm{TIMEOUT}$ then algorithm N terminates.
8. Otherwise output w_{2i+2} as the next element of those clusters defining (w); compute the starting point v_{2i+3} of the next cluster for sequence (v) as $v_{2i+3} := 2^{w_{2i+2}} + 1$.
9. If $v_{2i+3} > s$ then algorithm N terminates.
10. Otherwise output v_{2i+3}; increase i and goto 1.

The comparisons such as $v_{2i+3} > s$ can be performed without any order relation available, because it is assumed that the values involved are integers. The time bound for each comparison is $O(s)$.

Exponential gap and the fact that machines $p_1, \ldots, p_i$ are fooled follows from the construction of machine M.

The running time of each call to M is bounded by s, so the time bound $(O(s^2))$ follows.

3.6. Proof of Theorem 3. According to the previous subsections, given s one may compute in polynomial time (in s) the largest v_i and $w_i < s$. This implies the following lemma holds.

LEMMA 2. *$L_{\overline{\mathbb{Q}}}(v)$ (respectively, $L_{\overline{\mathbb{Q}}}(w)$) is in $\mathcal{NP}_{\overline{\mathbb{Q}}}$.*

Proof. Given input x, set $s = size(x)$. We may use machine N to compute the largest v_i (respectively, w_i) such that $v_i < s$ (respectively, $w_i < s$).

If i is even, $L_{\overline{\mathbb{Q}}}(v) \cap (\overline{\mathbb{Q}}^s, \overline{\mathbb{Q}}^s) = (\emptyset, \emptyset)$, so we may answer NO in polynomial time.

If i is odd, $L_{\overline{\mathbb{Q}}}(v) \cap (\overline{\mathbb{Q}}^s, \overline{\mathbb{Q}}^s) = \mathrm{HN}_{\overline{\mathbb{Q}}} \cap \overline{\mathbb{Q}}^{size(x)}$. This is known to be in $\mathcal{NP}_{\overline{\mathbb{Q}}}$. ∎

LEMMA 3. *$L_{\overline{\mathbb{Q}}}(v)$ (respectively, $L_{\overline{\mathbb{Q}}}(w)$) is not in $\mathcal{P}_{\overline{\mathbb{Q}}}$.*

Proof. This follows from the above construction: eventually, every machine p will appear in machine N, and it will be fooled at some time, inside a $\mathrm{HN}_{\overline{\mathbb{Q}}}$-cluster. Therefore, $L_{\overline{\mathbb{Q}}}(v)$ (respectively, $L_{\overline{\mathbb{Q}}}(w)$) cannot belong to $\mathcal{P}_{\overline{\mathbb{Q}}}$. ∎

According to Lemma 1 we proved that $L_{\overline{\mathbb{Q}}}(v)$ and $L_{\overline{\mathbb{Q}}}(w)$ both are not $\mathcal{NP}_{\overline{\mathbb{Q}}}$-complete. We also know that $L_{\overline{\mathbb{Q}}}(v) \in \mathcal{NP}_{\overline{\mathbb{Q}}} \backslash \mathcal{P}_{\overline{\mathbb{Q}}}$. Therefore, Theorem 3 is proved.

4. The main theorem over $\mathbb{C}$. We are now ready to prove our main theorem.

MAIN THEOREM. *Assume $\mathcal{P}_{\mathbb{C}} \neq \mathcal{NP}_{\mathbb{C}}$. Then there is a decision problem in $\mathcal{NP}_{\mathbb{C}} \setminus \mathcal{P}_{\mathbb{C}}$ that is not $\mathcal{NP}_{\mathbb{C}}$-complete.*

Proof. Let the sequences (v) and (w) be as in Theorem 3. We extend the corresponding problems $L_{\overline{\mathbb{Q}}}(v)$ and $L_{\overline{\mathbb{Q}}}(w)$ to the complex numbers. This will give $L_{\mathbb{C}}(v)$ and $L_{\mathbb{C}}(w)$, respectively. Note that both $L_{\mathbb{C}}(v)$ and $L_{\mathbb{C}}(w)$ are closely related to their $\overline{\mathbb{Q}}-$ counterparts: intersecting the input-, respectively, the "yes"-set of each of them with $\overline{\mathbb{Q}}^{\infty}$ will give exactly the input-, respectively, the "yes"-set of the corresponding $L_{\overline{\mathbb{Q}}}$-problem. Since this is obvious for the input sets, this follows for the "yes"-sets by Hilbert's Nullstellensatz; see [2, Chapter 6].

Now assume the class $\mathcal{NP}_{\mathbb{C}} \setminus \mathcal{P}_{\mathbb{C}}$ consists of $\mathcal{NP}_{\mathbb{C}}$-complete problems only. Both $L_{\mathbb{C}}(v)$ and $L_{\mathbb{C}}(w)$ belong to $\mathcal{NP}_{\mathbb{C}}$.

Given $x \in \mathbb{C}^n$, in order to check the cluster to which n belongs, the $\mathcal{P}_{\overline{\mathbb{Q}}}$ algorithm of section 3 can also be performed over $\mathbb{C}$. Moreover, $\mathrm{HN}_{\mathbb{C}}$ is in $\mathcal{NP}_{\mathbb{C}}$.

Furthermore, assume $L_{\mathbb{C}}(v)$ belongs to $\mathcal{P}_{\mathbb{C}}$. Then there is a polynomial time machine M deciding $L_{\mathbb{C}}(v)$. According to the transfer principle Theorem 1 there exists a BSS machine M' over $\overline{\mathbb{Q}}$ and a constant c s.t. M' solves $L_{\overline{\mathbb{Q}}}(v)$ in time $\leq T_M(y)^c \ \forall y \in \overline{\mathbb{Q}}^{\infty}$. This implies $L_{\overline{\mathbb{Q}}}(v) \in \mathcal{P}_{\overline{\mathbb{Q}}}$ in contradiction to Theorem 3.

Thus our second assumption is wrong and it follows $\mathcal{NP}_{\mathbb{C}}$-completeness of $L_{\mathbb{C}}(v)$; the same reasoning clearly holds true for $L_{\mathbb{C}}(w)$. But (v) and (w) have an exponential gap. Consequently according to Lemma 1 they are not $\mathcal{NP}_{\mathbb{C}}$-complete. Hence the introductory assumption was wrong, which finishes the proof. ∎

Let us conclude with some final remarks. The problems $L_{\mathbb{C}}(v)$ and $L_{\mathbb{C}}(w)$ used to establish the main theorem are quite artificial; the dimensions for which they coincide with $\mathrm{HN}_{\mathbb{C}}$ are chosen just in order to fool all polynomial time machines over $\overline{\mathbb{Q}}$ according to a given enumeration and not by a "natural" condition. We consider it an interesting task to figure out more natural problems neither belonging to $\mathcal{P}_{\mathbb{C}}$ nor being $\mathcal{NP}_{\mathbb{C}}$-complete. Here we just suggest one further problem which seems to be promising from that point of view.

Considering the $\mathcal{NP}_{\mathbb{C}}$-completeness proof of $\mathrm{HN}_{\mathbb{C}}$ in [3] it turns out to be sufficient if all polynomials of the given system depend on three variables only.

Let $2-\mathrm{HN}_{\mathbb{C}}$ be the problem of deciding solvability of such a system, if all involved polynomials just depend on at most two unknowns.

On the one hand, the completeness proof for $\mathrm{HN}_{\mathbb{C}}$ lets $2-\mathrm{HN}_{\mathbb{C}}$ seem unlikely to be $\mathcal{NP}_{\mathbb{C}}$-complete, too: reducing higher-degree systems to such a degree of two uses substitutions intrinsically including equations with three unknowns. We don't see any way to proof $\mathcal{NP}_{\mathbb{C}}$-completeness of $2-\mathrm{HN}_{\mathbb{C}}$. On the other hand it can be shown that problems like complex subset-sum (given $z_1,\ldots,z_n\in\mathbb{C}$, does there exist a subset $S\subset\{1,\ldots,n\}$ s.t. $\sum_{i\in S} z_i = 1$?) can be reduced to $2-\mathrm{HN}_{\mathbb{C}}$ in polynomial time. Note that the discrete version of subset-sum (rational inputs) is $\mathcal{NP}$-complete in the Turing model. Since there is no polynomial time algorithm known so far for complex subset-sum, this will give good reason to conjecture $2-\mathrm{HN}_{\mathbb{C}}\notin\mathcal{P}_{\mathbb{C}}$. Thus $2-\mathrm{HN}_{\mathbb{C}}$ seems to be a reasonable candidate of a noncomplete problem in $\mathcal{NP}_{\mathbb{C}}\setminus\mathcal{P}_{\mathbb{C}}$ (as well as complex subset-sum).

Finally consider the question treated in this paper for real BSS machines. The above proof fails: there is no transfer principle available in order to reduce the uncountable real case to a countable situation. However, over $\mathbb{R}$ one wouldn't have to pay attention to the computability of the sequences (v) and (w). The presence of the order relation allows us to code any such sequence in a single real number from which its components can be easily decoded (see [3]). Nevertheless, so far we see no way to build up sequences (v) and (w) such that the resulting problems $L_{\mathbb{R}}(v)$ and $L_{\mathbb{R}}(w)$ would force all (uncountable many) $\mathcal{P}_{\mathbb{R}}$-machines to fail.

The situation over the real numbers will be treated more intensively in the forthcoming paper [1].

Acknowledgment. Thanks are due to two unknown referees, whose comments helped to improve readability of an earlier version of this paper.

REFERENCES

[1] S. Ben-David, K. Meer, and C. Michaux, *A note on non-complete problems in* $\mathcal{NP}_{\mathbb{R}}$, 1997, preprint.

[2] L. Blum, F. Cucker, M. Shub, and S. Smale, *Complexity and Real Computation*, Springer-Verlag, Berlin, New York, to appear.

[3] L. Blum, M. Shub, and S. Smale, *On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines*, Bull. Amer. Math. Soc., 21 (1989), p. 1.

[4] A. L. Chistov and D. Y. Grigoriev, *Complexity of quantifier elimination in the theory of algebraically closed fields*, Lecture Notes in Comp. Sci. 176, Springer-Verlag, New York, 1984, pp. 17–31.

[5] D. Y. Grigoriev, *The complexity of the decision problem for the first order theory of algebraically closed fields*, Math. USSR Izvestiya, 29 (1987), pp. 459–475.

[6] L. Caniglia, A. Galligo, and J. Heintz, *Some new effectivity bounds in computational geometry*, Lecture Notes in Comput. Sci. 357, Springer-Verlag, New York, 1988, pp. 131–151.

[7] R. Loos, *Computing in algebraic extensions*, in Computer Algebra, Symbolic and Algebraic Computation, 2nd ed., B. Buchberger, R. Loos, and R. Albrecht, eds., Springer-Verlag, Wien, 1983, pp. 173–188.

[8] R. Ladner, *On the structure of polynomial time reducibility*, J. Assoc. Comput. Mach., 22 (1975), pp. 155–171.

[9] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann., 261 (1982), pp. 515–534.

[10] K. Meer and C. Michaux, *A Survey on Real Structural Complexity Theory*, Bull. Belg. Math. Soc. Simon Stevin, 4 (1997), pp. 113–148.

[11] J. Renegar, *On the computational complexity and geometry of the first-order theory of the reals,* I–III, J. Symbolic Comput., 13 (1992), pp. 255–352.

[12] U. Schöning, *A uniform approach to obtain diagonal sets in complexity classes*, Theoret. Comput. Sci., 18 (1982), pp. 95–103.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 36–56

WEIGHTED NP OPTIMIZATION PROBLEMS: LOGICAL DEFINABILITY AND APPROXIMATION PROPERTIES*

MARIUS ZIMAND†

Abstract. Extending a well-known property of NP optimization problems in which the value of the optimum is guaranteed to be polynomially bounded in the length of the input, it is observed that, by attaching weights to tuples over the domain of the input, all NP optimization problems admit a logical characterization. It is shown that any NP optimization problem can be stated as a problem in which the constraint conditions can be expressed by a Π_2 first-order formula. The paper analyzes the weighted analogue of all syntactically defined classes of optimization problems that are known to have good approximation properties in the nonweighted case. Dramatic changes occur when negative weights are allowed.

Key words. NP optimization problems, logical definability, approximation properties, multiprover interactive systems.

AMS subject classifications. 68Q15, 68Q25.

PII. S0097539795285102

1. Introduction.

Recent years have seen considerable progress in the understanding of the approximation properties of NP optimization problems. Two approaches have been the main cause for the new advancements: (a) the development of a robust model for optimization problems based on the logical definability of this type of problem, and (b) the characterization of NP in terms of multiprover interactive proof (MIP) or probabilistically checking proof (PCP) systems which allowed new reductions that preserve approximation properties between NP complete problems. The first approach was initiated by Papadimitriou and Yannakakis [PY91] who introduced the classes MAX SNP and MAX NP (later called MAX Σ_0 and MAX Σ_1 by Kolaitis and Thakur [KT94] in an attempt to uniformize notation). A maximization problem is in MAX SNP (MAX NP) if: (1) the input I is viewed as a finite structure, (2) the set of feasible solutions of an input I is given by the set of finite structures S having the same domain as I and satisfying a $\Sigma_0(\Sigma_1)$ first-order formula having a tuple of free variables, and (3) the objective function maps each feasible solution S to the cardinality of tuples over the domain of I satisfying ϕ when substituting the tuple of free variables and when the relation symbols in ϕ are interpreted as the relations of I and S. MAX SNP and MAX NP contain many natural problems, including MAX 3SAT, MAX CUT, and MAX SAT, and have the nice property that all the problems in these classes are approximable by polynomial-time algorithms with constant approximation ratio. Moreover, Khanna et al. [KMSV94] have shown that the class of NP optimization problems admitting polynomial-time constant ratio approximating algorithms coincides with the class of problems that are reducible to MAX SNP via a certain type of reduction that preserves approximation properties. It is proved in [ALM$^+$92],

*Received by the editors April 26, 1995; accepted for publication (in revised form) October 30, 1996; published electronically June 15, 1998.

http://www.siam.org/journals/sicomp/28-1/28510.html

†School of Computer & Applied Sciences, Georgia Southwestern State University, Americus, GA 31709 (zimand@gswrs6k1.gsw.peachnet.edu). This author was supported in part by grants NSF-CCR-8957604, NSF-INT-9116781/JSPS-ENG-207, and NSF-CCR-9322513 and by the Romanian Department of Education and Science grant, 4975-92. A preliminary version appeared in *Proc. of the* 10*th IEEE Symp. on Structure in Complexity Theory*, 1995, pp. 12–28. This work was done in part while the author was visiting the University of Rochester, Rochester, NY 14627.

via a reduction from a PCP, that no complete problem in MAX SNP can be approximated with arbitrarily small constant approximation ratio (i.e., have PTAS), unless NP = P. Many important minimization problems, like VERTEX COVER, DOMINATING SET, STEINER TREE, and TRAVELING SALESMAN with edge weights 1 and 2, and SHORTEST COMMON SUPERSTRING, are MAX SNP hard via reductions that preserve approximation properties (see Johnson [Joh92] for references and an excellent survey) and, thus, have the same approximation properties. Hence, the approximation capabilities of a large number of natural, important optimization problems have been fully understood by the joint contribution of the two approaches (a) and (b). However, MAX SNP and MAX NP are far from being the whole story. Panconesi and Ranjan [PR93] showed that MAX CLIQUE is not in MAX NP. This problem belongs to the class MAX Π_1 in which the feasible solutions are characterized by Π_1 formulae. By imposing a syntactical restriction on ϕ, Panconesi and Ranjan identified, inside MAX Π_1, the class RMAX(2) containing MAX CLIQUE and having the property that all problems in the class are self-improvable, i.e., the existence of approximation polynomial-time algorithms with a constant ratio of approximation implies the existence of PTAS. It has been shown, again by a reduction from a certain type of PCP, that there is no constant ratio, polynomial approximation algorithm for MAX CLIQUE, unless NP = P. Therefore, no complete problem in RMAX(2) has PTAS, unless NP = P. In a later development, Kolaitis and Thakur undertook in [KT94] a comprehensive investigation of the logical characterizations of NP optimization problems. Considering an arbitrary number of alternations of blocks of quantifiers, they introduced the classes MAX Π_n, MAX Σ_n, MIN Π_n, MIN Σ_n, $n \geq 0$. They showed that all NP optimization problems in which the value of the optimum is polynomially bounded in the length of the input are in MAX Π_2, in the case of maximization problems, and in MIN Π_1, in the case of minimization problems. Moreover, they proved that all polynomially bounded NP maximization and minimization problems can be placed in one of the levels of the following proper hierarchies: MAX $\Sigma_0 \subsetneqq$ MAX $\Sigma_1 \subsetneqq$ MAX Π_1 = MAX $\Sigma_2 \subsetneqq$ MAX Π_2, for maximization problems, and MIN Σ_0 = MIN $\Sigma_1 \subsetneqq$ MIN Π_1 = MIN Σ_2 = MIN Π_2, for minimization problems. Kolaitis and Thakur have also introduced and investigated in [KT94] and [KT95] the classes MAX FΣ_n, MAX FΠ_n, MIN FΣ_n, MIN FΠ_n, $n \geq 1$, which are closely related to the above classes and offer a more natural logical description for some NP polynomially bounded optimization problems. This time the formula characterizing the set of feasible solutions is closed (i.e., it has no free variables) and the objective function is defined to be the number of tuples satisfying a specified relation from the structure S over which we maximize or minimize. They identified two syntactically defined classes, MIN F$^+\Pi_1$ and MIN F$^+\Pi_2(1)$, containing many natural important problems and having good approximation properties. Thus, MIN F$^+\Pi_1$ contains VERTEX COVER and many minimization node and edge deletion problems. All the problems in this class have polynomial-time algorithms with constant approximation ratio. MIN F$^+\Pi_2(1)$ contains SET COVER, DOMINATING SET, HITTING SET, and other natural problems. All the problems in this class have polynomial-time approximation algorithms with the approximation ratio bounded by $\log n$, where n is the length of the input. A further step was taken by Behrendt, Compton, and Grädel [BCG92], who considered more powerful logics obtained by adding the least fixpoint operator to the first-order syntax. It is observed that MAX Σ_1^{FP}, which is just MAX Σ_1 extended with the least fixpoint operator, continues to be approximable within a constant factor. In [BCG92] and [GM93], limit laws are identified for many

of the above classes. These laws, similar to the 0-1 laws for various logics (see the survey paper [Com88]) constitute useful necessary criteria for the expressibility of a problem in the restricted syntax which define the respective classes.

In spite of the new techniques that have emerged in recent years, with one important exception, there has been no thorough investigation of arbitrary NP optimization problems, i.e., problems in which the optimum value is no longer polynomially bounded in the length of the input. Since these problems usually arise when numerical weights are added to the input data, we call them *weighted NP optimization problems*. There is no need to argue about the large percentage of problems of this type that appear in real applications. The exception paper alluded to above is again the work of Papadimitriou and Yannakakis [PY91]. They have considered the variants of MAX SNP and MAX NP in which positive weights are attached to the tuples over the domain of the input structure that can be substituted for the free variables $\vec{x}$ and showed that these more general classes continue to have polynomial-time algorithms with constant ratio.

In this work, we undertake a more comprehensive examination of weighted NP optimization problems. We consider the classes weight-MAX $\mathrm{F}\Sigma_n$, weight-MAX $\mathrm{F}\Pi_n$, weight-MIN $\mathrm{F}\Sigma_n$, weight-MIN $\mathrm{F}\Pi_n$, $n \geq 1$, which are just the weighted variants of the MAX $\mathrm{F}\Sigma_n$, MAX $\mathrm{F}\Pi_n$, MIN $\mathrm{F}\Sigma_n$, and MIN $\mathrm{F}\Pi_n$ classes from [KT94] and [KT95]. For example, a problem is in weight-MAX $\mathrm{F}\Pi_2$ if: (1) the set of feasible solutions for an input structure $I = (D_I, R_I)$ is given by the finite structures S having the same domain as I and the relation $S_1, S_2, \ldots, S_p$ and satisfying a formula of the form $\forall \vec{x}\ \exists \vec{y}\ \phi(\vec{x}, \vec{y}, R_I, S_1, \ldots, S_p)$ with ϕ closed and quantifier-free, (2) each m-tuple $(x_1, \ldots, x_m)$ over the domain of I has a real-valued weight, where m is the arity of S_1, and (3) the objective function is to maximize over all structures S as above the weight of tuples in S_1. If the weights are positive, then the problem is in weight(+)-MAX $\mathrm{F}\Pi_2$. We notice that all NP optimization problems, not just the polynomially bounded ones, admit a logical characterization. More precisely, all NP maximization (minimization) problems are in weight-MAX $\mathrm{F}\Pi_2$ (weight-MIN $\mathrm{F}\Pi_2$). It is easy to note that all problems in weight-MAX $\mathrm{F}\Sigma_1$ and weight(+)-MAX $\mathrm{F}\Sigma_1$ are solvable in polynomial time, and weight-MAX $\mathrm{F}\Sigma_2$ and weight(+)-MAX $\mathrm{F}\Sigma_2$ can be reduced in a way that preserves approximation properties to weight-MAX $\mathrm{F}\Pi_1$ and, respectively, to weight(+)-MAX $\mathrm{F}\Pi_1$. Similar properties hold for the analogue minimization classes and, consequently, these classes are less interesting from the point of view of approximation properties. From the syntactical point of view, the weight(+) classes satisfy the diagram in Figure 1, which is identical with the one satisfied by their non-weighted analogues. For the classes with arbitrary weights we only know the trivial relations represented in Figure 1. Making these relations more precise remains an open problem.

For all classes $\mathcal{C}$ earlier identified as having polynomial-time approximation algorithms with guaranteed low approximation ratio, we analyze the weight-$\mathcal{C}$ and weight(+)-$\mathcal{C}$ variants. We consider $\mathcal{C} \in \{\text{MAX SNP}, \text{MAX NP}, \text{MAX SNP}(\pi), \text{MAX } \mathrm{F}\Pi_1, \text{MAX } \mathrm{F}^{+}\Pi_2(1)\}$. (MAX SNP($\pi$) is a subclass of MAX SNP in which the structure S over which the maximum is searched is required to be a permutation of the domain of the input structure.)

In all cases, weight(+)-$\mathcal{C}$ continues to have the same approximation properties as $\mathcal{C}$, and weight-$\mathcal{C}$ fails to do likewise unless very unlikely hypotheses hold. Table 1 summarizes the approximation properties of the classes analyzed in this paper.

The notation is standard and most of it is explicitly introduced in the text. We

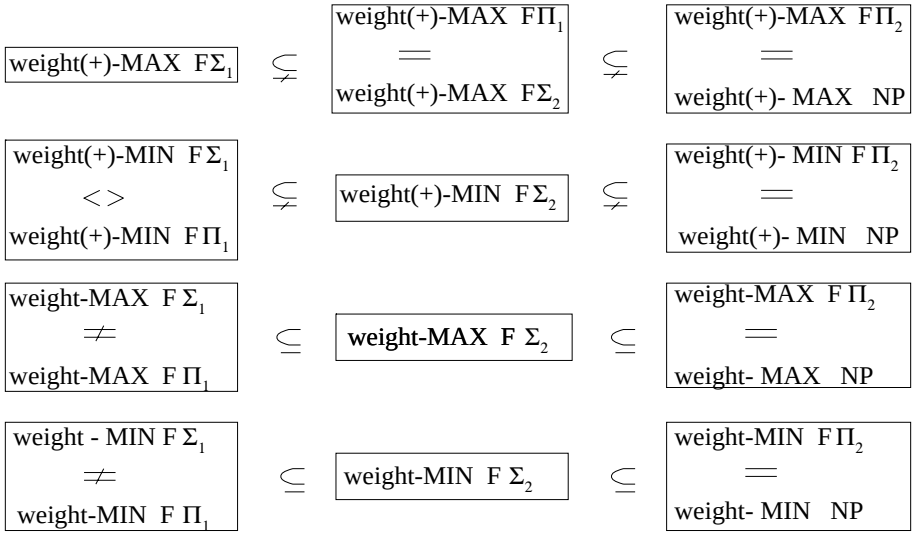


FIG. 1. *The relations between the syntactically defined* NP *optimization classes. (<> denotes incomparability.)*

TABLE 1
Approximation properties of the classes with good approximation properties in the nonweighted case.

Syntax	Positive weights	Arbitrary weights
MAX SNP	const. approximable	not approx. with ratio $< n^{1/4}$ unless P = NP
MAX NP	const. approximable	not approx. with ratio $< n^{1/4}$ unless P = NP
MAX SNP(π)	const. approximable	not approx. with ratio $2^{\log^{\mu} n}$, for some $\mu > 0$, unless NP $\subseteq$ DTIME$[2^{\log^{O(1)} n}]$
MIN $F^+\Pi_1$	const. approximable	not approx. with ratio 2^{n^q}, any q, unless P = NP
MIN $F^+\Pi_2(1)$	log. approximable	not approx. with ratio 2^{n^q}, any q, unless P = NP

note here only that Σ^* is the set of finite binary strings and $\Sigma^{=n}$ is the set of binary strings of length n; if $x \in \Sigma^*$, then $|x|$ is the length of string x; if S is a set, then $|S|$ is the cardinality of S; and, finally, if y is a real number, then $|y|$ is the modulus of y. $\tilde{\text{P}}$ denotes DTIME$[2^{\log^{O(1)} n}]$ and $\log n$ is the integer part of $\log_2 n$.

2. Logical definability. The elements which define an optimization problem A are:

(1) a set $\mathcal{I}_A$ of input instances; we assume that this set can be recognized in polynomial time,
(2) for each $I \in \mathcal{I}_A$, a set $\mathcal{F}_A(I)$ of feasible solutions associated to each input instance; we assume that each element in $\mathcal{F}_A(I)$ has size polynomially bounded in the size of I, and
(3) an objective function f_A which maps to real numbers each pair (I, J) with $I \in \mathcal{I}_A$ and $J \in \mathcal{F}_A(I)$; we assume that this function is computable in polynomial time. There is also a default value for the cases when the set of feasible solutions is empty.

If the objective function takes only nonnegative values then A is called a positive optimization problem. Given an instance $I \in \mathcal{I}_A$, the goal is to find $\text{opt}_{J \in \mathcal{F}_A(I)} f_A(I, J)$ or output the default value in case $\mathcal{F}_A(I)$ is empty, where opt is max or min depending on what kind of an optimization we have. It is convenient to denote $\text{opt}_{J \in \mathcal{F}_A(I)}\ f_A(I, J)$ by $\text{opt}_A(I)$. A max (min) optimization problem A is an NP optimization problem if the following associated decision problem B is in NP.

Instance: An input instance $I \in \mathcal{I}_A$ and $k \in \mathbf{Z}$.
Question: Does there exist a feasible solution $J \in \mathcal{F}_A(I)$ such that $f_A(I, J) \geq k$ ($f_A(I, J) \leq k$, in the case of a min problem)?

Kolaitis and Thakur [KT94] have shown that each NP optimization problem in which the optimum is polynomially bounded in the size of an encoding of the input instance can be syntactically described as an optimization problem in which the goal is to find the max (or min) cardinality of a relation which together with some other relations satisfies a given first-order formula. Thus the objective function is the cardinality of tuples satisfying a relation of some specified arity and the set of feasible solutions is given by the structures satisfying a first-order formula. In order to make the above statements more precise we need some standard definitions from descriptive computational complexity that we introduce in a simplified manner which mixes together syntactical and semantical notions. For a rigorous treatment see [Fag74], [Gra84], [Imm89], [Lyn82].

DEFINITION 2.1. *A finite type is a finite sequence of nonnegative integers. Given a finite type $T = (n_1, n_2, \ldots, n_k)$, a finite T-structure is a $(k+1)$-tuple $F = (X, f_1, f_2, \ldots, f_k)$, where X is a nonempty finite set called the domain of the structure F and, for all i, f_i is a relation over X of arity n_i.*

Input instances of problems can naturally be viewed as finite structures of some finite type. For example, graphs are finite structures of the form (V, E), where V is the domain (the set of nodes) and E is a relation of arity 2 (the set of edges). Boolean formulae in CNF are finite structures of the form $(\{x_1, x_2, \ldots, x_n, c_1, \ldots, c_m\}, P, N)$, where $\{x_1, x_2, \ldots, x_n, c_1, \ldots, c_m\}$ is the set of variables and clauses and P and N are relations of arity 2 such that $P(x_i, c_j)(N(x_i, c_j))$ has the significance that variable x_i appears positively (negatively) in clause c_j. The following well-known theorem of Fagin [Fag74] characterizes the class of NP decision problems in logical terms.

THEOREM 2.2 (see [Fag74]). *Let T be a finite type. A set L of finite T-structures is in* NP *if and only if there exists a finite type $\mathcal{S}$ and a quantifier-free first-order closed formula ϕ such that for all input structures I:*

$$I \in L \leftrightarrow \exists S \quad \forall \vec{x} \quad \exists \vec{y}\ \phi(\vec{x}, \vec{y}, I, S),$$

where S is a finite structure of type $\mathcal{S}$ having the same domain as I and $\vec{x}$ and $\vec{y}$ are tuples of variables ranging over I's domain.

The reader should be aware that the notation in the above statement of Theorem 2.2 is highly abusive. While the "I" in the left-hand side denotes a whole finite structure, the "I" and both "S"'s in the right-hand side are a shorthand notation for the relations of the corresponding structures denoted by the same symbols. We have implicitly assumed (by notation) that ϕ is compatible with I and S; i.e., for each relation symbol occurring in ϕ there is a correspondent relation in I or S and there is agreement on arities. Also, in $\forall \vec{x}$ and $\exists \vec{y}$, the quantifier is applied to all the components in the tuple $\vec{x}$ and, respectively, $\vec{y}$. These conventions will be used

throughout the rest of the paper. As an example, if I is a finite structure describing a CNF boolean formula as above, then

$$I \in \text{SAT} \leftrightarrow \exists T \quad \forall c \quad \exists x \; [(P(x,c) \wedge T(x)) \vee (N(x,c) \wedge \neg T(x))].$$

As mentioned above, Kolaitis and Thakur [KT94] have shown a similar property for polynomially bounded NP optimization problems.

THEOREM 2.3 (see [KT94]). *Let A be a polynomially bounded* NP *optimization problem. There exists a finite type $\mathcal{S}$ and a closed first-order formula ϕ such that for each input structure I*

$$\text{opt}_A(I) = \text{opt}_S \; \{|S_1| \; : \; \phi(\vec{x}, I, S)\},$$

where S is a finite structure of type $\mathcal{S}$ with the same domain as I and relations $S_1, S_2, \ldots, S_k$, $\vec{x}$ is a tuple of variables ranging over I's domain, and opt *is* max *or* min. *Moreover, formula ϕ has the form $\forall \vec{x} \; \exists \vec{y} \; \psi(\vec{x}, \vec{y}, I, S)$ with ψ quantifier-free.*

Lautemann has observed in [Lau92] that this result can be improved to state that all the values in the range of the objective function, not just the optimal one, can be obtained through logical descriptions. Related to the above example, consider the NP optimization problem MAX SAT. Input structures I are boolean formulae in CNF and the goal is to find the maximum number of clauses that can be simultaneously satisfied by some truth assignment. This problem can be expressed as

$$\max{}_{Max\ Sat}(I) = \max_{C,T}\{|C| \; : \; \forall c \, \exists x [C(c) \rightarrow [(P(x,c) \wedge T(x)) \vee (N(x,c) \wedge \neg T(x))]]\}.$$

We generalize the above result to all NP optimization problems. The price for removing the polynomial bounding restriction is the introduction of weights for all n_1-tuples over the I's domain, where n_1 is the arity of S_1. Note that the domain of I, being finite, can be identified with a set of the form $\{1, 2, \ldots, n\}$.

DEFINITION 2.4.

(1) *Let $k \in \mathbf{N}$. A k-weight assignment is a sequence of recursive functions $\{w_i\}_{i \in \mathbf{N}}$, where each $w_i \; : \; \{1, 2, \ldots, i\}^k \rightarrow \mathbf{R}$. For each k-tuplet $\vec{x}$ and all i and j, if $w_i(\vec{x})$ and $w_j(\vec{x})$ are defined, then $w_i(\vec{x}) = w_j(\vec{x})$.*

(2) *A k-positive weight assignment is a sequence of recursive functions $\{w_i\}_{i \in \mathbf{N}}$, where each $w_i \; : \; \{1, 2, \ldots, i\}^k \rightarrow \mathbf{R}_+$. For each k-tuplet $\vec{x}$ and all i and j, if $w_i(\vec{x})$ and $w_j(\vec{x})$ are defined, then $w_i(\vec{x}) = w_j(\vec{x})$.*

(3) *If S is a relation of arity k over $\{1, 2, \ldots, i\}$ and w is a k-weight assignment, the weight of S is $w(S) = \sum_{\vec{x} \in S} w_i(\vec{x})$.*

THEOREM 2.5. *Let A be a (positive)* NP *optimization problem. There exists a finite type $\mathcal{S} = (n_1, n_2, \ldots, n_k)$, a closed first-order formula ϕ, and an n_1-weight (positive) assignment w such that for each input instance I whose set of feasible solutions is not empty,*

$$\text{opt}_A(I) = \text{opt}_S \; \{w(S_1) \; : \; \phi(\vec{x}, I, S)\}, \tag{2.1}$$

where S is a finite structure of type $\mathcal{S}$ with the same domain as I and relations $S_1, S_2, \ldots, S_k$, $\vec{x}$ is a tuple of variables ranging over I's domain, and opt *is* max *or* min. *Moreover, formula ϕ has the form $\forall \vec{x} \; \exists \vec{y} \; \psi(\vec{x}, \vec{y}, I, S)$ with ψ quantifier-free.*

Proof. Let A be an NP optimization problem. For simplicity we assume that the objective function f_A is integer-valued. Let I be an input structure with domain $\{1, 2, \ldots, n\}$. The structure I is encoded by a string whose length is bounded by

a polynomial in n. Since the objective function f_A is polynomial-time computable, there exists a constant d such that $|\text{opt}_A I| \leq 2^{n^d} - 1$, for all I, where n, as explained, is the size of I. We define inductively the following $(d+1)$-weight assignment w. Initially we order in the lexicographical order the $(d+1)$-tuples over $\{1,2\}$, and we assign to them, in this order, the weights $-2^{2^d-1}, \ldots, -2^0, 2^0, 2^1, \ldots, 2^{2^d-1}, 2^0, \ldots, 2^0$. At the end of stage n, we have assigned the values $-2^{n^d-1}, \ldots, -2^0, 2^0, 2^1, \ldots, 2^{n^d-1}$ to all $(d+1)$-tuples over $\{1, \ldots, n\}$. At stage $n+1$, we order lexicographically all $(d+1)$-tuples over $\{1, \ldots, n+1\}$ that contain $n+1$ and assign to them the values $-2^{(n+1)^d-1}, \ldots, -2^{n^d}, 2^{n^d}, \ldots, 2^{(n+1)^d-1}, 2^0, \ldots, 2^0$. There are $(n+1)^{d+1} - n^{d+1}$ such tuples and $2((n+1)^d - n^d)$ values of the form $\pm 2^k (k \neq 0)$ to assign, and thus, such an assignment is possible. The fact that we need is that for each integer m in the interval $[-(2^{n^d} - 1), 2^{n^d} - 1]$ there exists a set of $(d+1)$-tuples over $\{1, \ldots, n\}$ whose w weights sums to m. Let us suppose that A is a maximization problem (the case of a minimization problem is similar). We consider the following decision problem B.

Instance: A finite input structure $I \in \mathcal{I}_A$ with domain $\{1, 2, \ldots, n\}$, a relation U over $\{1, 2, \ldots, n\}$ of arity $(d+1)$, the $(d+1)$-weight assignment w defined above.
Question: Is there a feasible solution $J \in \mathcal{F}_A(I)$ such that $f_A(I, J) \geq w(U)$?

This is the decision problem associated to the NP optimization problem A and therefore is in NP. Consequently, by Fagin's Theorem 2.2, (I, U) is a YES instance to B if and only if there exists a quantifier-free first-order formula ψ and a finite structure R such that $\forall \vec{x}\ \exists \vec{y}\ \psi(\vec{x}, \vec{y}, I, U, R)$, where $\vec{x}, \vec{y}$, and R satisfy the conditions in Theorem 2.2. Now it is easy to see that if ϕ is the formula $\forall \vec{x}\ \exists \vec{y}\ \psi(\vec{x}, \vec{y}, I, U, R)$, then

$$\text{opt}_A(I) = \max_{U,R}\{w(U)\ :\ \phi(\vec{x}, \vec{y}, I, U, R)\}.$$

Indeed, let $m^* = \text{opt}_A(I)$. By a previous observation, there exists a relation U of arity $(d+1)$ such that $w(U) = m^*$. It follows that (I, U) is a YES instance to the decision problem B, and consequently, $\max_{U,R}\{w(U)\ :\ \phi(\vec{x}, \vec{y}, I, U, R)\} \geq m^*$. Conversely, let U, R be relations such that $\phi(\vec{x}, \vec{y}, I, U, R)$ holds and $w(U)$ is maximum. Then (I, U) is a YES instance to problem B. Therefore there exists a feasible solution $J \in \mathcal{F}_A(I)$ such that $f_A(J) \geq w(U)$ and, consequently, $m^* \geq \max_{U,R}\{w(U)\ :\ \phi(\vec{x}, \vec{y}, I, U, R)\}$. The proof for an arbitrary objective function (i.e., not necessarily integer-valued) is similar but more tedious. The key observation is that in polynomial time f_A can compute only rational values with a number of digits that is polynomial in n, and all these possible values can be covered by weights assigned to tuples of constant arity. The proof for positive NP optimization problems is absolutely similar. □

It should be remarked that, except for the arity, the weight assignment built in the proof of the above theorem is independent of the problem, and thus we have proved a stronger fact. More precisely, there exists a canonical family of weight assignments $\{w_1, \ldots, w_k, \ldots\}$ where, for each k, w_k is a k-assignment, such that for any NP optimization problem the weight assignment w in the expression (2.1) belongs to the family. Thus, the situation is completely similar to the polynomially bounded case from Theorem 2.3 (where the canonical weight assignment assigns value 1 to each tuplet).

We can now classify all NP optimization problems with respect to the quantifier structure of the formulae describing them.

DEFINITION 2.6. *For $n \geq 1$, a $\Sigma_n(\Pi_n)$ formula is a first-order closed formula in prenex normal form that has n alternations of quantifiers starting with a block of*

existential (*universal*) *quantifiers. A* Σ_0 *or* Π_0 *formula is a first-order quantifier-free closed formula.*

DEFINITION 2.7.

(1) *For each* $n \geq 1$, *weight*-MAX FΣ_n *is the set of optimization problems that can be expressed as*

$$\max_A(I) = \begin{cases} \max_S\{w(S_1) \ : \ \phi(\vec{x}, I, S)\}, & \text{if there is } S \text{ such that } \phi(\vec{x}, I, S), \\ default & \text{otherwise,} \end{cases}$$

where S, S_1, w *are as in Theorem* 2.5, *default is a real constant, and* ϕ *is a* Σ_n *formula compatible with* I *and* S.

(2) *For each* $n \geq 1$, *weight(+)*-MAX FΣ_n *is the subclass of weight*-MAX FΣ_n *in which* w *is a positive weight assignment.*

(3) *The classes weight*-MAX FΠ_n, *weight(+)*-MAX FΠ_n, *weight*-MIN FΣ_n, *weight(+)*-MIN FΣ_n, *weight*-MIN FΠ_n, *weight(+)*-MIN FΠ_n *are defined in the similar obvious way.*

It follows from Theorem 2.5 that all NP maximization (minimization) problems (we denote these two classes by weight-MAX NP and weight-MIN NP) are in weight-MAX FΠ_2 (weight-MIN FΠ_2). Also, all positive NP maximization (minimization) problems (denoted weight(+)-MAX NP and weight(+)-MIN NP) are in weight(+)-MAX FΠ_2 (weight(+)-MIN FΠ_2). From an algorithmic point of view, we observe first (by considering all polynomially many possible substitutions for the existentially quantified variables) that all problems in $\mathcal{C}$ FΣ_1, where $\mathcal{C} \in \{$weight(+)-MAX, weight(+)-MIN, weight-MAX, weight-MIN$\}$, can be solved in polynomial time and the problems in $\mathcal{C}$ FΣ_2 can be reduced to problems in $\mathcal{C}$ FΠ_1, with $\mathcal{C}$ as before. From the syntactical characterization point of view, we can prove the following relations.

THEOREM 2.8.

(1) *weight(+)*-MAX FΣ_1 $\subsetneq$ *weight(+)*-MAX FΠ_1 = *weight(+)*-MAX FΣ_2 $\subsetneq$ *weight(+)*-MAX FΠ_2 = *weight(+)*-MAX NP.

(2) *weight(+)*-MIN FΣ_1 *(weight(+)*-MIN FΠ_1*)* $\subsetneq$ *weight(+)*-MIN FΣ_2 $\subsetneq$ *weight(+)*-MIN FΠ_2 = *weight(+)*-MIN NP, *and weight(+)*-MIN FΣ_1 *and weight(+)*-MIN FΠ_1 *are incomparable.*

(3) *weight*-MAX FΣ_1 *(weight*-MAX FΠ_1*)* $\subseteq$ *weight*-MAX FΣ_2 $\subseteq$ *weight*-MAX FΠ_2 = *weight*-MAX NP, *and weight*-MAX FΣ_1 *and weight*-MAX FΠ_1 *are not equal.*

(4) *weight*-MIN FΣ_1 *(weight*-MIN FΠ_1*)* $\subseteq$ *weight*-MIN FΣ_2 $\subseteq$ *weight*-MIN FΠ_2 = *weight*-MIN NP, *and weight*-MIN FΣ_1 *and weight*-MIN FΠ_1 *are not equal.*

Proof. Most of the proof follows closely the arguments in the similar proofs in [KT95]. For convenience, we provide the appropriate pointers.

By Theorem 2.5, $\mathcal{C}$ FΠ_2 = $\mathcal{C}$ NP, where $\mathcal{C} \in \{$weight(+)-MAX, weight(+)-MIN, weight-MAX, weight-MIN$\}$, and thus, just from the syntactical characterization, ($\mathcal{C}$ FΣ_1, $\mathcal{C}$ FΠ_1) $\subseteq$ $\mathcal{C}$ FΣ_2 $\subseteq$ $\mathcal{C}$ FΠ_2.

As in [KT95, Theorem 6.2], MIN VERTEX COVER is in weight(+)-MIN FΠ_1 and in weight-MIN FΠ_1 but not in weight(+)-MAX FΣ_1 and not in weight-MIN FΣ_1.

By the arguments in [PR93], it can be shown that MAX CLIQUE is in weight(+)-MAX FΠ_1 and in weight-MAX FΠ_1 but not in weight(+)-MAX FΣ_1 and not in weight-MAX FΣ_1.

As in [KT95, Remark 2.1 and Theorem 2.1], weight(+)-MAX FΠ_1 = weight(+)-MAX Π_1 and weight(+)-MAX FΣ_i = weight(+)-MAX Σ_i, $i = 1, 2$ (where weight(+)-MAX Σ_i and weight(+)-MAX Π_i are the analogues of MAX Σ_i and MAX Π_i but

with positive weights on tuples). As in [KT94, Theorem 2], weight(+)-MAX Σ_2 = weight(+)-MAX Π_1. Thus, weight(+)-MAX FΣ_2 = weight(+)-MAX Σ_2 = weight(+)-MAX Π_1 = weight(+)-MAX FΠ_1.

As in [KT95, Proposition 2.1], weight(+)-MIN FΠ_1 = weight(+)-MIN Σ_1 and there exists a problem in weight(+)-MIN FΣ_1 which is not in weight(+)-MIN Σ_1 and, therefore, is not in weight(+)-MIN FΠ_1.

We have shown that weight(+)-MIN FΣ_1 and weight(+)-MIN FΠ_1 are incomparable, and thus, both classes are properly included in weight(+)-MIN FΣ_2.

As in [KT95, Theorem 6.2], MIN CYCLE (given a graph G, return the size of the shortest cycle in G) is in weight(+)-MIN FΠ_2 but not in weight(+)-MIN FΣ_2.

The only relation left unproven is weight(+)-MAX F$\Sigma_2 \subsetneqq$ weight(+)-MAX FΠ_2. Consider the following problem $\mathcal{P}$: given a graph $G = (V, E)$, find the length of the longest cycle in G or output 0 if no cycle exists. By Theorem 2.5, $\mathcal{P}$ is in weight(+)-MAX FΠ_2. Since weight(+)-MAX FΣ_2 = weight(+)-MAX FΠ_1, it is enough to show that $\mathcal{P}$ is not in weight(+)-MAX FΠ_1. So, suppose that

$$\max_{\mathcal{P}}(G) = \max_{S_1,\ldots,S_q}\{w(S_1) \ : \ \forall\vec{x}\phi(\vec{x}, G, S_1, \ldots, S_q)\}.$$

Let the arity of S_1 be k and consider the graph G consisting of a cycle $a_1, a_2, \ldots, a_n$, i.e., $(a_i, a_{i+1}) \in E$ and $(a_n, a_1) \in E$, and these are the only tuples in E. Take $n > k$. Let $S^* = (S_1^*, \ldots, S_q^*)$ be a relation that is optimal for G with respect to the above formula. Since $w(S_1^*) = n$, there must be a k-tuple $(a_{i_1}, \ldots, a_{i_k})$ such that $w(a_{i_1}, \ldots, a_{i_k}) > 0$. Let H be the subgraph of G obtained by restricting G to the vertices $a_{i_1}, \ldots, a_{i_k}$. Since Π_1 formulas are closed under taking substructures, $\forall\vec{x}\ \phi(\vec{x}, H, S_{1,H}^*, \ldots, S_{q,H}^*)$ holds true, where $S_H^* = (S_{1,H}^*, \ldots, S_{q,H}^*)$ is the restriction of S^* to $a_{i_1}, \ldots, a_{i_k}$. Therefore, $\max_{\mathcal{P}}(H) \geq w(S_q^*) > 0$. But H is not a cycle, and thus $\max_{\mathcal{P}}(H) = 0$. □

3. Approximation properties. As explained in the Introduction, the logical definability of optimization problems has in many important cases a direct impact on their approximation properties. The syntactical form of formulae describing the constraint conditions of a problem can imply the existence of polynomial-time algorithms achieving a specified ratio of approximation. Or, on the other hand, if a problem is hard under L- or A-reductions (these are some restricted variants of reductions that preserve approximation properties; they were introduced by Papadimitriou and Yannakakis in [PY91] and Panconesi and Ranjan in [PR93]), then the existence of good approximations algorithms is highly improbable. We investigate to what extent the known properties of polynomially bounded optimization problems remain valid when we pass to the positively weighted and arbitrarily weighted corresponding problems.

Definition 3.1.

(1) *Let A be an optimization problem. An approximation algorithm B for A is a function that maps input instances $I \in \mathcal{I}_A$ to feasible solutions in $\mathcal{F}_A(I)$. As a technical convenience, we require that $f_A(I, B(I))$ and* opt $A(I)$ *have strictly positive value for all I. The approximation algorithm B has approximation ratio $r_B \ : \ \mathbf{N} \to [1, +\infty)$ if for all input instances I:*

$$r_B(|I|) \geq \begin{cases} \frac{f_A(I,B(I))}{opt_A I} & \text{if } A \text{ is a minimization problem,} \\ \\ \frac{opt_A I}{f_A(I,B(I))} & \text{if } A \text{ is a maximization problem.} \end{cases}$$

(2) *An optimization problem A is constant* (log)*-approximable if there exists a polynomial-time approximation algorithm B for A such that $r_B(|I|) = O(1)$ ($r_B(|I|) = O(\log(|I|))$) for all input structures I of A.*
(3) *An optimization problem A is superpolylog approximable if there exists a* DTIME$[2^{\log^{O(1)} n}]$ *approximation algorithm B for A and a constant $\mu > 0$ such that $r_B(|I|) = 2^{\log^{\mu} |I|}$ for all input structures I of A.*

For the sake of simplicity, the above definition assumes that the approximation algorithm B outputs the result of the objective function applied to a feasible solution. However, all the following results hold in the more liberal setting in which B is allowed to compute just an approximation (from below or above) of the optimum.

DEFINITION 3.2. *The optimization problem Π L-reduces to the optimization problem Π' if there are two polynomial-time algorithms f and g and constants $\alpha, \beta > 0$ such that*

(1) *given any instance I of Π, f produces an instance I' of Π' such that* $\mathrm{opt}_{\Pi'} I' \leq \alpha\ \mathrm{opt}_{\Pi} I$,
(2) *given a feasible solution of I' of cost c', g produces a feasible solution of I of cost c such that $|c - \mathrm{opt}_{\Pi} I| \leq \beta |c' - \mathrm{opt}_{\Pi'} I'|$.*

We next take into review all syntactically defined classes of NP optimization problems that have been identified in earlier works to have good approximation properties.

MAX SNP and **MAX NP**. A polynomially bounded maximization problem A is in MAX NP if $\max_A(I) = \max_S\ |\{\vec{x}\ :\ \exists \vec{y}\ \phi(\vec{x}, \vec{y}, I, S)\}|$, where ϕ is a first-order quantifier-free formula. If $\max_A(I) = \max_S\ |\{\vec{x}\ :\ \phi(\vec{x}, I, S)\}|$, then A is in MAX SNP. It is known from [PY91] that if A is in MAX NP or MAX SNP, then A is constant approximable. The weighted variants of these classes are defined by considering m-weight assignments for the $\vec{x}$-tuples in the formulae above (m is the arity of $\vec{x}$) and proceeding as in the definition of the classes MAX FΣ_i or MAX FΠ_i. Papadimitriou and Yannakakis have shown in [PY91] that all problems in weight(+)-MAX NP and weight(+)-MAX SNP are constant approximable. This property does not extend to weight-MAX SNP (and to weight-MAX NP), unless NP $\subseteq$ P. In order to prove this we consider the following well-known problem.

weight-MAX 2SAT: An instance of this problem is a CNF formula with two literals per clause and with real-valued weights assigned to each clause. The goal is to maximize the total weight of the clauses that can be simultaneously satisfied by a truth assignment.

The problem is in weight-MAX SNP by the following logical description. An input structure consists of a pair $(Var, C_0, C_1, C_2, C_3)$, where the domain Var is the set of variables and C_i are relations of arity 4 such that $x_1, x_2, x_1, x_2 \in C_0$ means that there exists a clause $c = \overline{x}_1 \vee \overline{x}_2$, $x_1, x_2, x_2, x_1 \in C_1$ means that there exists a clause $c = x_1 \vee \overline{x}_2$, $x_1, x_2, x_1, x_1 \in C_2$ means that there exists a clause $c = \overline{x}_1 \vee x_2$, and $x_1, x_2, x_2, x_2 \in C_3$ means that there exists a clause $c = x_1 \vee x_2$ (we assume that all clauses have distinct variables). Assigning weights to any 4-tuple of variables in the obvious way, we can write $\max_{MAX\ 2SAT}(C) = \max_T\ w(\{(t,u,v,w)\ :\ (C_0(t,u,v,w) \rightarrow (\neg T(t) \vee \neg T(u))) \wedge (C_1(t,u,v,w)\ \rightarrow\ (T(t) \vee \neg T(u))) \wedge (C_2(t,u,v,w)\ \rightarrow\ (\neg T(t) \vee T(u))) \wedge (C_3(t,u,v,w) \rightarrow (T(t) \vee T(u)))\})$. The following lemma, whose proof is deferred until the end of this section, holds.

LEMMA 3.3. *If* NP $\neq$ P*, then weight-MAX 2SAT is not approximable within ratio n^c for any $c < 1/4$, and, if* NP $\neq$ coRP*, then weight-MAX 2SAT is not approximable within ratio n^c for any $c < 1/3$.*

Consequently we obtain the following theorem.

THEOREM 3.4. *If* NP $\neq$ P*, there are problems in weight-*MAX SNP *that are not approximable with approximation ratio* n^c *for any* $c < 1/4$*. If* NP $\neq$ coRP*, there are problems in weight-*MAX SNP *that are not approximable with approximation ratio* n^c *for any* $c < 1/3$.

We do not know if weight-MAX 2SAT is complete for weight-MAX SNP under L- (or A-) reductions. Nevertheless, this problem could be a good starting point for a chain of L-reductions showing that other natural problems do not possess the superpolylog approximation property under the same hypothesis. For example, the reduction from [PY91] shows that weight-MAX NOT-ALL-EQUAL-2 SAT falls into this category.

Papadimitriou and Yannakakis have considered in [PY91] a variant of MAX SNP, called MAX SNP(π), in which the structure over which we maximize is required to be a total linear ordering (i.e., a permutation) of the domain of the input structure. This class contains natural problems like MAX SUBDAG: given a directed graph $G = (V, E)$, find an acyclic subgraph $G' = (V, E')$ with E' as large as possible. The weighted version of this problem is obtained, of course, by assigning weights to edges. It was shown in [PY91] that MAX SUBDAG is L-complete for MAX SNP(π) and it is straightforward to extend this result to weight(+)-MAX SUBDAG. MAX SUBDAG and weight(+)-MAX SUBDAG can be approximated in polynomial time with ratio 2: take any permutation of the vertices and its reverse and choose the one that yields an orientation whose weight is larger. It follows that all problems in MAX SNP(π) and weight(+)-MAX SNP(π) are constant approximable. Clearly, weight-MAX SUBDAG continues to be constant approximable, because we can just disregard the edges with negative weights. We consider another important natural problem (see [GM84, p. 465 ff.]) in weight-MAX SNP(π), which is closely related to MAX SUBDAG.

weight-PRIORITY ORDERING: Given a set X and real-valued weights w to all pairs of distinct elements in X, find the maximum over all permutations π of $\sum_{\{(x,y):\pi(x)<\pi(y)\}} w(x, y)$.

In the hypothesis NP $\not\subseteq \tilde{\text{P}}$, we show in the next section that this problem is not superpolylog approximable. Thus we have established the following theorem.

THEOREM 3.5. *If* NP $\not\subseteq \tilde{\text{P}}$*, there are problems in weight-*MAX SNP(π) *that are not superpolylog approximable.*

MIN $\text{F}^+\mathbf{\Pi_1}$ and **MIN**$^+$ $\text{F}\mathbf{\Pi_2}(\mathbf{1})$. These are subclasses of MIN $\text{F}\Pi_1$ and, respectively, MIN $\text{F}\Pi_2$ and have been identified in [KT95] as having good approximation properties. A minimization problem A is in MIN $\text{F}^+\Pi_1$ if $\min_A(I) = \min_S\{|S| : \forall \vec{x}\ \phi(\vec{x}, I, S)\}$, where the structure S consists of a single relation and ϕ is a quantifier-free formula in CNF with variables $\vec{x}$ in which all occurrences of S are positive. The minimization problem A is in MIN $\text{F}^+\Pi_2(1)$ if $\min_A(I) = \min_S\{|S| : \forall \vec{x}\ \exists \vec{y}\ \phi(\vec{x}, \vec{y}, I, S)\}$, where the structure S consists of a single relation and ϕ is a quantifier-free formula in DNF with variables $\vec{x}, \vec{y}$ in which all occurrences of S are positive and S occurs at most once in each disjunct. The weighted versions of these classes are defined in the obvious way. These apparently obscure classes have as complete sets (under L-reductions) some very important optimization problems. Thus, it is shown in [KT94] that k-HYPERVERTEX COVER is complete for MIN $\text{F}^+\Pi_1(k)$ (a subclass of MIN $\text{F}^+\Pi_1$, in which S is restricted to appear at most k times in each clause of ϕ) and SET COVER, DOMINATING SET, and HITTING SET are complete for MIN $\text{F}^+\Pi_2(1)$ (see [KT95]). It is straightforward to extend these results for the weight and weight(+) versions of the problems and classes (e.g., weight

VERTEX COVER is L-complete for weight-MIN $\mathrm{F}^+\Pi_1(2)$, and so on). Since k-HYPERVERTEX COVER is constant approximable and SET COVER is log approximable (see [KT95]), it follows that all the problems in MIN $\mathrm{F}^+\Pi_1$ are constant approximable and all the problems in MIN $\mathrm{F}^+\Pi_2(1)$ are log approximable. It can be shown that weight(+)-k-HYPERVERTEX COVER continues to be constant approximable (the results in [BYE81] and [Hoc82] concerning weight(+) VERTEX COVER can be extended to weight(+)-k-HYPERVERTEX COVER). Chvàtal [Chv79] has proved that weight(+)-SET COVER also continues to be log approximable. From the L-completeness of these problems, we obtain the following theorem.

THEOREM 3.6.

(1) *All problems in weight(+)-*MIN $\mathrm{F}^+\Pi_1$ *are constant approximable.*

(2) *All problems in weight(+)-*MIN $\mathrm{F}^+\Pi_2(1)$ *are* log *approximable.*

We next pass to the general weighted version of these classes. Observe that weight-VERTEX COVER is in both weight-MIN $\mathrm{F}^+\Pi_1$ and weight-MIN $\mathrm{F}^+\Pi_2(1)$, since $\min_{VC}(G) = \min_S\{w(S) \ : \ \forall x\ \forall y\ (\neg E(x,y) \vee S(x) \vee S(y))\}$. The following lemma, whose proof is deferred until the end of this section, holds.

LEMMA 3.7. *If* $\mathrm{P} \neq \mathrm{NP}$*, then for every constant* q*, weight-VERTEX COVER is not approximable in polynomial time with ratio* 2^{n^q}.

Consequently, we obtain the following theorem.

THEOREM 3.8. *If* $\mathrm{NP} \not\subseteq \tilde{\mathrm{P}}$*, then, for every* q*, there are problems in weight-*MIN $\mathrm{F}^+\Pi_1$ *and weight-*MIN $\mathrm{F}\Pi_2(1)$ *that are not approximable in polynomial time with approximation ratio* 2^{n^q}.

It follows that the problems weight-SET COVER, weight-DOMINATING SET, and weight-HITTING SET, being L-complete for weight-MIN $\mathrm{F}^+\Pi_2(1)$, are not approximable in polynomial time within a 2^{n^q} ratio, unless NP = P.

In the rest of this section, we prove the hard-to-approximate properties of weight-MAX 2SAT and weight-VERTEX COVER.

Proof of Lemma 3.3. The proof consists of an L-reduction of INDEPENDENT SET to weight-MAX 2SAT which is obtained by slightly modifying the reduction of INDEPENDENT SET for graphs with bounded degree to MAX 2SAT from [PY91]. It can be easily shown that if δ is a lower bound on the approximation ratio of Π and π L-reduces to Π' with the constants α and β as in Definition 3.2, then $\delta/(\alpha\beta)$ is a lower bound on the approximation ratio of Π'. We will achieve $\alpha = \beta = 1$, and then we use the results of Bellare, Goldreich, and Sudan [BGS95] stating that if $\mathrm{P} \neq \mathrm{NP}$ ($\mathrm{NP} \neq \mathrm{coRP}$) then INDEPENDENT SET is not approximable in polynomial time with ratio n^c for any $c < 1/4$ (respectively, $c < 1/3$).

We only have to exhibit the L-reduction. Given $G = (V, E)$, an instance of INDEPENDENT SET, we build a formula ϕ in 2-CNF as follows: for each node i, we consider a clause x_i, and for each edge (i, j) we consider a clause $(\overline{x}_i \vee \overline{x}_j)$. All these clauses have weight 1. We consider a new variable y and two more clauses, y and $\overline{y}$, each with weight $-|E|$. It is easy to observe the following fact.

Fact. Given an assignment for ϕ with cost c, we can find in polynomial time an assignment with cost $c' \geq c$ such that all the clauses corresponding to edges are satisfied in the new assignment. Such an assignment is said to be in *normal form.*

Now, it is easy to see that the nodes corresponding to the clauses that are made true by a normal form assignment form an independent set. Taking into account that one of the clauses y, $\neg y$ is satisfied by any assignment, it follows that

$$\mathrm{opt}_{MAX2SAT}(\phi) = \mathrm{opt}_{IND.SET}(G) + |E| - |E| = \mathrm{opt}_{IND.SET}(G).$$

Also, given any assignment with cost c', we can build in polynomial time a normal form assignment with cost $c \geq c'$ which corresponds to an independent set of size c and, thus, $\text{opt}(G) - c \leq \text{opt}(\phi) - c'$. □

Proof of Lemma 3.7. The proof consists of a slight modification of the "classical" reduction of 3-SAT to VERTEX COVER in [GJ79]. Let ϕ be a formula in 3-CNF having variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$. We build the following undirected graph $G = (V, E)$. The nodes in V are $x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n$, and for each clause $C_i = (\alpha \vee \beta \vee \gamma)$, there are the nodes $(\alpha, i), (\beta, i), (\gamma, i)$. The nodes described so far each have weight $W = 2^{n^q}$. There are two more special nodes, y and z, each of them having weight $1 - nW - 2mW$. For each variable x_i, there is the edge $(x_i, \overline{x}_i)$, and for each clause $C_i = (\alpha \vee \beta \vee \gamma)$, we introduce in E the edges $((\alpha, i), (\beta, i)), ((\beta, i), (\gamma, i))$, and $((\gamma, i), (\alpha, i))$ (forming a triangle). For all $(\alpha, i) \in V$, we introduce in E the edge $(\alpha, (\alpha, i))$ (the nodes labeled with over-lined variables correspond to negated variables). G also contains the edge (y, z).

Observe that in order to cover an edge of the form $(x_1, \overline{x}_i)$ corresponding to the variable x_i, we need to select at least one of x_i or $\overline{x}_i$ in the vertex cover, and in order to cover the three edges corresponding to a clause $C_i = (\alpha \vee \beta \vee \gamma)$, at least two of the nodes $(\alpha, i), (\beta, i), (\gamma, i)$ must be taken in the vertex cover. Also, one of y or z must also be chosen in the vertex cover. As in the classical reduction, it can be seen that if ϕ is satisfiable, there is a vertex cover of G having only the minimum number of nodes specified above and having weight $nW + 2mW + 1 - nW - 2mW = 1$. On the other hand, if ϕ is not satisfiable, at least one more node, other than y or z, must be taken in any vertex cover and, thus, in this case, $\min_{VC}(G) \geq 1 + W$. Consequently, if weight-VERTEX COVER would be approximable in polynomial time with ratio less than W, then we could solve 3-SAT in polynomial time. □

4. Reductions from MIP systems. The hard-to-approximate property of the problem PRIORITY ORDERING, which was introduced in the previous section, is established by a reduction from MIP systems running in one round. Such reductions have been extensively used in recent years to investigate and solve long-standing open problems regarding the approximation properties of many natural combinatorial optimization problems [ALM+92, Bel92, BS94, Con91, FGL+91, FL92, LY93a, LY93b, Zuc93, Zim93]. We now introduce the necessary terminology. Following [BGLR93], we denote by $\text{MIP}_1(r, p, a, q, \epsilon)$ a one-round MIP system in which the number of random bits is $r(n)$, the number of provers is $p(n)$, the size of each verifier's query is $q(n)$, the size of each prover's answer is $a(n)$, and the error probability is $\epsilon(n)$, where n is the size of the input (which, for conciseness, will be omitted). We describe the way such a system works. A $\text{MIP}_1(r, p, a, q, \epsilon)$ system involves $p + 1$ parties: one verifier V and p provers $P_1, P_2, \ldots, P_p$. All these parties share a common input x, and it is the joint goal of the provers to convince V to accept x. The interaction between $V, P_1, \ldots, P_p$ runs as follows. The verifier randomly selects a string R of length r and computes $\pi(x, R) = (q_1, q_2, \ldots, q_p)$, a p-tuplet of queries, all of length q, and sends, for all i, the query q_i to the prover P_i. The provers compute their answers $a_i = P_i(x, q_i)$, where P_i is the function defining the strategy of the prover P_i (note the overload in the notation P_i) and the length of all a_i's is a. After the verifier receives the answers $a_1, a_2, \ldots, a_p$, she computes $\rho(x, q_1, q_2, \ldots, q_p, a_1, a_2, \ldots, a_p)$, which tells her whether to accept x or not. We assume that π and ρ, which define the verifier strategy, are polynomial-time computable functions, but there is no restriction on the functions P_i (i.e., the provers are arbitrary powerful). Let $ACC_{V,(P_1,P_2,\ldots,P_p)}(x)$ denote the probability that $\rho(x, q_1, q_2, \ldots, q_p, a_1, a_2, \ldots, a_p) =$ "accept," when R is

chosen randomly of length r and the q_i's and a_i's are as above. The value of the verifier strategy V at x is the maximum of $ACC_{V,(P_1,P_2,\ldots,P_p)}(x)$ over all p-tuples $(P_1, P_2, \ldots, P_p)$ of prover strategies. We denote this value by $ACC_V(x)$. We say that V accepts a language L with error probability ϵ (where $\epsilon : \mathbf{N} \to \mathbf{R}$ and $L \subset \Sigma^*$) if:

(1) $x \in L$ implies $ACC_V(x) = 1$, and

(2) $x \notin L$ implies $ACC_V(x) < \epsilon(|x|)$.

We say that L is accepted by a $\mathrm{MIP}_1(r, p, a, q, \epsilon)$ if there is a verifier V running the above protocol that accepts L.

There has been a stream of important works, [BFL91], [ALM+92], [FGL+91], [FRS88], [FL92], [BGLR93], [FK94], [PS94], leading to multiprover interactive systems accepting the NP complete set SAT with increasingly better parameters r, p, a, q, ϵ. We use the following variant (see [BGLR93]).

THEOREM 4.1. *SAT is accepted by a* $\mathrm{MIP}_1(O(\log^3 n), 2, O(\log^3 n), O(\log^3 n), 1/n)$.

The general technique we use to prove that a maximization problem A is hard to approximate consists in reducing via a function Φ a $\mathrm{MIP}_1(r, p, a, q, \epsilon)$ verifier strategy V to A in such a way that valid provers' strategies $(P_1, P_2, \ldots, P_p)$ correspond to feasible solutions, $ACC_{V,(P_1,P_2,\ldots,P_p)}(x) = f_A(J)$, where $J \in \mathcal{F}_A(\Phi(x))$ and $\max_A \Phi(x) = 2^{r(|x|)} \cdot ACC_V(x)$.

DEFINITION 4.2. *We say that* $\mathrm{MIP}_1(r, p, a, q, \epsilon)$ *reduces to the maximization problem* A *if for any verifier strategy* V *running the* $\mathrm{MIP}_1(r, p, a, q, \epsilon)$ *protocol, there is a function* Φ_V *which maps a common input* x *for the verifier strategy into an instance* $\Phi_V(x) \in \mathcal{I}_A$ *of the optimization problem* A *and has the following properties:*

(1) *for all* $x \in \Sigma^*$, $\max_A \Phi_V(x) = 2^{r(|x|)} \cdot ACC_V(x)$, *and*

(2) $\Phi_V(x)$ *is computable in time* $2^{\log^c |x|}$ *for some constant* c.

The following lemma is an easy consequence of Theorem 4.1.

LEMMA 4.3. *Suppose that a* $\mathrm{MIP}_1(r, p, a, q, \epsilon)$ *of the type in Theorem* 4.1 *reduces to the maximization problem* A. *Then there is a constant* $\mu > 0$ *such that the existence of a* $\mathrm{DTIME}[2^{\log^\mu n}]$ *approximation algorithm* B *for* A *with* $r_B(|I|) < 2^{\log^\mu |I|}$ *implies* $\mathrm{NP} \subseteq \mathrm{DTIME}[2^{O(\log^{1/\mu} n)}]$.

Proof. Let ϕ be a boolean formula. There exists a verifier strategy V running a $\mathrm{MIP}_1(r, p, a, q, \epsilon)$ protocol of the type in Theorem 4.1 such that $\phi \in \mathrm{SAT} \to ACC_V(\phi) = 1$ and $\phi \notin \mathrm{SAT} \to ACC_V(\phi) \leq \epsilon(|\phi|) = 1/|\phi|$. There exists a constant c such that $|\Phi_V(\phi)| \leq 2^{\log^c |\phi|}$, where Φ_V is the reduction function. Note that

$$\phi \in \mathrm{SAT} \to ACC_V(\phi) = 1 \to \max A(\Phi_V(\phi)) = 2^{r(|\phi|)}$$

and

$$\phi \notin \mathrm{SAT} \to ACC_V(\phi) \leq \frac{1}{|\phi|} \to \max A(\Phi_V(\phi)) \leq \frac{1}{|\phi|} \cdot 2^{r(|\phi|)}.$$

Take $\mu = 1/c$ and suppose there is a $\mathrm{DTIME}[2^{\log^\mu n}]$ algorithm B for A with $r_B(|I|) < 2^{\log^\mu |I|}$, which means $r_B(|\Phi_V(\phi)|) < |\phi|$. In consequence:

$$\phi \in \mathrm{SAT} \to B(\Phi_V(\phi)) \geq \frac{1}{r_B(|\Phi_V(\phi)|)} \cdot \max A(\Phi_V(\phi)) > \frac{1}{|\phi|} \cdot 2^{r(|\phi|)}$$

and

$$\phi \notin \mathrm{SAT} \to B(\Phi_V(\phi)) \leq \max A(\Phi_V(\phi)) \leq \frac{1}{|\phi|} \cdot 2^{r(|\phi|)}.$$

Since $B(\Phi_V(\phi))$ can be computed in time $2^{O(\log^c |\phi|)}$, it follows that

$$\text{NP} \subseteq \text{DTIME}[2^{O(\log^{1/\mu} n)}]. \qquad \square$$

Taking into account Lemma 4.3, we need only to design the appropriate reduction from MIP systems with parameters as in Theorem 4.1 to PRIORITY ORDERING. This is the content of the next lemma.

LEMMA 4.4. $\text{MIP}_1(O(\log^3 n), 2, O(\log^3 n), O(\log^3 n), 1/n)$ *reduces to weight-PRIORITY ORDERING.*

Proof. Let V be a verifier executing a $\text{MIP}_1(d\log^3 n, 2, d\log^3 n, d\log^3 n, 1/n)$ protocol and let $n = |x|$, where x is the common input of the protocol. We build an instance for weight-PRIORITY ORDERING. Let $N = 2^{d\log^3 n}$. The set of elements to be ordered is

$$X = \{s_1, s_2, s_3, s_4\} \cup X_1 \cup X_2 \cup X_3,$$

where $X_1 = \{(u, v, a, 1) \ : \ u, v, a \in \Sigma^{=d\cdot\log^3 n}\}$, $X_2 = \{(u, v, b, 2) \ : \ u, v, b \in \Sigma^{=d\cdot\log^3 n}\}$, and $X_3 = \{(u, v, b, 3) \ : \ u, v, b \in \Sigma^{=d\cdot\log^3 n}\}$. We first overview the construction. Intuitively, u and v should be interpreted as queries to the first and, respectively, the second prover, and a and b, with or without subscripts, should be interpreted as answers provided by the first and, respectively, the second prover. By using large weights, we force any arrangement π that is a candidate for being the optimal one to achieve (i) $s_1 <_\pi s_2 <_\pi s_3 <_\pi s_4$, (ii) $s_2 <_\pi x <_\pi s_3$ for all $x \in X_2$, (iii) $x <_\pi s_4$ for all $x \in X_1$, and (iv) $s_1 <_\pi x$ for all $x \in X_3$. For all pairs $(u, v) \in (\Sigma^{=d\cdot\log^3 n})^2$ we order the set $X_{u,v,1} = \{(u, v, a, 1) \ : \ a \in \Sigma^{=d\cdot\log^3 n}\}$ in the lexicographical order $<_l$, obtaining $X_{u,v,1} = \{t_1 <_l t_2 <_l \cdots <_l t_N\}$. We assign weights such that ideally $s_3 <_\pi t_1 <_\pi t_2 <_\pi \cdots <_\pi t_N <_\pi s_1$. Since $s_1 <_\pi s_3$, this is not possible and therefore there exists j such that $t_{j+1} <_\pi t_{j+2} <_\pi \cdots <_\pi t_N <_\pi s_1 <_\pi s_3 <_\pi t_1 <_\pi \cdots <_\pi t_j$. Similarly, for all pairs $(u, v) \in (\Sigma^{=d\cdot\log^3 n})^2$ we order the set $X_{u,v,3} = \{(u, v, b, 3) \ : \ a \in \Sigma^{=d\cdot\log^3 n}\}$ in the lexicographical order $<_l$, obtaining $X_{u,v,3} = \{t_1 <_l t_2 <_l \cdots <_l t_N\}$, and we force the existence of $k \in \{1, \ldots, N\}$ with $t_{k+1} <_\pi t_{k+2} <_\pi \cdots <_\pi t_N <_\pi s_2 <_\pi s_4 <_\pi t_1 <_\pi \cdots <_\pi t_k$. Using some other combinations of weights involving the "stakes" s_1, s_2, s_3, s_4, we guarantee that for all $u \in \Sigma^{=d\cdot\log^3 n}$ there exists $j \in \{1, \ldots, N\}$ such that for all $v \in \Sigma^{=d\cdot\log^3 n}$, $(u, v, a_{j+1}, 1) <_\pi \cdots <_\pi (u, v, a_N, 1) <_\pi s_1 <_\pi s_3 <_\pi (u, v, a_1, 1) <_\pi \cdots <_\pi (u, v, a_j, 1)$ and for all $v \in \Sigma^{=d\cdot\log^3 n}$, there exists $k \in \{1, \ldots, N\}$ such that for all $u \in \Sigma^{=d\cdot\log^3 n}$, $(u, v, b_{k+1}, 3) <_\pi \cdots <_\pi (u, v, b_N, 3) <_\pi s_2 <_\pi s_4 <_\pi (u, v, b_1, 3) <_\pi \cdots <_\pi (u, v, b_k, 3)$. These unique values of j and k for each u and v define a pair of provers' strategies (P_1, P_2). Moreover, the weights are carefully defined in order to insure that $\max_\pi \sum_{\{(t,z) \ : \ \pi(t)<\pi(z)\}} w(t, z) = ACC_V(x)$.

Now we proceed with the complete and formal description of the reduction. We lexicographically order the sets of possible answers (i.e., the elements in $\Sigma^{=d\cdot\log^3 n}$) obtaining: $a_1 < a_2 \cdots < a_N$ and $b_1 < b_2 \cdots < b_N$ (of course, $a_i = b_i$, but keep in mind the above intuitive interpretation). Let $S = 3N^3 + 4$ denote the number of elements in X. We next define the weights, $w(\cdot, \cdot)$, for some pairs in $X \times X$. Given an ordering π of X and a subset $Y \subseteq X$, the contribution of Y is defined to be $C_\pi(Y) = \sum_{x,y\in Y,\pi(x)<\pi(y)} w(x, y)$. We call a permutation π *reasonable* if $C_\pi(X) \geq 0$. Since there exists an ordering σ (to be described below; it corresponds to a tuple of provers' strategies) such that $C_\sigma(X) \geq 0$, it follows that the optimal ordering π must be looked for among the reasonable ones.

For each 4-tuplet $(u, v, a, b) \in (\Sigma^{=d\cdot\log^3 n})^4$, we denote

$$val_{u,v,a,b} = \begin{cases} card\{R \in \Sigma^{=d\cdot\log^3 n} \ : \ \pi(x, R) = (u, v)\} & \text{if } \rho(x, u, v, a, b) = \text{ "accept,"} \\ 0 & \text{otherwise,} \end{cases}$$

and by solving a linear system, we fix for each u, v, a_i, b_h, a real value d_{u,v,a_i,b_h} such that for each $u, v \in \Sigma^{=d\cdot\log^3 n}$, and for each $j, k \in \{1, \ldots, N\}$, $\sum_{1\le i\le j} \sum_{1\le h\le k} d_{u,v,a_i,b_h} = val_{u,v,a_j,b_k}$. We now state the weights of pairs in X^2 in six stages (1)–(6) (all the pairs that are not mentioned below have weight 0).

(1) $w(s_2, s_1) = -M_1$, $w(s_3, s_2) = -M_1$, $w(s_4, s_3) = -M_1$,
$w(x, s_2) = -M_1$ and $w(s_3, x) = -M_1$ for all $x \in X_2$,
$w(s_4, x) = -M_1$ for all $x \in X_1$,
$w(x, s_1) = -M_1$ for all $x \in X_3$.
M_1 is defined as follows. Let k be the maximum weight for any pair defined in stages (2)–(6). Then $M_1 = \frac{S(S-1)}{2} \cdot k + 1$. The effect of these definitions is that if π is a reasonable ordering and $<_\pi$ denotes the order induced by π (i.e., $x <_\pi y$ if and only if $\pi(x) < \pi(y)$), then (i) $s_1 <_\pi s_2 <_\pi s_3 <_\pi s_4$, (ii) $s_2 <_\pi x <_\pi s_3$ for all $x \in X_2$, (iii) $x <_\pi s_4$ for all $x \in X_1$, and (iv) $s_1 <_\pi x$ for all $x \in X_3$. (See Figure 2.)

(2) For each pair $(u, a) \in (\Sigma^{=d\cdot\log^3 n})^2$, we fix a linear order $<_o$ on the set $Y_{u,a,1} = \{(u, v, a, 1) \ : \ v \in \Sigma^{=d\cdot\log^3 n}\}$, obtaining $Y_{u,a,1} = \{z_1 <_o z_2 <_o \cdots <_o z_N\}$. The order $<_o$ must have the following property: $\forall w \ \forall z$ [(w is the successor relative to $<_o$ of z) $\rightarrow$ (w is not the lexicographical successor of z) and (z is not the lexicographical successor of w)] and also z_1 and z_N are not the first and, respectively, last elements from $Y_{u,a,1}$ with respect to the lexicographical ordering. We define the weights:

$$\begin{array}{ll} w(s_3, z_1) = M_2, & w(z_1, s_3) = -M_2, \\ w(z_{i+1}, z_i) = -3M_2, & i = 1, \ldots, N-1, \\ w(z_N, s_1) = M_2, & w(s_1, z_N) = -M_2. \end{array}$$

M_2 is defined by $M_2 = \frac{S(S-1)}{2} \cdot k + 1$, where k is now the maximum weight defined in the future stages (3)–(6). The effect of these definitions is that for all $u, a \in \Sigma^{=d\cdot\log^3 n}$, if π is a reasonable ordering, then either all elements from $Y_{u,a,1}$ are smaller with respect to π than s_1 or all elements from $Y_{u,a,1}$ are larger with respect to π than s_3. (See Figure 2.)

(3) For each pair $(v, b) \in (\Sigma^{=d\cdot\log^3 n})^2$, we fix a linear order $<_o$ on the set $Y_{v,b,3} = \{(u, v, b, 3) \ : \ u \in \Sigma^{=d\cdot\log^3 n}\}$, obtaining $Y_{v,b,3} = \{z_1 <_o z_2 <_o \cdots <_o z_N\}$. The order $<_o$ is as in (2). We define the weights:

$$\begin{array}{ll} w(s_4, z_1) = M_3, & w(z_1, s_4) = -M_3, \\ w(z_{i+1}, z_i) = -3M_3, & i = 1, \ldots, N-1, \\ w(z_N, s_2) = M_3, & w(s_2, z_N) = -M_3. \end{array}$$

M_3 is defined by $M_3 = \frac{S(S-1)}{2} \cdot k + 1$, where k is the maximum weight defined in the future stages (4)–(6). The effect of these definitions is that for all $v, b \in \Sigma^{=d\cdot\log^3 n}$, if π is a reasonable ordering, then either all elements from $Y_{v,b,3}$ are smaller with respect to π than s_2 or all elements from $Y_{v,b,3}$ are larger with respect to π than s_4. (See Figure 2.)

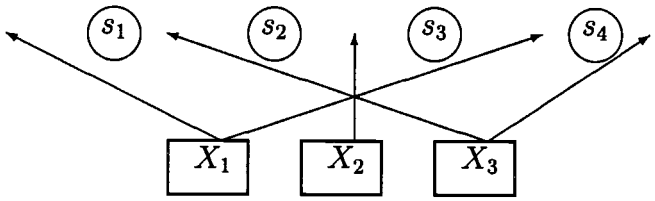


FIG. 2. *The effect of stages* (1), (2), *and* (3): *the elements of* X_1, X_2, *and* X_3 *must be positioned between the stakes* s_1, s_2, s_3, *and* s_4 *as in the figure.*

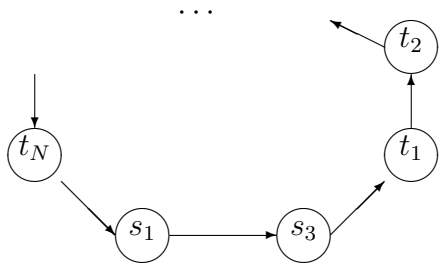


FIG. 3. *The effect of stage* (4): *only one of the edges* $(t_1, t_2), \ldots, (t_N, s_1)$ *should be "broken" by a reasonable* π.

(4) For all pairs $(u, v) \in (\Sigma^{=d\cdot\log^3 n})^2$ we order the set $X_{u,v,1} = \{(u, v, a, 1) : a \in \Sigma^{=d\cdot\log^3 n}\}$ in the lexicographical order $<_l$, obtaining $X_{u,v,1} = \{t_1 <_l t_2 <_l \cdots <_l t_N\}$. We define the weights:

$$\begin{array}{ll} w(s_3, t_1) = M_4, & w(t_1, s_1) = -M_4, \\ w(t_{i+1}, t_i) = -M_4, & i = 1, \ldots, N-1, \\ w(s_1, t_N) = -M_4, & \end{array}$$

with $M_4 = \frac{S(S-1)}{2} \cdot k + 1$, where k is now the maximum weight defined in the future stages (5)–(6). It can be seen that, by the precautions we took when selecting the order $<_o$, there is no conflict between the definitions in stages (2) and (4). It is important to note that, relative to an ordering π, $C_\pi(X_{u,v,1} \cup \{s_1, s_3\}) = 0$ if there exists $j \in \{1, \ldots, N\}$ such that

$$t_{j+1} <_\pi t_{j+2} <_\pi \cdots <_\pi t_N <_\pi s_1 <_\pi s_3 <_\pi t_1 <_\pi \cdots <_\pi t_j$$

or $C_\pi(X_{u,v,1} \cup \{s_1, s_3\})$ is less than $-M_4$ in all other situations. Hence, a reasonable ordering π must satisfy the above inequalities. (See Figure 3.)

(5) For all pairs $(u, v) \in (\Sigma^{=d\cdot\log^3 n})^2$ we order the set $X_{u,v,3} = \{(u, v, b, 3) : a \in \Sigma^{=d\cdot\log^3 n}\}$ in the lexicographical order $<_l$, obtaining $X_{u,v,3} = \{t_1 <_l t_2 <_l \cdots <_l t_N\}$. We define the weights:

$$\begin{array}{ll} w(s_4, t_1) = M_5, & w(t_1, s_4) = -M_5, \\ w(t_{i+1}, t_i) = -M_5, & i = 1, \ldots, N-1, \\ w(s_2, t_N) = -M_5, & \end{array}$$

with $M_5 = \frac{S(S-1)}{2} \cdot k + 1$, where k is now the maximum weight defined in stage (6). By the same arguments as in (4), a reasonable ordering π must

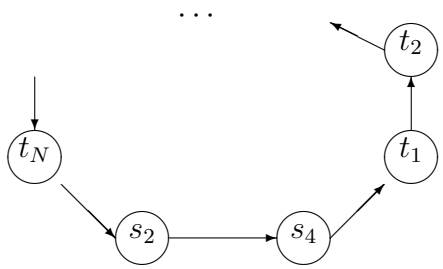


FIG. 4. *The effect of stage* (5): *only one of the edges* $(t_1, t_2), \ldots, (t_N, s_2)$ *should be "broken" by a reasonable* π.

satisfy

$$t_{k+1} <_\pi t_{k+2} <_\pi \cdots <_\pi t_N <_\pi s_2 <_\pi s_4 <_\pi t_1 <_\pi \cdots <_\pi t_k,$$

for some $k \in \{1, \ldots, N\}$. (See Figure 4.)

(6) For all 4-tuples $(u, v, a, b) \in (\Sigma^{=d\cdot\log^3 n})^4$,

$$w((u, v, b, 2), (u, v, a, 1)) = d_{u,v,a,b}, \quad w((u, v, b, 3), (u, v, a, 1)) = -d_{u,v,a,b}.$$

CLAIM 4.5. (i) *Let* $u \in \Sigma^{=d\cdot\log^3 n}$ *and let* π *be a reasonable ordering. There exists* $j \in \{1, \ldots, N\}$ *such that for all* $v \in \Sigma^{=d\cdot\log^3 n}$

$$(u, v, a_{j+1}, 1) <_\pi \cdots <_\pi (u, v, a_N, 1) <_\pi s_1 <_\pi s_3 <_\pi (u, v, a_1, 1) <_\pi \cdots <_\pi (u, v, a_j, 1).$$

(ii) *Let* $v \in \Sigma^{=d\cdot\log^3 n}$ *and let* π *be a reasonable ordering. There exists* $k \in \{1, \ldots, N\}$ *such that for all* $u \in \Sigma^{=d\cdot\log^3 n}$

$$(u, v, b_{k+1}, 3) <_\pi \cdots <_\pi (u, v, b_N, 3) <_\pi s_2 <_\pi s_4 <_\pi (u, v, b_1, 3) <_\pi \cdots <_\pi (u, v, b_k, 3).$$

Proof. We prove (i). From the definitions in stage (4), for each $v \in \Sigma^{=d\cdot\log^3 n}$ there exists a value $j(v)$ verifying the inequalities from (i). Suppose that for some pair $v \neq v'$, $j(v) < j(v')$. Then $(u, v, a_{j(v')}, 1) <_\pi s_1 <_\pi s_3 <_\pi (u, v', a_{j(v')}, 1)$, which means that two elements from $Y_{u,a_{j(v')},1}$ are on distinct "sides" of $s_1 <_\pi s_3$. But then π cannot be a reasonable ordering by the discussion following the definitions in stage (2). It follows that for all $v \in \Sigma^{=d\cdot\log^3 n}$, $j(v)$ represents the same value. The proof of (ii) is similar. □

If π is a reasonable ordering and (u, a_j) are as in Claim 4.5 (i) and (v, b_k) are as in Claim 4.5 (ii), then we say that π *forces* answer a_j to u and answer b_k to v.

CLAIM 4.6. *Let* π *be a reasonable ordering. We define the provers' strategies* (P_1, P_2) *by* $P_1(u) = a$ *if* π *forces answer* a *to* u, *and* $P_2(v) = b$ *if* π *forces answers* b *to* v. *Then* $C_\pi(X) = 2^{d\log^3 n} \cdot ACC_{V,(P_1,P_2)}(x)$.

Proof. Observe that the contribution of all pairs whose weight is defined in stages (1)–(5) is 0. It remains to consider only the pairs whose weight is defined in stage (6). For each u, let $j(u)$ be defined by $a_{j(u)} = P_1(u)$, and for each v, let $k(v)$ be defined by $a_{k(v)} = P_2(v)$. We can see that only pairs in which the first component is from X_2 and the second component is from X_1 or in which the first component is from X_3 and

the second component is from X_1 matter. From the way the elements of X_1, X_2, and X_3 are arranged between the "stakes" s_1, s_2, s_3, and s_4 in the stages (1), (2), and (3), the first type of pair occurs for (x, y) with $x \in X_2$ and $y \in X_1,\ y > s_3$, and the second type of pair occurs for (x, y) with $x \in X_3,\ s_1 < x < s_2$, and $y \in X_1,\ s_3 < y < s_4$. Keeping in mind the definitions of $j(u)$, $k(v)$, and the "complementary" way in which weights are defined in stage (6), we get:

$$\begin{aligned} C_\pi(X) &= \sum_u \sum_v \sum_{h=1}^{N} \sum_{i=1}^{j(u)} w((u, v, b_h, 2), (u, v, a_i, 1)) \\ &\quad + \sum_u \sum_v \sum_{h=k(v)+1}^{N} \sum_{i=1}^{j(u)} w((u, v, b_h, 3), (u, v, a_i, 1)) \\ &= \sum_u \sum_v \sum_{h=1}^{k(v)} \sum_{i=1}^{j(u)} d_{u,v,a_i,b_h} = \sum_u \sum_v val_{u,v,P_1(u),P_2(v)} \\ &= 2^{d \log^3 n} \cdot ACC_{V,(P_1,P_2)}(x). \end{aligned}$$

In the above sums, u and v range over $\Sigma^{=d \cdot \log^3 n}$. □

It follows from Claim 4.6 that $\max_\pi C_\pi(X) \leq 2^{d \log^3 n} \cdot ACC_V(x)$. On the other hand, from any pair of provers' strategies (P_1, P_2), one can easily build an ordering π such that $C_\pi(X) = 2^{d \log^3 n} \cdot ACC_{V,(P_1,P_2)}(x)$. It is sufficient that π satisfy the requirements from the discussions following stages (1), (2), and (3) and that it *force* to any queries u and v the real answers of the provers P_1 and P_2. In conclusion, $\max_\pi C_\pi(X) = 2^{d \log^3 n} \cdot ACC_V(x)$. It is not difficult to check that the whole construction can be made in DTIME$[2^{\log^c n}]$ for some constant c. □

5. Final remarks. It is known that many combinatorial problems become more difficult when we pass from positive parameters to arbitrary (i.e., both positive and negative) parameters. The most notorious example is probably the SHORTEST PATH problem (see [CLR90]). Our results offer an explicit expression to this paradigm: most classes of combinatorial problems known to be approximable when dealing with positive parameters lose this property when dealing with arbitrary parameters. This contrasts with the situation of numerical problems, where the additional algebraic structure provided by arbitrary parameters (i.e., the existence of inverses) may actually help. It would be interesting to detect a general, deep explanation for this dichotomy. There are also some other less obscure questions left open by this work. It would be interesting to find nontrivial syntactically defined classes which are well approximable even with arbitrary parameters. And, finally, another topic of interest is to investigate the real-valued weighted version of some other natural problems.

Acknowledgments. I thank Luca Trevisan for telling me the reductions in Lemmas 3.3 and 3.7, which strengthened and simplified the corresponding lemmas in an earlier version of this work. I thank Erich Grädel for valuable comments and useful pointers to the literature. I am grateful to Lane Hemaspaandra, Mitsunori Ogihara, Joel Seiferas, and the anonymous referees for helping improve the presentation of this paper.

REFERENCES

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and intractability of approximation problems*, in Proc. 32nd IEEE Symposium on Foundations of Computer Science, 1992, pp. 14–23.

[BCG92] Th. Behrendt, K. Compton, and E. Grädel, *Optimization problems: Expressibility, approximation properties and expected asymptotic growth of optimal solutions*, in Computer Science Logic, Selected Papers, San Miniato, Lecture Notes in Computer Science 702, Springer-Verlag, New York, 1992, pp. 43–60.

[Bel92] M. Bellare, *Interactive Proofs and Approximation*, Technical Report IBM RC 17969, IBM Thomas J. Watson Research Center, Yorktown, NY, May 1992.

[BFL91] L. Babai, L. Fortnow, and C. Lund, *Non-deterministic exponential time has two-prover interactive protocols*, Comput. Complexity, 1 (1991), pp. 3–40.

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russell, *Efficient probabilistically checkable proofs and applications to approximation*, in Proc. 23th ACM Symposium on Theory of Computing, 1993, pp. 294–304.

[BGS95] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and non-approximability-towards tight results*, in Proc. 35th IEEE Symposium on Foundations of Computer Science, 1995, pp. 422–431. Full paper available from ECCC at http://www.eccc.uni-trier.de/eccc/.

[BS94] M. Bellare and M. Sudan, *Improved non-approximability results*, in Proc. 24th ACM Symposium on Theory of Computing, 1994, pp. 184–193.

[BYE81] R. Bar-Yehuda and S. Even, *A linear time approximation algorithm for the weighted vertex cover problem*, J. Algorithms, 2 (1981), pp. 198–203.

[Chv79] V. Chvátal, *A greedy heuristics for the set covering problem*, Math. Oper. Res., 4 (1979), pp. 233–235.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[Com88] K. Compton, 0-1 *laws in logic and combinatorics*, in NATO Adv. Study Inst. on Algorithms and Order, D. Reidel, Dordrecht, the Netherlands, 1988, pp. 353–383.

[Con91] A. Condon, *The complexity of the max word problem and the power of one-way interactive proof systems*, in Proc. 8th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science 480, Springer-Verlag, New York, 1991, pp. 456–465.

[Fag74] R. Fagin, *Generalized first-order spectra and polynomial-time recognizable sets*, in Complexity of Computation, Proceedings of SIAM-AMS Symposium in Applied Mathematics, R. Karp, ed., 1974, pp. 27–41.

[FGL+91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Approximating clique is almost NP complete*, in Proc. 31st IEEE Symposium on Foundations of Computer Science, 1991, pp. 2–12.

[FK94] U. Feige and J. Kilian, *Two prover protocols - low error at affordable rates*, in Proc. 24th ACM Symposium on Theory of Computing, 1994, pp. 172–183.

[FL92] U. Feige and L. Lovász, *Two-prover one round systems: Their power and their problems*, in Proc. 24th ACM Symposium on Theory of Computing, 1992, pp. 733–744.

[FRS88] L. Fortnow, J. Rompel, and M. Sipser, *On the power of multi-prover interactive protocols*, in Proc. 3rd IEEE Structure in Complexity Theory Conference, pp. 156–161, 1988.

[GJ79] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979.

[GM84] M. Gondran and M. Minoux, *Graphs and Algorithms*, Wiley, New York, 1984.

[GM93] E. Grädel and A. Malmström, *Approximable minimization problems and optimal solutions on random inputs*, in Computer Science Logic, 7th Workshop, CSL '93, Swansea 1993, Selected Papers, Lecture Notes in Computer Science 832, Springer-Verlag, New York, 1995, pp. 139–149.

[Gra84] E. Grandjean, *The spectra of first-order sentences and computational complexity*, SIAM J. Comput., 13 (1984), pp. 356–373.

[Hoc82] D. S. Hochbaum, *Approximation algorithms for the set covering and vertex covering problems*, SIAM J. Comput., 11 (1982), pp. 555–556.

[Imm89] N. Immerman, *Descriptive and computational complexity*, in Computational Complexity Theory, Proceedings of AMS Symposium in Applied Mathematics, J. Hartmanis, ed., AMS, Providence, RI, 1989, pp. 75–91.

[Joh92] D. Johnson, *The NP-completeness column: An ongoing guide*, J. Algorithms, 13 (1992), pp. 502–524.

[KMSV94] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani, *On syntactic versus computational views of approximability*, in Proc. 34th IEEE Symposium on Foundations of Computer Science, 1994, pp. 819–830.

[KT94] P. G. Kolaitis and M. N. Thakur, *Logical definability of NP optimization problems*, Inform. and Comput., 115 (1994), pp. 321–353.

[KT95] P. G. Kolaitis and M. N. Thakur, *Approximation properties of NP minimization classes*, J. Comput. System Sci., 50 (1995), pp. 390–411.

[Lau92] C. Lautemann, *Logical definability of NP optimization problems with monadic auxiliary predicates*, in Computer Science Logic, Selected Papers, San Miniato, Lecture Notes in Computer Science 702, Springer-Verlag, New York, 1992, pp. 327–339.

[LY93a] C. Lund and M. Yannakakis, *The approximation of maximum subgraph problems*, in Proc. 20th International Colloquium on Automata, Languages, and Programming, Springer-Verlag, Berlin, 1993, pp. 40–51.

[LY93b] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, in Proc. 23th ACM Symposium on Theory of Computing, 1993, pp. 286–293.

[Lyn82] J. Lynch, *Complexity classes and theories of finite models*, Math. Systems Theory, 15 (1982), pp. 127–144.

[PR93] A. Panconesi and D. Ranjan, *Quantifiers and approximation*, Theoret. Comput. Sci., 107 (1993), pp. 145–163.

[PS94] A. Polishchuk and D. Spielman, *Nearly-linear size holographic proofs*, in Proc. 24th ACM Symposium on Theory of Computing, 1994, pp. 194–203.

[PY91] C. Papadimitriou and M. Yannakakis, *Optimization, approximation and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.

[Zim93] M. Zimand, *The Complexity of the Optimal Spanning Hypertree Problem*, Technical Report 471, Dept. of Computer Science, Univ. of Rochester, NY, September 1993.

[Zuc93] D. Zuckerman, *NP-complete problems have a version that's hard to approximate*, in Proc. 8th IEEE Structure in Complexity Theory Conference, 1993, pp. 305–312.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 57–104

THE COMPUTATIONAL STRUCTURE OF MONOTONE MONADIC SNP AND CONSTRAINT SATISFACTION: A STUDY THROUGH DATALOG AND GROUP THEORY*

TOMÁS FEDER† AND MOSHE Y. VARDI‡

Abstract. This paper starts with the project of finding a large subclass of NP which exhibits a dichotomy. The approach is to find this subclass via syntactic prescriptions. While the paper does not achieve this goal, it does isolate a class (of problems specified by) "monotone monadic SNP without inequality" which may exhibit this dichotomy. We justify the placing of all these restrictions by showing, essentially using Ladner's theorem, that classes obtained by using only two of the above three restrictions do not show this dichotomy. We then explore the structure of this class. We show that all problems in this class reduce to the seemingly simpler class CSP. We divide CSP into subclasses and try to unify the collection of all known polytime algorithms for CSP problems and extract properties that make CSP problems NP-hard. This is where the second part of the title, "a study through Datalog and group theory," comes in. We present conjectures about this class which would end in showing the dichotomy.

Key words. satisfiability, graph coloring, datalog, group theory, linear equations

AMS subject classifications. 68Q15, 68R99

PII. S0097539794266766

1. Introduction.

We start with a basic overview of the framework explored in this paper; for an accompanying pictorial description, see Figure 1. A more detailed presentation of the work and its relationship to earlier work is given in the next section.

It is well known that if P≠NP, then NP contains problems that are neither solvable in polynomial time nor NP-complete. We explore the following question: what is the most general subclass of NP that we can define that may not contain such in-between problems? We investigate this question by means of syntactic restrictions. The logic class SNP is contained in NP and can be restricted with three further requirements: *monotonicity*, *monadicity*, and *no inequalities*. We show that if any two out of these three conditions are imposed on SNP, then the resulting subclasses of SNP are still general enough to contain a polynomially equivalent problem for every problem in NP, and in particular for the in-between problems in NP. We thus address the question by imposing all three restrictions simultaneously; the resulting subclass of SNP is called *MMSNP*.

We examine MMSNP and observe that it contains a family of interesting problems. A constraint-satisfaction problem is given by a pair I (the *instance*) and T (the *template*) of finite relational structures over the same vocabulary. The problem is satisfied if there is a homomorphism from I to T. It is well known that the constraint-satisfaction problem is NP-complete. In practice, however, one often encounters the situation where the template T is fixed and it is only the instance I that varies. We define *CSP* to be the class of constraint-satisfaction problems with

* Received by the editors April 27, 1994; accepted for publication (in revised form) October 31, 1996; published electronically June 15, 1998.
http://www.siam.org/journals/sicomp/28-1/26676.html
† IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120-6099 (tomas@theory.stanford.edu).
‡ Rice University, Department of Computer Science, Herman Brown 230, Houston, TX 77251 (vardi@cs.rice.edu).

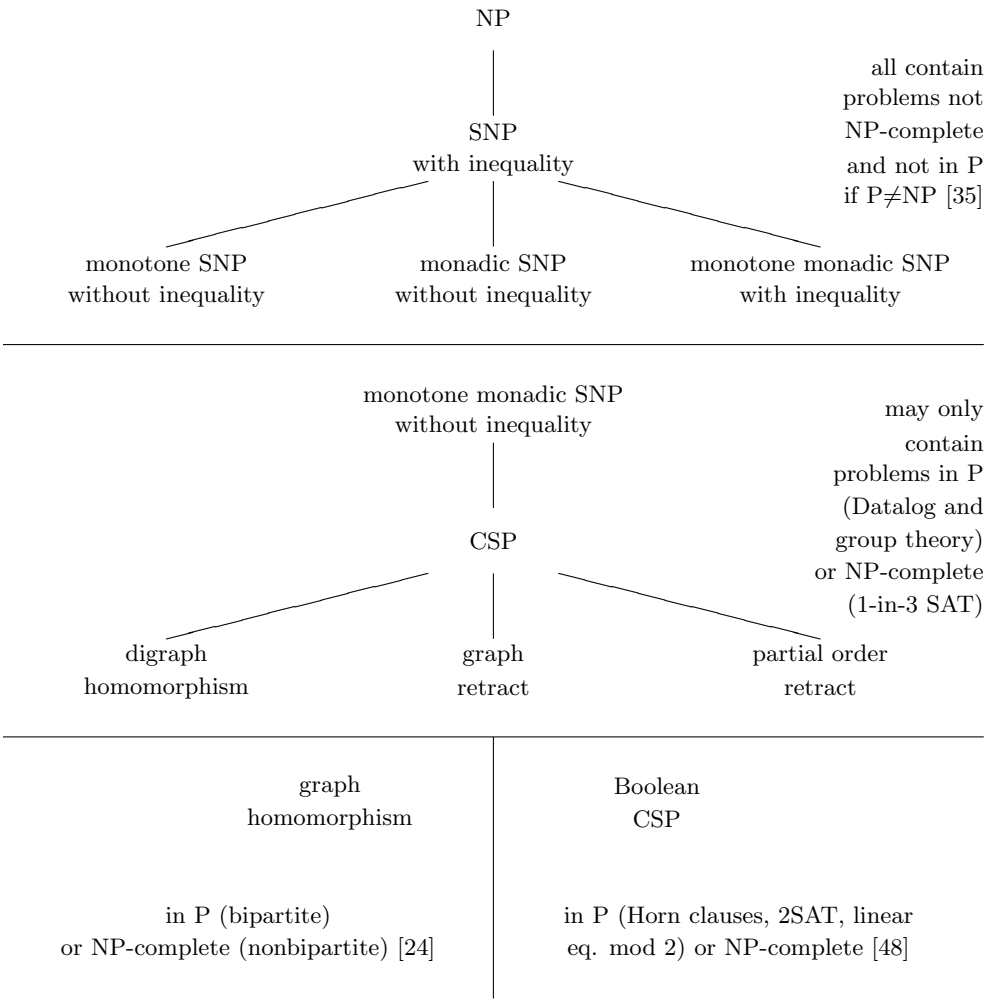


FIG. 1. *Summary.*

respect to fixed templates; that is, for every template T, the class CSP contains the problem P_T that asks for an instance I over the same vocabulary as T whether there is a homomorphism from I to T or not. The class CSP is contained in MMSNP. We show that CSP is in a sense the same as all of MMSNP: every problem in MMSNP has an equivalent problem in CSP under randomized polynomial time reductions.

The class CSP in turn has some interesting subclasses: the *graph-retract*, *digraph-homomorphism*, and *partial order-retract* problems. We show that in fact every problem in CSP has a polynomially equivalent problem in each of these three subclasses, so that all three of them are as general as CSP. Some special cases were previously investigated; for CSP with a Boolean template it was shown that there are three polynomially solvable problems, namely, Horn clauses, 2SAT, and linear equations modulo 2, while the remaining problems are NP-complete. For the graph-homomorphism problem, it was shown that bipartite graph templates are polynomially solvable and

nonbipartite graph templates are NP-complete. Could it then be that every problem in CSP is either polynomially solvable or NP-complete?

Some representative problems that were previously observed as belonging to CSP are k-satisfiability, k-colorability, and systems of linear equations modulo q; a polynomially solvable problem that can be less obviously seen to belong to CSP is labeled graph isomorphism. We notice here that at present, all known polynomially solvable problems in CSP can be explained by a combination of Datalog and group theory. More precisely, we define *bounded-width* problems as those that can be defined by Datalog programs, and *subgroup* problems as those whose relations correspond to subgroups or cosets of a given group; both subclasses are polynomially solvable. Two decidable subclasses of the bounded-width case are the *width* 1 and the *bounded strict width* problems; in fact, the three polynomially solvable cases with a Boolean template are width 1, strict width 2, and subgroup, respectively. The three main polynomial cases, namely, width 1, bounded strict width, and subgroup, are all characterized by algebraic closure properties, and so is convex programming, a polynomial constraint satisfaction problem over the reals.

We finally observe that the only way we know how to show that a problem is not bounded-width requires the problem to have a property that we call the *ability to count*. Once a problem has the ability to count, it seems that it must necessarily contain the general subgroup problem for an abelian group as a special case. When a new type of subset of a group, which we call *nearsubgroup*, is also allowed in a subgroup problem, the resulting problem reduces to subgroup problems, at least for solvable groups, and if an allowed nonsubgroup subset is not a nearsubgroup, then the subgroup problem becomes NP-complete. Does this sequence of observations lead to a classification of the problems in CSP as polynomially solvable or NP-complete?

2. Preliminaries. A large class of problems in artificial intelligence and other areas of computer science can be viewed as *constraint-satisfaction problems* [10, 34, 40, 41, 42, 43, 45]. This includes problems in machine vision, belief maintenance, scheduling, temporal reasoning, type reconstruction, graph theory, and satisfiability.

We start with some definitions. A *vocabulary* is a set $V = \{(R_1, k_1), \ldots, (R_t, k_t)\}$ of relation names and their arities. A *relational structure* over the vocabulary V is a set S together with relations R_i of arity k_i on the set S. An instance of constraint satisfaction is given by a pair I, T of finite relational structures over the same vocabulary. The instance is satisfied if there is a homomorphism from I to T; that is, there exists a mapping h such that for every tuple $(x_1, \ldots, x_k) \in R_i$ in I we have $(h(x_1), \ldots, h(x_k)) \in R_i$ in T. Intuitively, the elements of I should be thought of as variables and the elements of T should be thought of as possible values for the variables. The tuples in the relations of I and T should be viewed as constraints on the set of allowed assignments of values to variables. The set of allowed assignments is nonempty if and only if there exists a homomorphism from I to T. In what follows, we shall use the homomorphism and variable-value views interchangeably in defining constraint-satisfaction problems.

It is well known that the constraint-satisfaction problem is NP-complete. In practice, however, one often encounters the situation where the structure T (which we call the *template*) is fixed, and it is only the structure I (which we call the *instance*) that varies.

For example, the template of the 3SAT problem has domain $\{0, 1\}$ and four ternary relations C_0, C_1, C_2, C_3 that contain all triples except for $(0, 0, 0)$ in the case of C_0, except for $(1, 0, 0)$ in the case of C_1, except for $(1, 1, 0)$ in the case of C_2, and

except for $(1,1,1)$ in the case of C_3. The tuples in the instance describe the clauses of the problem. For example, a constraint $C_2(x,y,z)$ imposes a condition on the three variables x,y,z that is equivalent to the clause $\overline{x} \vee \overline{y} \vee z$.

As a second example, the template of the 3-coloring problem is the graph K_3; i.e., it has domain $\{r,b,g\}$ and a single binary relation E that holds for all pairs (x,y) from the domain with $x \neq y$. The tuples in the instance describe the edges of the graph. Thus, the variables $x_1, x_2, \ldots, x_n$ can be viewed as vertices to be colored with r,b,g, and the constraints $E(x_i,x_j)$ can be viewed as describing the edges whose endpoints must be colored differently. If we replace the template K_3 by an arbitrary graph H, we get the so-called *H-coloring* problem [24].

As a third example, given an integer $q \geq 2$, the template of the linear equations modulo q problem has domain $\{0,1,\ldots,q-1\}$, a monadic relation Z that holds only for the element 0, and a ternary relation C that holds for the triples (x,y,z) with $x+y+z=1 \pmod q$. It is easy to show that any other linear relations on variables modulo q can be expressed by introducing a few auxiliary variables and using only the Z and C relations.

In this paper we consider constraint-satisfaction problems with respect to fixed templates. We have defined *CSP* to be the class of such problems. It is easy to see that CSP is contained in NP. We know that NP contains polynomially solvable problems and NP-complete problems. We also know that if P≠NP, then there exist problems in NP that are neither in P nor NP-complete [35]. The existence of such "intermediate" problems is proved by a diagonalization argument. It seems, however, impossible to carry this argument in CSP. This motivates our main question, which follows.

Dichotomy question: *Is every problem in CSP either in P or NP-complete?*

Our question is supported by two previous investigations of constraint-satisfaction problems that demonstrated dichotomies. Schaefer [48] showed that there are essentially only three polynomially solvable constraint-satisfaction problems on the set $\{0,1\}$, namely, (1) 0-valid problems (problems where all-zeros is always a solution, and similarly 1-valid problems); (2) Horn clauses (problems where every relation in the template can be characterized by a conjunction of clauses with at most one positive literal per clause, and similarly anti-Horn clauses, with at most one negative literal per clause); (3) 2SAT (problems where every relation in the template can be characterized by a conjunction of clauses with two literals per clause); (4) linear equations modulo 2 (problems where every relation in the template is the solution set of a system of linear equations modulo 2). All constraint-satisfaction problems on $\{0,1\}$ that are not in one of these classes are NP-complete. The NP-complete cases include one-in-three SAT, where the template has a single relation containing precisely $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$, and not-all-equal SAT, where the template has a single relation that contains all triples except $(0,0,0)$ and $(1,1,1)$.

Hell and Nešetřil [24] showed that the H-coloring problem is in P if H is bipartite and NP-complete for H nonbipartite. Bang-Jensen and Hell [8] conjecture that this result extends to the digraph case when every vertex in the template has at least one incoming and at least one outgoing edge: if the template is equivalent to a cycle then the problem is polynomially solvable, otherwise NP-complete.

The issue that we address first is the robustness of the class CSP. We investigate the dichotomy question in the context of the complexity class *SNP*, which is a subclass of NP that is defined by means of a logical syntax [32, 44] and which, in particular, includes CSP. We show that SNP is too general a class to address the

dichotomy question, because every problem in NP has an *equivalent* problem in SNP under polynomial time reductions. Here two problems are said to be equivalent under polynomial time reductions if there are polynomial time reductions from one to the other, in both directions. We then impose three syntactic restrictions on SNP, namely, *monotonicity*, *monadicity*, and *no inequalities*, since CSP is contained in SNP with these restrictions imposed. It turns out that if only two of these three restrictions are imposed, then the resulting subclass of SNP is still general enough to contain an equivalent problem for every problem in NP.

When all three restrictions are imposed, we obtain the class *MMSNP*: *monotone monadic* SNP *without inequality*. This class is still more general than CSP, because it strictly contains CSP. We prove, however, that every problem in MMSNP has an equivalent problem in CSP, this time under randomized polynomial time reductions (we believe that it may be possible to derandomize the reduction).

Thus, CSP is essentially the same as the seemingly more general class MMSNP. In the other direction, there are three special cases of CSP, namely, the *graph-retract*, the *digraph-homomorphism*, and the *partial-order-retract* problems, that turn out to be as hard as all of CSP, again under polynomial time reductions. The equivalence between CSP and classes both above and below it seems to indicate that CSP is a fairly robust class.

We then try to solve the dichotomy question by considering a more practical question.

Primary classification question: *Which problems in CSP are in P and which are NP-complete?*

In order to try to answer this question, we again consider Schaefer's results for constraint-satisfaction problems on the set $\{0, 1\}$ [48]. Schaefer showed that there are only three such polynomially solvable constraint-satisfaction problems. We introduce two subclasses of CSP, namely, *bounded-width* CSP and *subgroup* CSP, respectively, as generalizations of Schaefer's three cases. Bounded-width problems are problems that can be solved by considering only bounded sets of variables, which we formalize in terms of the language Datalog [50]. Both Horn clauses and 2SAT fall into this subclass; we show that linear equations modulo 2 do not. Subgroup problems are group-theoretic problems where the constraints are expressed as subgroup constraints. Linear equations modulo 2 fall into this subclass. Not only are these subclasses solvable in polynomial time, but, at present, *all* known polynomially solvable constraint-satisfaction problems can be explained in terms of these conditions.

Assuming that these conditions are indeed the only possible causes for polynomial solvability for problems in CSP, this poses a new classification problem.

Secondary classification question: *Which problems in CSP are bounded-width problems and which are subgroup problems?*

The main issue here is that it is not clear whether membership in these subclasses of CSP is decidable.

Our results provide some progress in understanding the bounded-width and subgroup subclasses. For example, for the bounded-width problems, our results provide a classification for the 1-width problems in CSP (these are the problems that can be solved by *monadic* Datalog programs). We also identify a property of problems, which we call *the ability to count*. We prove that this property implies that the problem cannot be solved by means of Datalog. Once a constraint-satisfaction problem has the ability to count, it is still possible in many cases to solve it by group-theoretic means.

While all known polynomially solvable problems in CSP can be reduced to the bounded-width and group-theoretic subclasses, not all such problems belong to those classes from the start. For example, we show that under some conditions nonsubgroup problems can be reduced to the subgroup subclass. These conditions are stated in terms of the new notion of *nearsubgroup*, and delineating the boundary between polynomially solvable and NP-complete group-theoretical problems seems to require certain progress in finite-group theory.

The remainder of the paper is organized as follows. Section 3 introduces the logic class MMSNP as the largest subclass, in some sense, of SNP, that is not computationally equivalent to all of NP. Section 4 introduces the class CSP as a subclass of MMSNP which is essentially equivalent to MMSNP. Section 5 studies three subclasses of CSP, the graph-retract, digraph-homomorphism, and partial order-retract problems, essentially equivalent to all of CSP. Section 6 considers classes of problems in CSP that are polynomially solvable. Section 6.1 considers the bounded-width problems, which are those that can be solved by means of Datalog, and their relationship to two-player games. Sections 6.1.1 and 6.1.2 examine two special subclasses of the bounded-width case for which membership is decidable, namely, the width 1 case with its connection to the notion of tree duality, and the strict width l case with its connection to the Helly property. Section 6.2 examines which problems are not of bounded width via a notion called the ability to count. Section 6.3 considers the group-theoretic case and introduces the notion of nearsubgroup in an attempt to understand the boundary between tractability and intractability. Section 7 explores further directions for a possible complete classification. An appendix explores the connection between constraint satisfaction and Etter's link systems.

3. Monotone monadic SNP without inequality. The class SNP [32, 44] (see also [14]) consists of all problems expressible by an existential second-order sentence with a universal first-order part, namely, by a sentence of the form $(\exists S')(\forall \mathbf{x})\Phi(\mathbf{x}, S, S')$, where Φ is a first-order quantifier-free formula. That is, Φ is a formula built from relations in S and S' applied to variables in $\mathbf{x}$, by means of conjunctions, disjunctions, and negation. Intuitively, the problem is to decide, for an input structure S, whether there exists a structure S' on the same domain such that for all values in this domain for the variables in $\mathbf{x}$ it is true that $\Phi(\mathbf{x}, S, S')$ holds. We will refer to the relations of S as *input relations*, while the relations of S' will be referred to as *existential relations*. The 3SAT problem is an example of an SNP problem: the input structure S consists of four ternary relations C_0, C_1, C_2, C_3 on the domain $\{0, 1\}$, where C_i corresponds to a clause on three variables with the first i of them negated. The existential structure S' is a single monadic relation T describing a truth assignment. The condition that must be satisfied states that for all x_1, x_2, x_3, if $C_0(x_1, x_2, x_3)$ then $T(x_1)$ or $T(x_2)$ or $T(x_3)$, and similarly for the remaining C_i by negating $T(x_j)$ if $j \leq i$. We are interested in the following question.

Which subclasses of NP have the same computational power as all of NP?

That is, which subclasses of NP are such that for every problem in NP there is a problem in the subclass equivalent to it under polynomial time reductions. More precisely, we say that two problems A and B are *equivalent* under polynomial time reductions if there is a polynomial time reduction from A to B, as well as a polynomial time reduction from B to A. It turns out that every problem in NP is equivalent to a problem in SNP under polynomial time reductions. This means that for every problem A in NP, there is a problem B in SNP such that there is a polynomial time reduction from A to B, as well as a polynomial time reduction from B to A.

In fact, we now show that this is the case even for restrictions of SNP. We start by assuming that the equality or inequality relations are not allowed in the first-order formula, only relations from the input structure S or the existential structure S'. For *monotone* SNP *without inequality*, we require that all occurrences of an input relation C_i in Φ have the same polarity. (The polarity of a relation is positive if it is contained in an even number of subformulas with a negation applied to it, and it is negative otherwise.) By convention, we assume that this polarity is negative, so that the C_i can be interpreted as constraints, in the sense that imposing C_i on more elements of the input structure can only make the instance "less satisfiable." Note that 3SAT as described above has this property. For *monadic* SNP *without inequality*, we require that the existential structure S' consist of monadic relations only. This is again the case for 3SAT described above. For *monotone monadic* SNP *with inequality*, we assume that the language also contains the equality relation, so both equalities and inequalities are allowed in Φ. (If we consider that equalities and inequalities appear with negative polarity, then only inequalities give more expressive power, since a statement of the form "if $x = y$ then $\Phi(x, y)$" can be replaced by "$\Phi(x, x)$.")

We have thus taken the class SNP, and we are considering three possible syntactic restrictions, namely, *monotonicity*, *monadicity*, and *no inequality*. We shall later be especially interested in SNP with all three syntactic restrictions imposed. However, for now, we are only considering the cases where only two of these three syntactic restrictions are simultaneously imposed.

THEOREM 1. *Every problem in NP has an equivalent (under polynomial time reductions) problem in monotone monadic SNP with inequality.*

Proof. Hillebrand et al. [30] showed that monadic Datalog with inequality (but without negation) can verify a polynomial time encoding of a Turing machine computation; the machine can be nondeterministic. A Datalog program is a formula Φ that consists of a conjunction of formulas of the form $R_0(\mathbf{x}_0) \leftarrow R_1(\mathbf{x}_1) \wedge \cdots \wedge R_k(\mathbf{x}_k)$, where the $\mathbf{x}_i$ may share variables. The relation R_0 cannot be an input relation, and monadicity here means that R_0, as a relation that is not an input relation, must be monadic or of arity zero; furthermore an R_i may be an inequality relation. There is a particular $\hat{R}_0$ of arity zero that must be derived by the program in order for the program to accept its input; this means that the input is rejected by the Datalog program if $(\exists \mathbf{R})(\forall \mathbf{x})(\Phi(\mathbf{R}, \mathbf{S}, \mathbf{x}) \wedge \neg \hat{R}_0)$. Notice that this formula Φ' is a monotone monadic SNP with inequality formula. Here $\mathbf{S}$ describes the computation of a nondeterministic Turing machine, including the input, the description of the movement of the head on the tape of the machine, and the states of the machine and cell values used during the computation. We would now like to assume that the computation of the machine is not known ahead of time. That is, only the input to the machine is given; the movement of the head and the cell values are not known and are quantified existentially. Unfortunately, the description of the movement of the head does not consist of monadic relations and may depend on the input to the machine. We avoid this difficulty by assuming that the Turing machine is *oblivious*; i.e., the head traverses the space initially occupied by the input back and forth from one end to the other, and accepts in exactly n^k steps for some k. We can then assume that the movement of the head is given as part of the input, since it must be independent of the input for such an oblivious machine. Thus only the states of the machine and cell values used during the computation must be quantified existentially, giving a monotone monadic SNP with inequality formula that expresses whether the machine accepts a given in-

put or not. A particular computation is thus described by a choice of states and cell values, which are described by monadic existential relations that are then used as inputs to the Datalog program. The condition that must be satisfied is that if a state is marked as being the (n^k)th state (this is determined by a deterministic component of the machine), then it must also be marked as being an accepting state (this depends on the nondeterministic choice of computation). The monotone monadic SNP with inequality formula will thus reject an instance if it does not describe an input followed by the correct movement of the head for the subsequent oblivious computation, accept the instance if the number of cells allowed for the computation is smaller than n^k, and otherwise accept precisely when the machine accepts. □

THEOREM 2. *Every problem in NP has an equivalent (under polynomial time reductions) problem in monadic SNP without inequality.*

Proof. Since the existence of an equivalent problem in monotone monadic SNP with inequality for every problem in NP was previously shown, it is sufficient to remove inequalities at the cost of monotonicity.

To remove inequalities at the cost of monotonicity, introduce a new binary input relation *eq*, augment the formula by a conjunct requiring *eq* to be an equivalence relation with the property that if an input or existential monadic relation holds on some elements, then it also holds when an element in an argument position is replaced by an element related to it under *eq*; finally, replace all occurrences of $x \neq y$ by $\neg eq(x, y)$. Thus the formula no longer contains inequalities, but it contains an input relation that appears with both positive and negative polarity; i.e., it is no longer monotone. The formula is therefore a monadic SNP without inequality formula. □

THEOREM 3. *Every problem in NP has an equivalent (under polynomial time reductions) problem in monotone SNP without inequality.*

Proof. Since the existence of an equivalent problem in monotone monadic SNP with inequality for every problem in NP was previously shown, it is sufficient to remove inequalities at the cost of monadicity.

To remove inequalities at the cost of monadicity, the intuition is that up to equivalence, certain *marked* elements form a *succ* path with *pred* as its transitive closure. Introduce a monadic input relation *special*, a binary input relation *succ*, a monadic existential relation *marked*, a binary existential relation *eq*, and a binary existential relation *pred*. Require now that every *special* element be *marked*, and every element related to a *marked* element under *succ* (in either direction) be *marked*. Require that *pred* be transitive but not relate any element to itself, that two elements related by *succ* be related by *pred* (in the same direction), that *eq* be an equivalence relation, that any two *special* elements be related by *eq*, that *pred* be preserved under the replacement of an element by an element related to it by *eq*, and that if two elements are related by *eq* and if each has a related element under *succ* (in the same direction), then these two other elements are also related by *eq*. Finally, restrict the original formula to *marked* elements, replace $x \neq y$ by $\neg eq(x, y)$, and consider that a relation holds on some elements if it is imposed on elements related to them by *eq*. Note that on elements that are forced to be *marked*, the relations *eq* and *pred* can be defined in at most one way, giving a *succ* path (up to *eq*, with *pred* as its transitive closure). □

From these three theorems, by Ladner's result [35] that if P≠NP then there exist problems in NP that are neither in P nor NP-complete, Theorem 4 follows.

THEOREM 4. *If* P $\neq$ NP*, then there are problems in each monotone monadic SNP with inequality, monadic SNP without inequality, and monotone SNP without inequality, that are neither in* P *nor* NP*-complete.*

We now consider the class *MMSNP*, which is *monotone monadic SNP, without inequality*. That is, in MMSNP we impose all three restrictions simultaneously (instead of just two at a time as in the three subclasses of SNP considered above). It seems impossible to carry out Ladner's diagonalization argument in MMSNP. Thus, the dichotomy question from the introduction also applies to this class.

4. Constraint satisfaction. Let S and T be two finite relational structures over the same vocabulary. A *homomorphism* from S to T is a mapping from the elements of S to elements of T such that all elements related by some relation C_i in S map to elements related by C_i in T. If T is the substructure of S obtained by considering only relations on a subset of the elements of S, and the homomorphism h from S to T is just the identity mapping when restricted to T, then h is called a *retraction*, and T is called a *retract* of S. If no proper restriction T of S is a retract of S, then S is a *core*; otherwise its core is a retract T that is a core. It is easy to show that the core of a structure S is unique up to isomorphism.

Here a *constraint-satisfaction problem* (or *structure-homomorphism problem*) will be a problem of the following form. Fix a finite relational structure T over some vocabulary; T is called the *template*. An *instance* is a finite relational structure S over the same vocabulary. The instance is satisfied if there is a homomorphism from S to T. Such a homomorphism is called a *solution*. We define *CSP* to be the class of constraint-satisfaction problems. We can assume that T is a core and include a copy of T in the input structure S, so that the structure-homomorphism problem is a structure-retract problem.

Remark. It is possible to define constraint satisfaction with respect to infinite templates. For example, digraph acyclicity can be viewed as the question of whether a given digraph can be homomorphically mapped to the transitive closure of an infinite directed path. We will not consider infinite templates in this paper. If we allow infinite structures T, then the constraint-satisfaction problems are just the problems whose complement is closed under homomorphisms, with the additional property that an instance with satisfiable connected components is satisfiable. Note that all problems in monotone SNP, without inequality, have a complement closed under homomorphisms.

We shall later show that CSP is strictly contained in MMSNP, and that every problem in MMSNP has an essentially equivalent problem in CSP, up to randomized polynomial time reductions. To obtain this result, we need a preliminary result that will often be used later in the paper as well.

We say that a structure S has *girth* greater that k if for any choice of at most k occurrences of relations R_i of arity r_i in S, the total number of elements mentioned by these k occurrences is at least $1 + \sum(r_i - 1)$. That is, k or fewer occurrences of relations never form a cycle.

THEOREM 5. *Fix two integers k, d. Then for every structure S on n elements there exists a structure S' on n^a elements (where a depends only on k and d) such that the girth of S' is greater than k, there is a homomorphism from S' to S, and for every structure T on at most d elements over the same vocabulary as S, there is a homomorphism from S' to T if and only if there is a homomorphism from S to T. In brief, every instance S of the constraint-satisfaction problem defined by T can be replaced by an instance S' of high girth. Furthermore, S' can be constructed from S in randomized polynomial time.*

Proof. The transformation that enforces large girth is an adaptation of a randomized construction of Erdős [11] of graphs with large girth and large chromatic number. Given a structure S on n elements, define S' by making $N = n^s$ copies of each element, where s is a large constant. If a relation R of arity r was initially imposed on some r elements, then it could a priori be imposed on N^r choices of copies. Impose R on each such choice with probability $N^{1-r+\epsilon}$, where ϵ is a small constant. We thus expect to impose R on $N^{1+\epsilon}$ copies. If R has arity $r = 1$, impose R on all copies of the element.

Finally, remove one relation from each cycle with at most k relations, i.e., minimal sets of relations R_i of arity r_i involving $t \leq \sum(r_i - 1)$ elements all together. Now, given such a cycle, it must correspond to a cycle that existed before the copies were made. The number of possible such short cycles is at most n^a for some constant a. Each such short cycle could occur in N^t choices of copies. For each such choice, the probability that it occurs is $\prod N^{1-r_i+\epsilon}$, so the expected number of occurrences is $n^a N^t \prod N^{1-r_i+\epsilon} \leq N^{a/s+k\epsilon} = N^{\epsilon'}$, and hence no more than twice this much with probability at least $1/2$ by Markov's inequality.

It is clear that if before making copies, there is a homomorphism from S to some T, then there is a homomorphism from S' to T as well: S' maps to S by a homomorphism. To obtain a converse, suppose that a homomorphism maps S' to T. If we consider the N copies of a particular element, then at least N/d of the copies agree on the image in T. If we select these copies, for each of the n elements, then the expected number of copies of a relation R of arity r is $(N/d)^r N^{1-r+\epsilon} = N^{1+\epsilon}/b$ for some constant b, and hence the probability that the number of copies is not even half this much is only $e^{-N^{1+\epsilon}/c}$ for some constant c by the Chernoff bound, since the occurrences of copies are independent. The total number of occurrences of relations R in the instance is n^r, and the number of possible choices of subsets of size N/d for the copies of the elements involved is at most 2^{rN}, and hence the probability that some choice of subsets will involve only $N^{1+\epsilon}/2b$ copies of some relation is at most $n^r 2^{rN} e^{-N^{1+\epsilon}/c}$, hence very small. Once $N^{1+\epsilon}/2b$ copies are present, the removal of $2N^{\epsilon'}$ of them is insignificant, provided s is large enough and ϵ is small enough. Therefore any choice of values in T for elements that appears on N/d of the copies will give a mapping from S to T. □

It remains open to derandomize this construction. The key question seems to be whether the construction of Erdős can be derandomized, i.e., whether given a fixed integer k, for integers n, there is a deterministic algorithm running in time polynomial in n that produces a graph of size polynomial in n, chromatic number at least n, and girth at least k. It may be possible to use quasi-random graphs for this purpose.

It is easy to see that CSP is contained in MMSNP. Let T be a template. Then there is a monadic monotone existential second-order sentence ϕ_T (without inequality) that expresses the constraint-satisfaction problem defined by T. For each element a in the domain of the template T, we introduce an existentially quantified monadic relation T_a; intuitively, $T_a(x)$ indicates that a variable x has been assigned value a by the homomorphism. The sentence ϕ_T says that the sets T_a are disjoint and that the tuples of S satisfy the constraints given by T.

It can be shown that CSP is strictly contained in MMSNP. Nevertheless, as the following two theorems show, in terms of the complexity of its problems, CSP is just as general as MMSNP.

We begin with a simple example of an MMSNP problem that is not a constraint-satisfaction problem: testing whether a graph is triangle-free. If it were a constraint-

satisfaction problem, there would have to exist a triangle-free graph to which one can map all triangle-free graphs by a homomorphism. This would require the existence of a triangle-free graph containing as induced subgraphs all triangle-free graphs such that all nonadjacent vertices are joined by both a path of length 2 and a path of length 3 (since a homomorphism can add edges or collapse two vertices); there are $2^{\Omega(n^2)}$ such graphs on n vertices, forcing T to grow exponentially in the size of S. On the other hand, this monotone monadic SNP problem can be solved in polynomial time, and is hence equivalent to a trivial constraint-satisfaction problem.

A more interesting example is the following: testing whether a graph can be colored with two colors with no monochromatic triangle (it can easily be related to the triangle-free problem to show that it is not a constraint-satisfaction problem). However, it can be viewed as a special case of not-all-equal 3SAT, where each clause is viewed as a triangle, and it is essentially equivalent to this NP-complete constraint-satisfaction problem.

THEOREM 6. *CSP is contained in MMSNP. Every problem in MMSNP is polynomially equivalent to a problem in CSP. The equivalence is by a randomized Turing reduction from the CSP problem to the MMSNP problem and by a deterministic Karp reduction from the MMSNP problem to the CSP problem.*

Proof. We just saw that CSP is contained in MMSNP; we prove the second statement. In fact, we can use a randomized Karp reduction if we only consider connected instances of the constraint-satisfaction problem; disconnected instances simply require a solution for each connected component.

Consider an MMSNP problem that asks, for an input structure S, whether there exists a monadic structure S' such that for all $\mathbf{x}$, $\Phi(\mathbf{x}, S, S')$. We write Φ in conjunctive normal form, or more precisely, as a conjunction of negated conjunctions. We can assume that each negated conjunction describes a biconnected component. For consider first the disconnected case, so that we have a conjunct of the form $\neg(A(\mathbf{x}) \wedge B(\mathbf{y}))$, where $\mathbf{x}$ and $\mathbf{y}$ are disjoint variable sets. We can then introduce an existential zero-adic relation p and write instead $(A(\mathbf{x}) \to p) \wedge (B(\mathbf{y}) \to \neg p)$. The case where A and B share a single variable z is treated similarly; we introduce an existential monadic relation q, and replace $\neg(A(\mathbf{x}, z) \wedge B(\mathbf{y}, z))$ by $(A(\mathbf{x}, z) \to q(z)) \wedge (B(\mathbf{y}, z) \to \neg q(z))$. If the conjunction cannot be decomposed into either two disconnected parts, or two parts that share a single articulation element z, we say that it describes a biconnected component. Before carrying out this transformation, we assume for each negated conjunction that every replacement of different variables by the same variable is also present as a negated conjunction. For instance, if $\neg A(x, y, z, t, u)$ is present, then so is $\neg A(x, x, z, z, u)$. This must be enforced beforehand, since biconnected components may no longer be biconnected when distinct variables are collapsed. We also assume that if an input relation R appears with all arguments equal, say, as $R(x, x, x)$, then it is the only input relation in the negated conjunction; otherwise, we can introduce an existential monadic p, replace such an occurrence of R by $p(x)$, and add a new condition $R(x, x, x) \to p(x)$; in other words, x is in this case an articulation element.

The next transformation is the main step; we enforce that each negated conjunction contains at most one input relation, and that the arguments of this input relation are different variables. For each negated conjunction, introduce a relation R whose arity is the number of distinct variables in the negated conjunction. Intuitively, this relation stands for the conjunction C of all input relations appearing in the negated conjunction. We replace the conjunction C by the single relation R; also, if the corresponding conjunction C' for some other negated conjunction is a subconjunction of C,

then we include the negated conjunction obtained by replacing C' with the possibly longer C, using new variables if necessary, and then replace C by R. Here we are not considering the case where it might be necessary to replace two arguments in R by the same argument; this will be justified because such instantiations were handled beforehand.

We must argue that the new MMSNP problem of the special form is equivalent to the original problem. Clearly, every instance of the original problem can be viewed as an instance of the new problem, simply by introducing a relation R on distinct input elements whenever the conjunction C that is represented by R is present on them in the input instance.

On the other hand, the converse is not immediately true. If we replace each occurrence of R by the appropriate conjunction C, then some additional occurrences of R may be implicitly present. Consider, for example, the case where triangles $E(x,y) \wedge E(y,z) \wedge E(z,x)$ have been replaced by a single ternary relation $R(x,y,z)$. Then an instance of the new problem containing $R(x_1,y_1,x_2)$, $R(x_2,y_2,x_3)$, and $R(x_3,y_3,x_1)$ also contains the triangle represented by $R(x_3,x_2,x_1)$, when each R is replaced by the conjunction that it stands for.

To avoid such hidden occurrences of relations, we show that every instance of the new problem involving the R relations can be transformed into an equivalent instance of large *girth*; the girth is the length of the shortest cycle. Fix an integer k larger than the number of conjuncts in any conjunction C that was replaced by an R. We shall ensure that for any choice of at most k occurrences of relations R_i of arity r_i in the instance, the total number of elements mentioned by these k occurrences is at least $1+\sum(r_i-1)$, so the girth is greater than k; this implies that such k occurrences define an acyclic substructure, so any biconnected R' implicitly present in the union of k such occurrences must be entirely contained in one of the R_i, and then the condition stated by R' was already stated for this R_i as well.

Large girth can be enforced by applying the previous theorem, provided the new problem can be viewed as a constraint-satisfaction problem. The new problem is very close to a constraint-satisfaction problem. In fact, it is a constraint-satisfaction problem if there are no zero-ary existential relations: construct the structure T by introducing one element for each combination of truth assignments for existential monadic relations on a single element, except for those combinations explicitly forbidden by the formula; impose a relation R on all choices of elements in T except for those combinations explicitly forbidden by the formula. To remove the assumption that there are no existential zero-ary relations, do a case analysis on the possible truth assignments for such relations, and make T the disjoint union of the T_i obtained in the different cases. The only difficulty here is that we must ensure that a disconnected instance still maps to a single T_i, so we introduce a new binary relation that holds on all pairs of elements from the same T_i, and consider only connected instances of the constraint-satisfaction problem. This concludes the proof. As mentioned before, solving disconnected instances is equivalent to solving all connected components of the constraint-satisfaction problem. □

The construction in the last theorem can also be used to show the following result, which will be proven useful later. The *containment* problem asks whether given two problems A and B over the same vocabulary, every instance accepted by A is also accepted by B.

THEOREM 7. *Containment is decidable for problems in MMSNP.*

Proof. (This problem becomes undecidable when the antecedent of the containment is generalized to monotone binary SNP without inequality, using Datalog to

encode Turing machines as before.) To decide whether A is contained in B for MMSNP problems, first assume that A and B are written in the canonical form involving biconnected components from the above proof. Also remove the existential quantifier in A (since it is in the antecedent of an implication), so that A is now a universal formula which is monotone except for monadic relations. Now, if B has an instance with no solution that satisfies A, then making copies of elements of the instance as before, we can assume that the only biconnected components that arise are those explicitly stated in the conditions for B, so go through the forbidden biconnected components stated in A and remove all negated conjunctions in B that mention them (since the stated condition will never arise on instances satisfying A). Here we must assume that B stated explicitly, for each element mentioned in a negated conjunction, which monadic relations are true or false. Now we can assume that A holds, and we are left to decide whether B is a tautology; this can be decided by considering the instance consisting of one element for each possible combination of truth and falsity of monadic input relations, and then imposing all other kinds of relations on all elements. □

5. Graphs, digraphs, partial orders. We have seen that CSP has the same computational power as all of MMSNP. We ask the following question.

Which subclasses of CSP have the same computational power as all of CSP?

The *graph-retract problem* is an example of a constraint-satisfaction problem. Fix a graph H, and for an input graph G containing H as a subgraph, ask whether H is a retract of G. (Note that when G and H are disjoint, we get the *graph-homomorphism* or H-coloring problem mentioned in the introduction [24].)

The *digraph-homomorphism problem* is another example of a constraint-satisfaction problem: this is the case where the template is a digraph. For an oriented cycle (cycle with all edges oriented in either direction), the *length* of the cycle is the absolute value of the difference between edges oriented in one direction and edges oriented in the opposite direction. A digraph is *balanced* if all its cycles have length zero; otherwise it is *unbalanced*. The vertices of balanced digraphs are divided into levels, defined by $level(v) = level(u) + 1$ if (u, v) is an edge of the digraph.

A *partial order* is a set with a reflexive antisymmetric transitive relation $\leq$ defined on it. If reflexive is replace by antireflexive, we have a *strict partial order*. We may also consider homomorphism and retract problems for partial orders.

THEOREM 8. *Every constraint-satisfaction problem is polynomially equivalent to a bipartite graph-retract problem.*

Proof. First ensure that the structure T defining the problem in CSP is a core. This ensures that each element in T is uniquely identifiable by looking at the structure T, up to isomorphisms of T; i.e., we can include a copy of T in an instance S and then assume that the elements of the copy of T in S must map to the corresponding elements of T. Next, assume that T can be partitioned into disjoint sets A_j so that for each relation C_i, the possible values for each argument come from a single A_j, and the possible values for different arguments come from different A_j; this can be ensured by making copies A_j of the set of elements of T and allowing an equality constraint between copies of the same element in different A_j. Now, the bipartite graph H consists of a single vertex for each A_j which is adjacent to vertices representing the elements of A_j; a single vertex for each relation C_i which is adjacent to vertices representing the tuples satisfying C_i; a bipartite graph joining the tuples coming from each C_i to the elements of the tuples from the A_j; and two additional adjacent vertices, one of them adjacent to all the elements of sets A_j, and the other one adjacent to all the tuples for conditions C_i.

To see that the resulting retract problem on graphs is equivalent to the given constraint-satisfaction problem, observe in one direction that an instance of the constraint-satisfaction problem can be transformed into an instance of the graph-retract problem, by requiring each element to range over the copy A_1 (just make it adjacent to the vertex for A_1 in H), and then to impose a constraint C_i of arity r on some elements, create a vertex adjacent to the vertex for C_i, and make this vertex adjacent to r vertices, each of which is adjacent to the vertex for the appropriate A_j (the r values for j are distinct); then make sure that the value chosen in A_j is the same as the value for the intended element in A_1, using an equality constraint. In the other direction, an instance of the graph-retract problem can be assumed to be bipartite, since H is bipartite; furthermore, each vertex can be assumed to be adjacent to either an A_j or C_i, since all other vertices can always be mapped to the two additional vertices that were added for H at the end of the construction. Then each vertex adjacent to vertex A_j can be viewed as an element ranging over A_j, and each vertex adjacent to vertex C_i can be viewed as the application of C_i on certain elements. □

Given a bipartite graph H, we say for two vertices x, y on the same side of H that x *dominates* y if every neighbor of y is a neighbor of x. We say that H is *domination-free* if it has no $x \neq y$ such that x dominates y. If R is a graph, we say that H is R-free if H contains no induced subgraph isomorphic to R. We are interested in the cases where R is either $K_{3,3}$, a complete bipartite graph with three vertices on each side, or $K_{3,3}$ minus a single edge. The following result will be used later in the study of the partial order-retract problem.

THEOREM 9. *Every constraint-satisfaction problem is polynomially equivalent to a domination-free, $K_{3,3}$-free, $K_{3,3}\backslash\{e\}$-free bipartite graph-retract problem.*

Proof. We first show that every constraint-satisfaction problem is equivalent to a $K_{3,3}$-free, $K_{3,3}\backslash\{e\}$-free bipartite graph-retract problem. We then show how domination-freedom can in addition also be achieved.

We know that every constraint-satisfaction problem can be encoded as a bipartite graph-retract problem. To achieve $K_{3,3}$-freedom and $K_{3,3}\backslash\{e\}$-freedom, we encode the bipartite graph-retract problem again as a bipartite graph-retract problem, by reusing essentially the same reduction.

So we are given a bipartite graph-retract problem with template $H = (S, T, E)$, which we shall show polynomially equivalent to another bipartite graph-retract problem with template $H' = (U, V, F)$. We introduce five new elements r, s, t, s', t', and define H' by $U = \{r\} \cup S \cup T$, $V = \{s, t, s', t'\} \cup E$, and $F = (\{r\} \times (\{s', t'\} \cup E)) \cup (S \times \{s, s'\}) \cup (T \times \{t, t'\}) \cup \{(u, e) : u \in S \cup T, e \in E, u \in e\}$.

Let $G = (S', T', E')$ be an instance for H. (We can assume that G is bipartite since it otherwise cannot map to H, and that we know that S' maps to S and T' maps to T because it is connected to the subgraph H; any other component of G can be mapped to a single edge of H.) We define an instance $G' = (U', V', F')$ for H' by letting $U' = \{r\} \cup S' \cup T'$, $V' = \{s, t, s', t'\} \cup E'$, and defining F' by letting r be adjacent to all of E', s adjacent to all of S', t adjacent to all of T', and each $e \in E'$ adjacent to the two vertices in $S' \cup T'$ it joins in G. It is immediate in the instance G' for H' that S' must map to S, T' to T, and E' to E with $e \in E'$ mapping to the element of E joining the images of the two vertices incident on e in G, so the retractions mapping G to H and those mapping G' to H' correspond to each other.

In the other direction, let $G' = (U', V', F')$ be an instance for H'. Since s', t' dominate s, t, respectively, no element need ever be mapped to s, t, other than s, t

themselves. So we can require that the neighbors of s, t map to elements of S, T, respectively, and then remove s, t from H' and G'. We can then assume that every element of U' that is not required to map to S or T maps to r, since r is adjacent to what remains of V. We can now remove r from H' and G'. Now if a vertex in V' is only adjacent to vertices that map to S we map it to s', if only to vertices that map to T, we map it to t', and if to both, it must map to E, thus defining a retract problem instance for H.

It only remains to show that H' is $K_{3,3}$-free and $K_{3,3}\backslash\{e\}$-free, and then to enforce dominance-freedom. Suppose that H' contains H_0, which is either a $K_{3,3}$ or a $K_{3,3}\backslash\{e\}$. Then every vertex v in the right side of H_0 must belong to the 3-side of a $K_{2,3}$. This immediately gives $v \neq s, t$ because any pair of neighbors of s is only adjacent to s, s', and similarly for t. So we can remove s, t in looking for H_0. Two vertices in S, T, respectively, share only one neighbor, so H_0 involves at most one of S, T, and we can remove one of them, say T. Two vertices in S have only s' as a common neighbor, so H_0 can have at most one vertex u in S. But this leaves only two vertices r, u in one side, so there is no H_0.

The last step enforces dominance-freedom. Suppose that x dominates y. Let H_1 be the graph consisting of an 8-cycle $C_8 = (1, 2, 3, 4, 5, 6, 7, 8)$, a 4-cycle $C_4 = (1', 2', 3', 4')$, and additional edges joining each i' to both i and $i + 4$. We join H_1 to H', with $y = 1'$ as common vertex in H_1 and H'. This does not introduce any new dominated vertices, and y is no longer dominated. Furthermore H_1 contains no $K_{3,2}$. So we only need to show that joining an H_1 at a vertex y gives an equivalent retract problem. If an instance for H' maps to H', it also maps to H' with H_1 joined; if it maps to H' with H_1 joined, since no vertices are forced to map to H_1 other than y, we can map all of H_1 to y and one of its neighbors in H', so the instance maps to H'. In the other direction, consider an instance for H' with H_1 joined. Certain vertices are required to map to specific vertices in C_8. If a vertex is adjacent to two vertices at distance 2 in C_8, then it must map to their unique common neighbor in C_8. So if a vertex v is adjacent to vertices in C_8, we may assume it is adjacent to either just one vertex in C_8 or two opposite vertices in C_8; in either case, we may assume that such a v maps to the unique vertex in C_4 having these adjacencies, and remove C_8 from the template. So the template is now H' with C_4 joined at a vertex $y = 1'$. Now for $3'$, we may insist that its neighbors map to $\{2', 4'\}$, and no other vertex maps to $3'$ since $1'$ now dominates $3'$, so we may remove $3'$ from the graph. Then if a vertex is labeled $2'$, $4'$, or $\{2', 4'\}$, its neighbors must map to $1'$, and we may remove $2'$ and $4'$ from the graph since every neighbor of $y = 1'$ in H' now dominates them. So we have reduced the instance for H' with H_1 joined to an instance for H' alone, as desired. □

THEOREM 10. *Every constraint-satisfaction problem is polynomially equivalent to a balanced digraph-homomorphism problem.*

Proof. We encode the graph-retract problem as a balanced digraph-homomorphism problem. Draw the bipartite graph with one vertex set on the left and the other on the right, and orient the edges from left to right. What remains is to distinguish the different vertices on each side. We describe the transformation for vertices in the right; a similar transformation is carried out for vertices in the left. In an oriented path, let 1 denote a forward edge and 0 a backward edge. If there are k vertices $0, 1, \ldots, k-1$ on the right, starting at the ith vertex, add an oriented path $(110)^i 1 (110)^{k-i-1} 11$. The intuition is that none of these paths maps to another one of them, and that if a digraph maps to two of them, then it maps to $(110)^k 11$, hence to all of them. Fur-

thermore, the question of whether a digraph maps to an oriented path is polynomially solvable; see sections 6.1.1 and 6.1.2. □

THEOREM 11. *Every constraint-satisfaction problem is polynomially equivalent to an unbalanced digraph-homomorphism problem.*

Proof. First assume that the given constraint-satisfaction problem consists of a single relation R of arity k; multiple relations can always be combined into a single relation by taking their product and adding their arities. Now, define a new constraint-satisfaction problem whose domain consists of k-tuples from the original domain; thus R is now a monadic relation. In order to be able to state an equality constraint among different components of different tuples, define a "shift" relation $S(t, t')$ on tuples $t = (x_1, x_2, \ldots, x_{k-1}, y)$ and $t' = (z, x_1, x_2, \ldots, x_{k-1})$; one can use such shifts to state that certain components of certain tuples coincide.

We have thus reduced the general constraint-satisfaction problem to a single monadic and a single binary relation. It is clear that any instance of the original problem can be represented using tuples on which the constraint R is imposed, and the relation S allows us to state that components of different tuples take the same value; similarly, the new problem only allows us to impose constraints from the original problem. We wish to have a single binary relation alone, i.e., a digraph. Define the following dag D. It has vertices corresponding to the tuples from the constraint-satisfaction problem just constructed. For each relation $S(t, t')$ that holds, introduce a new vertex joined by a path of length 1 to t and by a path of length 2 to t'. For each relation $R(t)$ that holds, introduce a new vertex joined by a path of length 3 to t. This completes the dag.

We show that the digraph homomorphism problem for D is equivalent to the original constraint-satisfaction problem. In one direction, given an instance of the original problem involving S and R, tag each element with a reverse path of length 2 followed by a path of length 1 followed by a reverse path of length 2. This ensures that the element can be mapped precisely to vertices in D representing elements of the domain; note here that we are using the fact that each t' is related to some t by S in the domain of the constraint-satisfaction problem. To state $S(t, t')$ and $R(t)$ on such elements, use incoming paths of lengths 1, 2, 3 as for D above. In the other direction, suppose that we have an instance of the digraph-homomorphism problem. We can assume that the instance is a dag, since D is a dag. We can also assume that if a vertex has both incoming and outgoing edges, then it has only a single incoming and a single outgoing edge, because this holds in D, so we could always collapse neighbors to enforce this. The input dag now looks like a bipartite graph (A, B, P), with disjoint (except at their endpoints) paths of different lengths joining vertices in A to vertices in B (all in the same direction). We can assume that the paths have length at most 3, since D has no path of length 4. We can also assume that a vertex at which a path of length 3 starts necessarily starts just this path, because this is the case in D. We can also assume that a vertex can start at most a single path of length 2, since this is the case in D. We also assume that a vertex starts at most a single path of length 1; the only way two different paths of length 1 could go in different directions would be if one of them mapped on the path of length 2 out of an out-degree-2 vertex v in D; but then, since the endpoint has no outgoing edges, all its neighbors would necessarily map to v, and so we could have mapped this endpoint to the other neighbor of v along the path of length 1. We can also assume that the only vertices that will map to an attached path of length 3 are vertices on a path of length 3; the reason is that if a directed graph containing no path of length 3 can be mapped to a reverse path of

length 3, then it can be mapped to a reverse path of length 2 followed by a path of length 1 followed by a reverse path of length 2, and this configuration can be found in D from a fixed endpoint of a path of length 3 without using this path (we used this same configuration before). We can now assume that vertices in A and B map to vertices in the two corresponding sides of the bipartite graph corresponding to D. For vertices in B, this is clear if they have incoming paths of length 3, or of length 2 since we have assumed that they do not map to a vertex inside a path of length 3, or of length 1 since we can assume that they do not map to a vertex inside a path of length 2; the same is then clear for vertices in A. But then the vertices in B can be viewed as elements of the original constraint-satisfaction problem, and the vertices in A can be viewed as imposing constraints on them. □

THEOREM 12. *Every constraint-satisfaction problem is polynomially equivalent to a bipartite graph-retract problem, but now only allowing three specific vertices of the template H to occur in the input G (but not just two vertices, which is polynomially solvable).*

Proof. We encode a digraph-homomorphism problem. The encoding introduces three special vertices r, b, g, which may occur in G, three additional vertices r', b', g' (which cannot appear in G), and a vertex a adjacent to r', b', g'; it replaces each vertex of the dag with a vertex adjacent to r and each edge in the dag with a path $0, 1, 2, 3, 4, 5, 6$ of length 6, where the intermediate vertices in positions 2 and 4 are adjacent to b and g, respectively, while those in positions $1, 3, 5$ are adjacent to a; and, finally, it links r', b', g' to all the vertices linked to r, b, g, respectively. The proof is here a straightforward encoding argument.

A bipartite graph with just two distinguished vertices can always be retracted to just a path joining the two vertices, namely, a shortest such path; this is then the core, which defines a polynomially solvable problem. □

THEOREM 13. *Every constraint-satisfaction problem is polynomially equivalent to a balanced digraph-homomorphism problem, but now for a balanced digraph with only five levels (but not just four levels, which is polynomially solvable).*

Proof. The digraph-homomorphism problem can be encoded as a balanced digraph-homomorphism problem with only five levels. Given an arbitrary digraph without self-loops, represent all vertices as vertices at level 1, and all edges as vertices at level 5. If vertex v has outgoing edge e, join their representations by an oriented path 111011. If vertex v has incoming edge e, join their representations by an oriented path 110111. If neither relation holds, join their representations by an oriented path 11011011. The key properties are that neither of the first two paths map to each other, and that a digraph maps to the third path if and only if it maps to the first two. Now given a digraph, we can decompose it into connected components by removing the vertices at levels 1 and 5. Each such component either maps to none of the three paths (in which case no homomorphism exists), or to all three of them (in which case it imposes no restriction on where the boundary vertices at levels 1 and 5 map), or to exactly one of the first two paths (in which case it indicates an outgoing or incoming edge in the original graph). Thus every instance of the new problem can be viewed as an instance of the original problem, given the fact that mapping digraphs to paths is polynomially solvable; see sections 6.1.1 and 6.1.2. □

Another case of interest is that of *reflexive graphs*, i.e., graphs with self-loops. The homomorphism problem is not interesting here, since all vertices may be mapped to a single self-loop. We consider the reflexive graph-retract problem, as well as two other related problems. The *reflexive graph-list* problem is the homomorphism

problem where in addition we may require that some vertex maps to a chosen subset of the vertices in the template. The *reflexive graph-connected list* problem allows only subsets that induce a connected subset of the vertices in the template. The following results are from Feder and Hell [18, 19].

THEOREM 14. *Every constraint-satisfaction problem is polynomially equivalent to a reflexive graph-retract problem. The reflexive graph-retract problem is NP-complete for graphs without triangles other than trees. The reflexive graph-list problem is polynomially solvable for interval graphs, and NP-complete otherwise. The reflexive graph-connected list problem is polynomially solvable for chordal graphs, and NP-complete otherwise. The graph-retract problem for connected graphs with some self-loops is NP-complete if the vertices with self-loops induce a disconnected subgraph.*

For partial orders and strict partial orders, the homomorphism problem is easy, since the core is either a single vertex, in the case of partial orders, or a total strict order, in the case of strict partial orders. We examine the corresponding retract problem. For strict partial orders, even if the strict partial order is bipartite, the problem is equivalent to the bipartite graph-retract problem and hence to all of CSP. For partial orders, there are applications to type reconstruction; see Mitchell [41], Lincoln and Mitchell [42], Wand and O'Keefe [43]. Pratt and Tiuryn [45] showed that the bipartite partial order-retract problem is polynomially solvable if the underlying graph is a tree (in fact, in NLOGSPACE), and NP-complete otherwise. We can give an alternative proof of this result here. If the underlying graph is a tree, we have a directed reflexive graph whose underlying graph is a tree. If we then associate a Boolean variable with each subtree, the problem is just a 2SAT problem, because if a set of subtrees pairwise intersect, then they jointly intersect. For the NP-completeness result, define an undirected reflexive graph on the same set as the bipartite partial order, making x, y adjacent if there exist s, t such that $s \leq x, y \leq t$; in this case, this means that $x \leq y$ or $y \leq x$. This gives a reflexive graph-retract problem on a graph without triangles and not a tree, which is NP-complete by the preceding theorem. We now examine the general case. Let the *depth* of a partial order be the number of elements in a total suborder.

THEOREM 15. *Every constraint-satisfaction problem is polynomially equivalent to a partial order-retract problem, even if only the top and bottom elements of the partial order can occur in an instance. The equivalence holds even for depth-3 partial order-retract problems.*

Proof. We prove the equivalence to the domination-free bipartite graph-retract problem, which was shown to be equivalent to all of CSP above. We shall assume that only the top and bottom elements of the partial order can be used in an instance; to extend the result to the case where all elements can be used, we can simply consider the core of the partial order. Let $H = (S, T, E)$ be a domination-free bipartite graph. Define the corresponding partial order $P = (Q, \leq)$ as follows. Let Q be the set of all bipartite cliques $A \times B \subseteq E$, with $A, B \neq \emptyset$. Let $A \times B \leq A' \times B'$ if $A \subseteq A'$ and $B' \subseteq B$. If $N(v)$ denotes the set of neighbors of v in H, then the bottom elements are $\{a\} \times N(a)$ for $a \in S$ and the top elements are $N(b) \times \{b\}$ for $b \in T$.

Given an instance G for H, we can assume that G is bipartite since H is bipartite, and that G and H share at least a vertex, since otherwise G can be mapped to a single edge in H, so that we know which side of G maps to S and which to T. Replace adjacency in G with $\leq$ from the side mapping to S to the side mapping to T, replace any occurrence of an H vertex in G by the corresponding bottom or top element in P, and ask whether this partial order maps to P. We can assume that top and bottom

elements map to top and bottom elements, respectively, and on these elements the $\leq$ relation in P corresponds to edges in H, so solving the problem on P solves the instance G for H.

In the other direction, given an instance R for P, where R and P only share top and bottom elements of P, we can assume that such elements are also top and bottom in R, since everything below a bottom element must map to that bottom element and everything above a top element must map to that top element. We can also assume that top and bottom elements in R map to top and bottom elements in P. To map such elements, determine the bipartite $\leq$ relation on them, and map them by solving the problem as a bipartite graph-retract problem for H, with the natural correspondence between bottom and top elements of P and vertices in the S, T sets of H. Clearly, if the bipartite graph-retract problem does not have a solution, neither does the partial order-retract problem. If the bipartite graph-retract problem has a solution, it only remains to map the middle vertices. If a middle element is between bottom and top elements that were mapped to subsets $A \subseteq S$ and $B \subseteq T$, map that middle element to $A \times B$ in P, completing the retraction.

In order to prove the depth-3 result, we first determine the core of the partial order. Clearly, the core must contain every $\{a\} \times N(a)$ for $a \in S$ and every $N(b) \times \{b\}$ for $b \in T$, because these elements can be used in an instance. As a result, it must also contain all maximal bipartite cliques $A \times B$, because such maximal bipartite cliques are the only bipartite cliques above the corresponding elements $\{a\} \times N(a)$ for $a \in A$ and below the corresponding elements $N(b) \times \{b\}$ for $b \in B$. Notice that the maximal bipartite cliques are precisely those bipartite cliques $A \times B$ with $A = N(B)$ and $B = N(A)$, where $N(C)$ denotes here the vertices adjacent to all of C. We now observe that the maximal bipartite cliques are indeed the entire core, because the mapping $f(A \times B) = N(N(A)) \times N(A)$ is an appropriate retraction.

Having identified the core of the partial order as the partial order on maximal bipartite cliques, we use the fact that the domination-free bipartite graph can also be assumed to be $K_{3,3}$-free and $K_{3,3}\backslash\{e\}$-free. Now, if $A_1 \times B_1 < A_2 \times B_2 < A_3 \times B_3 < A_4 \times B_4$ for maximal bipartite cliques, then the containments on the A_i and on the B_i must be strict, so $|A_i| \geq i$ and $|B_i| \geq 5 - i$. In particular, $A_2 \times B_2$ must contain a $K_{2,3}$ and $A_3 \times B_3$ must contain a $K_{3,2}$, and furthermore their union must contain a $K_{3,3}\backslash\{e\}$ by maximality of the bipartite cliques. This establishes the depth-3 claim. Notice also that $K_{3,3}$-freedom implies that a bipartite clique $A \times B$ must have $|A| \leq 2$ or $|B| \leq 2$, and since for maximal bipartite cliques, A and B uniquely determine each other, the partial order has size polynomial in the size of the bipartite graph. □

Therefore, the dichotomy question for MMSNP is equivalent to the dichotomy question for CSP, which in turn is equivalent to the dichotomy questions for graph-retract, digraph-homomorphism problems, and partial order-retract problems.

6. Special classes. Schaefer [48] showed that there are only three polynomially solvable constraint-satisfaction problems on the set $\{0, 1\}$, namely, Horn clauses, 2SAT, and linear equations modulo 2; all constraint-satisfaction problems on $\{0, 1\}$ that do not fit into one of these three categories are NP-complete. For general constraint-satisfaction problems, we introduce two classes, namely, *bounded-width* and *subgroup*, and examine two subclasses of the bounded-width class, namely, the *width* 1 and *bounded strict width* classes. These are generalizations of Schaefer's three cases, since Horn clauses have width 1, 2SAT has strict width 2, and linear equations modulo 2 is a subgroup problem. At present, all known polynomially solvable constraint satisfaction problems are simple combinations of the bounded-width case and the subgroup case.

Remark. A similar situation of only three polynomially solvable cases was observed for Boolean network stability problems by Mayr and Subramanian [39] and Feder [15]; the three cases there are monotone networks, linear networks, and nonexpansive networks, in close correspondence with Horn clauses, linear equations modulo 2, and 2SAT, respectively; it is the generalization of the nonexpansive case to metric networks that leads to characterizations along the lines of the bounded strict width case described below.

We describe the work on network stability in the context of constraint-satisfaction in more detail here. Say that a template is *functional* if all relations have some arity $k+l$ with $k, l \geq 0$ and are described by a function $f(x_1, x_2, \ldots, x_k) = (y_1, y_2, \ldots, y_l)$, called a *gate*, where the x_i are called *inputs* and the y_i are called *outputs*. A *network stability* problem is a constraint-satisfaction problem where the template is functional, and where the input structure has the property that every element participates in exactly two relation occurrences, one as an input and one as an output. The input structure is then called a *network*. The work in [39, 15] established the following. Let the *constant* gates be $f() = e$, and the *absorption* gate be $f(x) = ()$.

THEOREM 16. *Every network stability problem over a Boolean functional template containing the constant and absorption gates is NP-complete, with the exception of the following polynomially solvable cases:*

(1) *monotone functional templates, where every output y_j of every gate is a monotone function of the inputs x_i. For the general case with AND and OR gates, the problem is P-complete and determining whether there is a solution other than the zero-most and one-most solutions is NP-complete*;

(2) *linear functional templates, where every output y_i of every gate is a linear function of the inputs x_i modulo* 2;

(3) *adjacency-preserving functional templates, where every gate f has the property that changing the value of just one of the x_i inputs can affect at most one of the y_j outputs. Here the set of solutions can be described by a 2SAT instance because the median of three solutions, obtained by taking coordinate-wise majority, is also a solution.*

The case (3) *extends to a non-Boolean domain case by assuming that the template has an associated distance function on the elements satisfying the triangle inequality, such that for every gate f, if $f(x_1, x_2, \ldots, x_k) = (y_1, y_2, \ldots, y_l)$ and $f(x'_1, x'_2, \ldots, x'_k) = (y'_1, y'_2, \ldots, y'_l)$, then $\sum d(y_j, y'_j) \leq \sum d(x_i, x'_i)$. The functional template is then called nonexpansive and the associated network is metric*; *this case is also polynomially solvable. Here the structure of the set of solutions is a strict width* 2 *problem because the solutions form a* 2*-isometric subspace, where the corresponding* 2*-mapping property is obtained with the imprint function, yielding the* 2*-Helly property (see* [15] *for the definitions of* 2*-isometric subspace and imprint function).*

It is the structure presented in this theorem and its connection to Schaefer's work that initially led to the work presented here. In this paper, we are not considering special cases, such as network stability or planar graph coloring; here the template is fixed and the instance is not constrained.

6.1. Bounded-width problems. A problem is said to have *bounded width* if its complement (i.e., the question of nonexistence of a solution) can be expressed in Datalog. More precisely, it is said to have *width* (l, k) if the corresponding Datalog program has rules with at most l variables in the head and at most k variables per rule, and is said to have *width* l if it has width (l, k) for some k. For a related notion of width, see Afrati and Cosmadakis [4].

Datalog is the language of logic programs without function symbols [50]. The following Datalog program checks that an input graph is not 2-colorable:

$$\begin{array}{lcl} \mathit{oddpath}(X,Y) & :\!- & \mathit{edge}(X,Y) \\ \mathit{oddpath}(X,Y) & :\!- & \mathit{oddpath}(X,Z), \mathit{edge}(Z,T), \mathit{edge}(T,Y) \\ \mathit{not2colorable} & :\!- & \mathit{oddpath}(X,X). \end{array}$$

In this example, *edge* is an input binary relation, *oddpath* is a binary relation computed by the program, and *not2colorable* is a zero-ary relation computed by the program. The first rule says that a single edge forms an odd path; the second rule tells that adding two edges to an odd path forms an odd path; and the third rule says that the input graph is not 2-colorable if the graph contains an odd cycle. In Datalog programs for constraint-satisfaction instances we assume that there is a distinguished predicate p of arity zero (*not2colorable* in the above example) that must be derived when no solution exists for the instance. We say that such a program *solves* the problem.

The example above shows that 2-colorability has width $(2, 3)$, since it can be solved by a Datalog program with at most two variables in rule heads and at most three variables per rule. Also, 3SAT-Horn can be shown to have width 1. It is not hard to show that bounded-width problems are in monotone SNP without inequality. Furthermore, problems of width 1 are in MMSNP.

It is easy to see that all bounded-width problems are in P, since the rules can derive only a polynomial number of facts. Thus, we ask the following question.

Which problems in CSP have bounded width?

The predicates from the instance are called EDB predicates, and the new auxiliary predicates are called IDB predicates. Given a Datalog program with EDB predicates corresponding to the constraints of a constraint-satisfaction problem defined by a template T, we assign to each predicate in the program a relation on values from T. The EDB predicates already have an assigned relation in T. For IDB predicates, we initially assign to them the empty relation, then add tuples as follows. Given a rule, involving at most k variables, we consider the assignments of values from T to these k variables such that the constraints imposed on them by the current relations assigned to predicates in the body of the rule are satisfied. For these satisfying assignments for the body, we consider the induced assignments on the at most l variables in the head, and add all these tuples to the relation associated with the head of the rule. This process of adding tuples over values from T to the relations associated with IDB predicates must eventually terminate, mapping each IDB to a relation on values from T.

The relation associated with the distinguished p at the end of this process must be the empty relation. Otherwise, we could design an instance that has a solution yet for which p can be derived, simply by viewing the derivation tree that made p nonempty as an instance, contrary to the assumption that the Datalog program only accepts instances with no solution.

Even if we know that a constraint-satisfaction problem has width (l, k), there could be many Datalog programs that express the complement of the problem. Thus, it seems that to answer the question above we need to consider all possible (l, k)-programs. Surprisingly, it suffices to focus on very specific Datalog programs.

THEOREM 17. *For every constraint-satisfaction problem P there is a canonical Datalog (l, k)-program with the following property: if any Datalog (l, k)-program solves P, then the canonical one does.*

Proof. Intuitively, the canonical program of width (l, k) infers all possible constraints on l variables at a time by considering k variables at a time. This canonical program infers constraints on the possible values for the variables in the instance, both considered l at a time and k at a time, as follows. Initially, all constraints from the instance can be viewed as constraints on variables, k at a time. Now, a constraint on k variables can be projected down to a constraint on an l-subset of these k variables. In the other direction, a constraint on an l-subset can be extended up to a constraint on the k variables. This process can be iterated, and if a constraint on variables ever becomes the empty set, we can infer that the instance has no solution. This inference process can easily be described by a Datalog (l, k)-program, and in fact the inferences carried out by this canonical program contain all inferences performed by any Datalog (l, k)-program, with the interpretation of IDBs as relations on values from T defined above. □

Consider now the following two-player game on the structures S, the instance, and T, the template. Player I selects k variables and asks Player II to assign to them values from T. Then Player I keeps l out of these k assigned variables, extends this set of l to a new set of k, and asks Player II to assign values to the new $k-l$ variables, back to the earlier situation with k assigned variables. The game proceeds from there as before. Player I wins if at some point, some of the k assigned values violate a constraint from the instance.

THEOREM 18. *The canonical (l, k)-program for a constraint-satisfaction problem accepts an instance precisely when Player I has a winning strategy in the associated (l, k)-two-player game.*

Proof. Suppose the canonical program accepts an instance (such an instance necessarily fails to have a solution). Consider the corresponding derivation tree. Each node of the tree corresponds to a relation on at most k elements from S. This relation has an associated set of tuples from T as defined above. Player I traverses a path from the root to a leaf; at each step he holds a tuple that is not in the associated set. He starts at the root, where the relation has arity zero; there he holds the arity-zero tuple, which does not belong to the associated empty set. In general, at a given node v where Player I holds a tuple of arity at most l not in the associated set, Player I selects the rule corresponding to the node v and its children, and asks Player II to assign values to the remaining variables in the rule, up to a total of at most k. It cannot be that all the resulting assignments to at most l variables corresponding to the children of the node v are tuples in the sets associated with them, because then the original tuple would have been in the set associated with v. So Player I can select some child of v such that the assignment to its at most l variables is not in the associated set. When a leaf is reached, Player I holds an assignment violating a given constraint in the instance.

For the converse, suppose that Player I has a winning strategy. The playing of the game depending on the moves by Player II can then be viewed as a tree. For instance, at the root, after Player I has made its initial choice k elements, the children correspond to the possible choices of l elements out of these k that can be made by Player I, depending on the assignment of values to the k elements by Player II. At the next level, the extension of the l assigned values to a tuple of k variables chosen by Player I is again considered, until at the leafs we have assignments to at most l variables that violate a constraint. This tree is then precisely a derivation tree by which the canonical program can accept the instance. □

For related games, see Afrati, Cosmadakis, and Yannakakis [5], Kolaitis and Vardi

[33], and Lakshmanan and Mendelzon [36].

This notion of bounded width for constraint-satisfaction problems can also be extended to allow infinite Datalog programs (allowing infinitely many IDBs, infinitely many rules, and infinitely many conjuncts per rule); such programs have been studied before under the name L^ω. For constraint-satisfaction problems on a *finite* domain, infinite programs are no more powerful than finite programs; the reason here is that the canonical program has IDBs corresponding to constraint sets, but there are only finitely many possible constraint sets.

Remark. For constraint-satisfaction problems, it can be shown that Datalog is equivalent to Datalog($\neq, \neg$), finite or infinite. This equivalence holds more generally for problems closed under homomorphisms, finite and infinite cases being separate. For such problems, it turns out that monadic SNP with inequality is no more powerful than monotone monadic SNP without inequality, and the same holds for (binary) SNP with inequality compared with monotone (binary) SNP without inequality [16].

6.1.1. Width 1 and tree duality. Horn clauses have width $(1, k)$, where k is the maximum number of variables per Horn clause. To see this, express Horn clauses as implications with a conjunction of positive literals in the antecedent and at most one positive literal in the consequent. An instance has no solution if it implicitly contains a clause with an empty antecedent, which stands for "true" or 1, and an empty consequent, which stands for "false" or 0. This situation can be detected by Player I by selecting an appropriate clause with an empty consequent. Then Player II must assign value 0 to some variable in the antecedent, then Player I selects some appropriate clause with this variable as the consequent, and so on, until a clause with an empty antecedent is reached; then Player I wins.

Using Theorems 17 and 7, we can prove the following theorem.

THEOREM 19. *The question of whether a constraint-satisfaction problem has width* $(1, k)$ *or width* 1 *is decidable.*

Proof. The canonical program describes in that case a monotone monadic SNP problem, and we have seen that containment for such problems is decidable. In fact, we have also seen that it is never necessary to look at conditions for a monotone monadic SNP problem that do not define a biconnected component contained in biconnected components of the statement of the problem. However, these are only single relations in the case of constraint-satisfaction problems, so we can assume that the Datalog program looks only at single relations from the input. Thus k need not be larger than the largest arity, and hence width 1 is decidable. There is another way to see this. By Theorem 5, we may assume that an instance recognized by the monadic Datalog program has high girth—larger than the size of any rule. We may then assume that the body of the rule is a tree, and then the rule can be replaced by rules having only one EDB per rule, by introducing additional monadic IDB relations. So k is bounded by the largest arity. □

Let S be a connected structure. An element x is an *articulation element* if the structure S can be decomposed into two nonempty substructures that share only x. If we decompose a structure into substructures by identifying all its articulation elements, we say that the resulting substructures without articulation elements are *biconnected components.* A *tree* is a structure whose biconnected components consist of a single relation occurrence each. Following and generalizing the terminology of Hell, Nešetřil, and Zhu [25], we say that a constraint-satisfaction problem defined by a template T has *tree duality* if a structure S can be mapped to T if and only if every tree that can be mapped to S can be mapped to T.

THEOREM 20. *A constraint-satisfaction problem has tree duality if and only if it has width* 1.

Proof. Trees are precisely the objects generated by derivation trees of Datalog programs with at most one variable in the head and at most one EDB per rule. We observed in the proof of the previous theorem that if a problem has width 1, then it has a Datalog program of this form. Then an instance S does not map to T if and only if it is accepted by the Datalog program; i.e., some tree maps to S but does not map to T. Thus width 1 implies tree duality. Conversely, if tree duality holds, then width 1 follows by considering the Datalog program that generates all the trees that do not map to T; this is an infinite program, but we have observed before that infinite programs can be transformed into finite ones for constraint-satisfaction problems, e.g., the canonical program from Theorem 17. □

In fact, as observed in the proof of Theorem 7, width 1 can be decided as follows.

Tree duality decision procedure: Given a constraint-satisfaction problem with template T, let U be the structure defined as follows. The elements of U are the nonempty subsets A of the elements of T. For a relation R of arity k, impose $R(A_1, A_2, \ldots, A_k)$, the A_i not necessarily distinct, if for every $1 \leq i \leq k$ and every a_i in A_i there exist elements a_j in the remaining A_j such that $R(a_1, a_2, \ldots, a_k)$ is in T. Then tree duality holds if and only if U maps homomorphically to T.

For example, 2SAT will in particular enforce $x \vee y$ and $\overline{x} \vee \overline{y}$ on $x = y = \{0, 1\}$, hence no solution exists, showing that 2SAT does not have width 1. In general, the question of whether a constraint-satisfaction problem has bounded width (or width l, width (l, k), beyond the case $l = 1$) is not known to be decidable.

We give here an alternative proof of the correctness of the above decision procedure based on tree duality and its equivalence to the existence of a Datalog program for the problem with one EDB relation per rule and at most one variable in the head of each rule, which infers constraints on the possible values for elements of the structure.

THEOREM 21. *The above decision procedure correctly decides tree duality.*

Proof. Suppose that tree duality holds. Consider the structure U defined in the decision procedure whose elements are nonempty sets A. We show that U can be mapped to T. Every tree that maps to U has elements that are nonempty sets A. The tree can be mapped to T by choosing one element from the root of the tree, one consistent element from each of its children, and so on. Thus, by the definition of tree duality, U maps to T. Conversely, suppose that U maps to T, and let S be a structure such that every tree that maps to S maps to T. If we use the Datalog program on S, then every element of S will be assigned a nonempty set A by the program, otherwise the derivation tree would provide a tree that maps to S but not to T. This gives a mapping from S to U, and by composition from S to T. Therefore tree duality holds. □

Thus tree duality has a simple decision procedure. Consider the special case where the template T is an oriented path (a path each of whose edges may be oriented in either direction). This T was shown to have tree duality by Hell and Zhu [28]; in fact, they showed that it satisfies the stronger path duality property that S maps to T if and only if every oriented path that maps to S maps to T. We give a simple proof of tree duality via the above decision procedure. Suppose that the elements of the path T are numbered $1, 2, \ldots, r$ in order. For every nonempty subset A of $\{1, \ldots, r\}$, map A to the least numbered element of A. If there is an edge from A to B, then the least elements of A and B cannot be the same element a, since otherwise T would contain an edge from a to $a + 1$ and one from $a + 1$ to a. So either a is the least element of

A, $a+1$ is the least element of B, and T has an edge from a to $a+1$, or a is the least element of B, $a+1$ is the least element of A, and T has an edge from $a+1$ to a. Thus T has an edge from the image of A to the image of B, as required.

A more general case is the case of oriented trees, which defines both polynomial and NP-complete problems, as well as problems that have not yet been classified [26].

We say that a constraint-satisfaction problem defined by a template T with T a core has *extended tree duality* if a connected structure S with one element s preassigned a value t in T can be mapped to T, if and only if every tree that can be mapped to S can be mapped to T in such a way that the elements of the tree that map to s end up mapping to t. Just like tree duality could be decided by the existence of a mapping from a particular structure U to T, extended tree duality can be decided by the existence of a mapping from a particular structure U' to T, where U' is the substructure of U consisting of the union of the connected components containing the singletons $\{t\}$ for t in T. The proof of correctness is similar to the tree duality case. Since the extended tree duality property involves one special element s, and problems with tree duality have width 1, problems with extended tree duality have width 2. (The converse is false, e.g., 2SAT.)

THEOREM 22. *For a constraint-satisfaction problem with template T, tree duality is equivalent to the existence of a homomorphism from U to T. If T is a core, then extended tree duality is equivalent to the existence of a homomorphism from the substructure U' to T. Problems with extended tree duality have width* 2.

Say that two elements are *related* if they both belong to the same set A in the connected component U'. A special case of extended tree duality for digraphs is the case where there is a total ordering of the vertices of T such that if (a,b) and (c,d) are two edges of T with $a<b$ and $c<d$, with a,c related and b,d related, then (a,d) is also an edge of T. In this case we can map a nonempty set A in U' to the least element of A under the ordering. This case is essentially the same as the extended $\underline{\text{X}}$-property from [23, 25].

More generally, an (l,k)-*tree* is a structure given by a derivation tree of a Datalog program whose rules have at most l variables in the head and at most k variables per rule. Combinatorially, a structure S is an (l,k)-tree if there exists a tree t whose nodes are sets of elements of S of cardinality at most k, where a node and a child in t share at most l elements of S, the nodes in which an element of S participates form a subtree of t, and each relation occurrence in S involves elements contained in a single node of t. In the literature, when S is a graph and S is an (l,k)-tree, then it is said to have *tree-width* $k-1$ (see, e.g., [38, 47]).

Along the general lines of duality of graph homomorphisms (see Hell, Nešetřil, and Zhu [27]), a constraint-satisfaction problem defined by a template T has (l,k)-*tree duality* if a structure S can be mapped to T if and only if every (l,k)-tree that can be mapped to S can be mapped to T. The following is immediate from the definition of acceptance by a Datalog program.

THEOREM 23. *A constraint-satisfaction problem has (l,k)-tree duality if and only if it has width (l,k).*

Recall from section 3 the definition of the length of an oriented cycle and of balanced and unbalanced digraphs. Hell and Zhu [29] show that in the case where T is an unbalanced cycle, it is sufficient to test oriented paths, which are $(1,2)$-trees, and oriented cycles, which are $(2,3)$-trees; therefore unbalanced cycles define a problem of width $(2,3)$. The property that cycles of the instance S must satisfy is that their length is a multiple of the length of T. Therefore, in the special case where the length

is 1, cycles need not be tested, only oriented paths, and the problem has width $(1, 2)$. We show here that unbalanced cycles have extended tree duality, hence width 2, and that when their length is 1, they have tree duality, hence width 1. Suppose that a cycle has length $m \geq 1$. Enumerate the vertices $0, 1, \ldots, k$ in order around the cycle, where vertices 0 and k coincide, in such a way that there is a *level* function such that every edge (i, j) satisfies $level(j) = level(i) + 1$, with $level(0) = m$, $level(k) = 0$, and $level(i) \geq 1$ for $i \neq k$. Then define the total ordering mentioned above for extended tree duality to be the lexicographical ordering on pairs $(level(i), i)$. Notice that if two elements are related, their levels must differ by a multiple of m. In the case where $m = 1$, this imposes no restriction; i.e., we may use the structure U instead of U'. The same construction can be used to establish tree duality for balanced cycles such that if we denote by l an occurrence of a lowest level element and by h an occurrence of a highest level element, then the ordering $lhlh$ does not occur when the cycle is traversed once, so that the complete ordering of l and h is l^+h^+.

In general, for an arbitrary constraint-satisfaction problem, if we only consider instances that are (l, k)-trees with l, k fixed, then the problem is solved in polynomial time by the canonical Datalog (l, k)-program, since this program has as derivation trees precisely the (l, k)-trees that do not have a solution for the constraint-satisfaction problem.

THEOREM 24. *On (l, k)-tree instances, constraint-satisfaction problems are polynomially solvable, for l, k fixed.*

For more general results on polynomially solvable problems in the case of graphs of bounded tree-width, see, e.g., [1, 9].

6.1.2. Bounded strict width and the Helly property. The canonical algorithm for problems of width (l, k) involved inferring all possible constraints on l variables at a time by considering k variables at a time. We may in addition require that if this inference process does not reach a contradiction (the empty set), then it should be possible to obtain a solution by greedily assigning values to the variables one at a time while satisfying the inferred l-constraints. We say that a constraint-satisfaction problem that can be solved in this way has *strict width* (l, k), and we say that it has *strict width* l if it has strict width (l, k) for some k. It turns out that strict width l is equivalent to strict width (l, k), for all $k > l$, so we can assume $k = l + 1$.

This intuition behind strict width l can also be captured in two other ways. First, we can require that if we have an instance and that, after assigning specific values to some of the variables, we obtain an instance with no solution, then some l out of the specific value assignments chosen are sufficient to give an instance with no solution. We refer to this property as the l-Helly property. Second, we could require that there exists a function g that maps $l + 1$ elements from the domain of the structure T to another element, with the property that if all but at most one of the $l + 1$ arguments are equal to some value b then the value of g is also b, and, furthermore, for all constraints C_i in T, if we have $l + 1$ tuples satisfying C_i, then the tuple obtained by applying g componentwise also satisfies C_i. We call this property the l-mapping property.

THEOREM 25. *Strict width l, the l-Helly property, and the l-mapping property are equivalent. These properties are polynomially decidable (for a fixed l).*

Proof. The proof of the equivalence of the various formulations of bounded strict width shows first that the correctness of the greedy (l, k)-algorithm implies that the Helly property for l must hold. The reason is that the final step of finding a solution uses only the inferred constraints on l variables at a time, hence these constraints

must precisely characterize the solutions, showing that the Helly property for l holds. This in turn implies that the instance stating the existence of g must have a solution. The reason is that l out of the constant value assignments imposed on g can always be satisfied, because each of these constant value assignments has l out of the $l+1$ arguments equal for g; there must be an argument position that is not the one exceptional argument for any of the l constant value assignments being considered, so we can always return the value of this argument, satisfying all conditions. Since the constant value assignments considered l at a time are satisfiable, then altogether they must be satisfiable by the Helly property for l, and hence g exists.

Finally, the existence of g implies the correctness of the greedy $(l, l+1)$-algorithm; it implies in fact the correctness of a more restrictive algorithm that eliminates variables one by one in arbitrary order (by considering just the $l+1$-subsets containing a chosen variable to infer a constraint on the remaining l variables) and assigns values in reverse order. Consider the elimination of the first variable x_1. We claim that any solution x' of the resulting instance on the remaining variables $x_2, \ldots, x_n$ can be extended to a solution for x_1. We must show that there exists a value for x_1 satisfying all constraints involving x_1 in conjunction with x'. We first consider constraints involving only l of the variables in x'. If no value of x_1 satisfies the constraints involving x_1 and the chosen l variables, then this would result in the inference step in forbidding the value assignment on l variables induced by x', and x' would not have been a solution of the resulting instance. Therefore constraints involving only x_1 and l of the variables in x' can be satisfied (we are including here in the inference step constraints obtained as projections of constraints involving x_1 and variables possibly different from the chosen l). Consider now constraints involving only x_1 and k of the variables in x', for $k \geq l+1$, where we assume inductively that $k-1$ can be handled. Consider $l+1$ particular variables out of the chosen k from x'. For each choice of one variable (say, the ith one) out of these $l+1$, if we ignore it, then a value x_1^i for x_1 can be found by inductive assumption. But then, we can set $x_1 = g(x_1^1, x_1^2, \ldots, x_1^{l+1})$ and satisfy the constraints on x_1 and all k chosen variables. The reason is that any such constraint involves l variables including x_1. One of the $l-1$ variables other than x_1 may have been ignored in choosing x_1^i, but a value for it can be found since otherwise x_1^i would not have been considered consistent. Furthermore, a variable was ignored in choosing at most one x_1^i, so g applied to the $l+1$ values for this variable gives the correct majority value. Since the constraint is closed under g, the value x_1 obtained by applying g satisfies all the constraints, completing the induction. To handle the elimination of subsequent variables analogously, it is only important to observe that the constraints obtained by the inference step are also closed under g, since they are obtained by intersection and projection of constraints closed under g. This proves the correctness of the algorithm.

Note that the existence of g is itself an instance of the constraint-satisfaction problem, hence strict width l is decidable and in fact polynomially decidable, for fixed l. It is not known to be decidable when l is not fixed. □

2-colorability is an example of a constraint-satisfaction problem with strict width 2. If a graph is bipartite, but after having 2-colored some of the vertices there is no two-coloring consistent with this partial coloring, then either two vertices on different sides of the bipartite graph were given the same color or two vertices in the same side were given different colors. Also, 2SAT has strict width 2, and so does integer programming with two variables per inequality with variables ranging over a fixed range.

Feder [17] showed that digraph-homomorphism for oriented cycles is either polynomially solvable or NP-complete. The proof is a good illustration of some of the techniques we have been using up to this point; we give here a sketch of the proof.

THEOREM 26. *Every oriented cycle digraph-homomorphism problem is either polynomially solvable or NP-complete.*

Proof (sketch). The case where a template is an oriented path, which we saw in the previous section has width 1, also has strict width 2. To see this, number again the vertices $1, 2, \ldots, r$ and consider the mapping $g(x, y, z) = \text{median}(x, y, z)$. In the case where the template is a directed graph that maps to a cycle C, it is sufficient to consider only argument lists for g such that all arguments map to the same vertex in C. For unbalanced oriented cycles, which we saw have extended tree duality, we also have strict width 2; the argument considers the lexicographic ordering on pairs $(\mathit{level}(i), i)$ as before and uses the median(x, y, z) function on three arguments whose levels differ by a multiple of the length of the cycle. The same argument applies to the balanced cycles not containing the ordering $lhlh$ as mentioned before, so that the complete pattern of l and h is l^+h^+. There is a family of balanced cycles that does not have extended tree duality (it can encode 2SAT) yet has strict width 2: these are the balanced cycles with two l and two h elements that form the pattern $lhlh$ along the cycle. Consider the four paths $1 = l_1h_1$, $2 = l_2h_1$, $3 = l_2h_2$, $4 = l_1h_2$ on the cycle; given three paths $i - 1$, i, $i + 1$ out of these four (modulo 4), the *middle path* is i. The mapping $g(x, y, z)$, with the three arguments at the same level, is defined as follows. (1) If x, y, z belong to three different paths, return the one that belongs to the middle path. (2) If x, y, z belong to the same path, return the one in the middle position on the path. (3) If exactly two out of x, y, z belong to the same path, return the one of these two occurring earliest on the path. In each of the classes of the form $l^+h^+l^+h^+$, other than the polynomial class $lhlh$, there are both polynomial and NP-complete templates that are cores; the cases $(l^+h^+)^{\geq 3}$ are NP-complete for cores. It is shown in [17] that in the remaining case $l^+h^+l^+h^+$, all problems are either in P or NP-complete, completing the classification of oriented cycles. The proof uses the 2-Helly property for paths, which have strict width 2, and a generalization of Schaefer's classification of Boolean satisfiability to a k-partite version [17, 48]. □

Problems with bounded strict width are a special case of bounded-width CSP. For such problems we have a more efficient algorithm.

THEOREM 27. *A simplified version of the canonical algorithm runs in parallel $O^*(n)$ time using a polynomial number of processors (the O^* notation ignores polylogarithmic factors). In the case $l = 2$, this algorithm can be implemented in parallel $O^*(\sqrt{n})$ time, because variables can be eliminated in parallel.*

Proof. First, observe that the number of constraints that could be inferred is $O(n^2)$. Therefore the number of steps that infer at least $n^{1.5}$ constraints is only $O(\sqrt{n})$. Suppose that a step would infer only $n^{1.5}$ constraints, i.e., only these many pairs of values for pairs of variables are discarded. Suppose that we "charge" such a pair of values eliminated for two variables x_j, x_k to a variable x_i such that the inference on these three variables causes the pair of values to be eliminated. Since there are n variables, they are charged $n^{0.5}$ eliminations each in average, and thus at least half of them are charged at most $2n^{0.5}$ eliminations, involving at most $4n^{0.5}$ other variables. But then, it must be possible to choose a set of $n^{0.5}/8$ variables that are charged only $2n^{0.5}$ eliminations, each involving two other variables not in the set. But then these $n^{0.5}/8$ variables can be eliminated simultaneously, and hence this type of step need only be performed only $O(\sqrt{n})$ times as well. All inferences will thus be

obtained by the standard algorithm in $O(\sqrt{n})$ steps (consider eliminating a chosen pair last). To obtain a solution, either a value assignment for one variable restricts the values for $n^{0.5}$ others or there are $n^{0.5}$ pairwise unconstrained variables; alternatively, a maximal independent set computation works. □

6.2. Problems with the ability to count. Which problems in CSP do not have bounded width? Say that a constraint-satisfaction problem has *the ability to count* if the following two conditions hold: (1) The template T contains at least the values $0, 1$, as well as a ternary relation C that includes at least the triples $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$, as well as a monadic relation Z that includes at least 0; (2) If an instance consists of only the constraints C and Z, with all constraints partitioned into two sets A and B such that A contains one more C constraint than B, and furthermore each variable appears in exactly two constraints, one from A and one from B, then the instance has no solution.

Intuitively, we can think of $C(x, y, z)$ as $x + y + z = 1$, and of $Z(x)$ as $x = 0$, with an obvious contradiction for instances of the special form, since adding the constraints from A and subtracting those from B yields $0 = 1$. Some problems of this form include linear equations over an abelian group (finite or infinite), where 0 is the identity of the group and 1 is any other element. In particular, this includes the problem for linear equations modulo 2. Another example of such a problem is linear programs over the nonnegative reals. For this last case, inexpressibility in Datalog($\neq$, *succ*) was shown by Afrati, Cosmadakis, and Yannakakis [5] using Razborov's monotone circuit lower bound for matching.

We shall first show that if a constraint-satisfaction problem has the ability to count, then it does not have polynomial size monotone circuits. We begin by citing Razborov's lower bound for matching [46]. In fact, Razborov's lower bound is not just for matching; it applies to any monotone problem such that certain particular instances are "yes" instances, certain other instances are "no" instances, and the remaining instances may be either "yes" or "no." Matching is just a specific application of the result. The exact statement of Razborov's result is the following.

THEOREM 28. *Consider a monotone problem on bipartite graphs such that* (1) *if the instance has a perfect matching, then the answer is "yes," and* (2) *if the instance contains a bipartite connected component with a different number of vertices in the two sides, then the answer is "no." Then monotone circuits for the problem have size* $m^{\Omega(\log m)}$.

The lower bound for matching is thus obtained by requiring a "yes" answer for the instances (1) and a "no" answer for all remaining instances, not just instances (2). The other extreme case is also interesting, namely, the problem that requires a "no" answer for the instances (2) and a "yes" answer for all remaining instances, not just instances (1). This gives a lower bound for systems of linear equations over the integers, the rationals, or the reals, as follows. View a complete bipartite graph with n vertices on each side as an instance with n^2 variables corresponding to the n^2 edges. View each vertex as stating the constraint that the sum of the variables corresponding to edges incident on the vertex is equal to 1. If the bipartite graph is not complete, then each missing edge is viewed as a constraint stating that the corresponding variable is equal to 0. Now if an instance contains a bipartite connected component with k vertices in the left side and k' vertices in the right side, and with $k \neq k'$, then the kk' variables corresponding to the possible edges joining the two sides must add to k according to the left side, and to k' according to the right side, so an instance (2) is indeed a "no" instance. On the other hand, if all connected components have $k = k'$, then we can

pair up the k and k' vertices in the two sides. For each such pair (u, v), there is an odd length path from u to v, so we can assign to the edges on the path the values 1 and -1 in alternation, so that the sum is 1 for edges incident on u, and on v, but 0 for edges incident on all other vertices; doing this for all chosen pairs (u, v) and adding up the values corresponding to the different paths gives a sum 1 for each of the k and k' vertices. So the answer is "yes" for every instance other than (2).

This gives the basic idea for getting a monotone circuit size lower bound on problems that have the ability to count, since both problems just considered have the ability to count. However, the theorem cannot be applied directly for other problems that have the ability to count, such as linear equations modulo 2. Nevertheless, a result just slightly stronger than Razborov's will yield what we need; we just weaken (2) a little bit.

THEOREM 29. *Consider a monotone problem on bipartite graphs such that* (1) *if the instance has a perfect matching, then the answer is "yes," and* (2) *if the instance contains a subgraph not connected to the rest of the graph with one more vertex on the left than on the right, then the answer is "no." Then monotone circuits for the problem have size* $m^{\Omega(\log m)}$.

Proof. Razborov's proof involves a choice of a random bipartite E_- on two vertex sets A and B with m elements each by selecting random subsets $A' \subseteq A$ and $B' \subseteq B$ and linking all vertices in A' to all vertices in B', as well as all vertices in $\overline{A'}$ to all vertices in $\overline{B'}$. The random E_- is used to bound probabilities for three events, namely (stating equations directly from [46]) $P(E_- \in [E]) \geq 2^{-s}$ (30), $P(E_- \in A(f_m)) \leq m^{-1/2}$ (31), and $P(E_- \in S) \leq h(t, r, s, m)$ (32). We replace E_- by an event E' chosen from a smaller space, conditioning on $|A'| = |B'| + 1$ as well as $|(|A'|) - (|\overline{A'}|)| > s$ (the latter for convenience only). Formula (30) measures the probability that a fixed matching of size s will be contained in E_-. To measure this for E', we can first choose $|A'|$ from the appropriate distribution, then assume that A' and B' are fixed while the s edges of the matching are chosen at random. From the assumptions on $|A'|$ and $|B'|$, it is clear that the first of these edges will fall in E' with probability at least $1/2$, and that if this has happened for the first $i-1$ edges, then it will happen for the ith edge with probability at least $1/2$ as well, as long as $i \leq s$. Hence (30) can only become stronger for E'. Formula (31) measures the probability that E_- has a perfect matching, but this probability is zero for E'. Only (32) becomes weaker; an E_- will satisfy the conditioning with probability $\Omega(1/\sqrt{m})$, so for E' the bound increases by a factor of $\sqrt{m}$; this factor carries over to the final $m^{\Omega(\log m)}$ lower bound on monotone circuit size (where it is insignificant). □

This result can now be applied to linear equations modulo q, with $q \geq 2$. Relate linear equations to bipartite graphs as before. If the bipartite graph has a perfect matching, then giving value 1 to the edges in the matching and value 0 to all other edges satisfies the linear equations. If the bipartite graph has a subgraph not connected to the rest of the graph with k vertices on the left and k' on the right, and $k = k' + 1$, then the kk' variables involved must add up to k and to k', yet now $k \neq k'$ holds even modulo q. We are now ready to apply the theorem to any problem with the ability to count.

THEOREM 30. *If a constraint-satisfaction problem has the ability to count, then monotone circuits for it have size* $m^{\Omega(\log m)}$.

Proof. We first represent the complete bipartite graph with n vertices on each side by a bipartite graph with vertices of degree 2 or 3, as follows. Replace every vertex of degree n with a path on $2n-1$ vertices $0, 1, 2, \ldots, 2n-2$, with the n incident edges

attached to the n vertices in even positions, thus achieving the degree constraint. Notice that in a perfect matching, the $n-1$ vertices in odd positions must be matched to $n-1$ out of the n vertices in even positions, leaving exactly 1 vertex in the even position left to be matched to a vertex not on that path. Thus perfect matchings for subgraphs of the complete bipartite graph and perfect matchings for subgraphs of the new graph obtained by removing edges other than those on the paths are in 1-to-1 correspondence. Similarly, subgraphs of the complete bipartite graph with one more vertex in the left side correspond to subgraphs of the new graph obtained by removing edges other than those on the paths, also with one more vertex in the left.

So we have reduced the previous theorem to the case of graphs that are subgraphs of a bipartite graph G with vertices of degree at most 3. Now view each edge as a variable, a vertex with three incident edges x, y, z as a constraint $C(x, y, z)$, and a vertex with two incident edges x, y as a constraint $C(x, y, z)$, where z is an auxiliary variable that is also constrained by $Z(z)$. Removing an edge x not on one of the paths corresponds to adding a constraint $Z(x)$. Clearly, if the graph has a perfect matching, then setting the variables in the matching to 1 and all other variables to 0 gives a solution for the problem with the ability to count. Similarly, if the graph has a subgraph with one more vertex in the left, this gives an instance of the problem with the ability to count that was required not to have a solution. The only technical point here is that removing an edge x incident to two vertices corresponds to having $C(x, y, z)$, $C(x, y', z')$, and $Z(x)$, so x participates in three constraints instead of just two as was required for problems with the ability to count with no solution. However, we can replace x with two variables x, x' constrained by $C(x, y, z)$, $C(x', y', z')$, $Z(x)$, and $Z(x')$, use the fact that the resulting instance has no solution by the definition of the ability to count, and then observe that identifying x and x' cannot make the constraint-satisfaction problem have a solution if it had no solution before identifying x and x'. □

Remark. The idea of obtaining monotone circuit lower bounds for problems that are strictly between the required "yes" and "no" instances was used previously by Tardos [49], who obtained a truly exponential monotone circuit lower bound for the Lovasz θ function, a polynomially computable function strictly between the NP-complete functions maximum clique and chromatic number.

Afrati, Cosmadakis, and Yannakakis [5] showed that if a problem does not have polynomial size monotone circuits, then it is not expressible in Datalog($\neq$, *succ*). Thus problems with the ability to count do not have bounded width. If we just want to show that a problem does not have bounded width, then it suffices to show that it does not have a Datalog (l, k)-program for any fixed l, k. This can be proved more easily via two-player games, by an argument similar again to an argument used by Afrati, Cosmadakis, and Yannakakis [5] for linear programs related to matching.

THEOREM 31. *If a constraint-satisfaction problem has the ability to count, then it does not have bounded width.*

Proof. An instance of the special form with no solution, as in the definition of the ability to count, can be viewed as a bipartite graph with no perfect matching, where the C constraints in A and B are viewed as two vertex sets, and the variables without a Z constraint imposed on them are viewed as edges joining the two vertices in A and B representing the constraints where they occur. Clearly, no perfect matching exists, since there is one more C constraint in A than in B. However, we can construct such an instance of size n such that if two players play the game with l, k about $\sqrt{n}$, where Player I selects edges and Player II indicates whether they are in the matching or

not, then Player II can ensure that if the two or three edges incident on a vertex are ever selected together, then exactly one of them is claimed to be in the matching, corresponding to satisfaction of C. First ignore the degree constraint and consider a complete $k+1$ by k bipartite graph. Then, intuitively, as long as fewer than k edges are currently in the matching, an unmatched vertex can always be matched by Player II. To bound the degree, we replace each vertex of degree d in this graph by a path on $2d-1$ vertices, with the d incident edges attached to alternating vertices in the graph; hence all vertices now have degree 2 or 3. (This last transformation is similar to the one given in the proof of the previous theorem.) Here $|A|$, $|B|$ and the number of variables (edges) are quadratic in k.

Hence rules with about $\sqrt{n}$ variables per rule will be needed to recognize certain instances with no solution on n variables. In the case of abelian groups, we can improve this lower bound to about n. The basic idea is that the complete bipartite graph considered above can be viewed as describing constraints $\sum_j x_{ij} = 1$ for all $1 \leq i \leq k+1$ and $\sum_i x_{ij} = 1$ for all $1 \leq j \leq k$, giving an obvious contradiction $\sum_i \sum_j x_{ij} = 1 + \sum_j \sum_i x_{ij}$. This equation is of the form $\sum_i y_i = 1 + \sum_i y_i$, where the y_i are added in different order on the left- and right-hand sides. If we consider the graph consisting of a path joining the y_i in the order from the left-hand side and another path joining the y_i in the order from the right-hand side, then in the case where the order of summation was exchanged, we essentially have a grid. For a square grid, a bipartition of the vertices into two sets of about equal size must have about $\sqrt{n}$ edges across the bipartition. On the other hand, an expander increases this quantity to about n, e.g., in the case where the order of y_i in the two sides is chosen independently at random. The game played by the two players consists of selecting vertices (variables y_i) and edges (variables representing partial sums); the removal of these may disconnect the graph, and Player II can always ensure that values are assigned so that the connected components with fewer than some constant fraction of the vertices have a solution consistent with the boundary constraints. A contradiction can only be reached by growing such components, but then the fact that the graph is an expander forces the number k of edges out of a sufficiently large component to be about n. We don't know whether such a bound can be obtained in the more general case via matchings. □

Thus, the polynomial solvability of linear equations modulo 2 cannot be explained in terms of Datalog.

It seems possible that the ability to count is the converse of having bounded width. The intuition for this is that the nonexistence of solutions can be attributed to the presence of the same variable in different constraints, something that cannot be remembered by Datalog if these occurrences in two different places are not ordered, and it seems that to keep track of equality of variables in different order one needs the ability to count.

Say that a core T *can simulate* a core T' if for every relation C_i in T' there is an instance S_i that defines on some variables in S_i a relation C_i' over the domain of T whose core is precisely C_i. (Here we can bound $|S_i| \leq |T|^{|C_i|}$.)

We are then saying that it may be that a constraint-satisfaction problem is not of bounded width if and only if it can simulate a problem that has the ability to count.

6.3. Subgroup problems. In this section we study subgroup problems, whose template has as elements the elements of a finite group G, and a relation of arity k in the template must be a subgroup or coset in the direct product G^k. We first show that if a problem gets the ability to count with the two relations $x = 0$ and

$x + y + z = 1$ modulo p, then it can simulate the general subgroup problem for $G = Z_p$. Furthermore, if a set that is not a subgroup or a coset in G^k is added to the general subgroup problem for an abelian G, the problem becomes NP-complete. The natural way to obtain more general problems is to allow nonabelian groups G. Here the subgroup problem with subgroups and cosets in G^k is polynomially solvable. Here again, if a subset is allowed whose projection into some abelian section of G^k is not a subgroup or a coset, the problem can simulate one-in-three SAT and is NP-complete. We call a subset containing the identity 1 whose projection into abelian sections forms subgroups a nearsubgroup. So non-nearsubgroups added to the general subgroup problem for G give NP-completeness. On the other hand, by a result of Aschbacher [3], the intersection of nearsubgroups is a nearsubgroup, so nearsubgroups alone cannot simulate one-in-three SAT and are thus unlikely to give NP-completeness. In fact, we identify a 2-element property, which Aschbacher shows holds for solvable groups, which implies polynomiality for nearsubgroups added to the subgroup problem. We finally identify a weaker nearsubgroup intersection property that also implies polynomiality for nearsubgroups added to the subgroup problem, and here Aschbacher has found no finite group counterexample.

The simplest example of a problem that has the ability to count is the constraint-satisfaction problem whose template is the integers modulo p for some prime p, with two of the relations in the template given by $x = 0$ and $x + y + z = 1$, modulo p. As far as we know, any problem with a finite template that has the ability to count can simulate this problem for some prime p. We wish to study the interaction between these relations on Z_p and other relations in the template. We begin with a simple observation.

THEOREM 32. *Suppose that a problem with template Z_p for p prime contains at least the two relations $x = 0$ and $x + y + z = 1$ modulo p, thus getting the ability to count. Then from these two relations, every relation that is a subgroup or a coset of a subgroup of some power Z_p^k can also be obtained.*

Proof. Every subgroup or coset of a subgroup of Z_p^k is an intersection of sets defined by linear equations $\sum_{i=1}^{k} a_i x_i = b$ modulo p. So it suffices to show that every such linear equation can be obtained. To obtain such a linear equation, it suffices to obtain all equations of the form $x = a$, $y = ax$, and $x + y = z$, modulo p, since every linear equation can be defined by combining these three basic types. In fact, we only need $x = 1$ and $x + y = z$, since $x = a$ is $x = 1 + 1 + \cdots + 1$ and $y = ax$ is $y = x + x + \cdots + x$, with a terms in the sums. To get $x = 1$, just set $x + y + z = 1$, $y = 0$, $z = 0$. To get $x + y = z$, just set $x + y + t = 1$, $z + u + t = 1$, $u = 0$. □

We shall later see that if in addition to all these subgroups and cosets for Z_p and its powers, the template also has a subset of Z_p^k that is not a subgroup or coset, then the constraint-satisfaction problem for that template is NP-complete. Thus if we wish to examine constraint-satisfaction problems that have at least the ability to count modulo p but are not NP-complete, it does not make sense to add new constraints in Z_p. It may still make sense, however, to examine templates where the p elements defining Z_p are only a subset of all the elements. Furthermore, the remaining elements should interact in a natural manner with the p elements defining Z_p, to avoid defining subsets of Z_p^k that are neither subgroups nor cosets. One way of achieving this is to encode Z_p as a problem such as digraph-homomorphism; the encoding adds extra elements to the template, but only defines linear equations modulo p on some p specific elements. This approach, however, does not create a template that is essentially different from Z_p itself, just an encoding of Z_p. There seems to be only one way of obtaining a

template with more than the p elements for Z_p without interfering with the structure of Z_p: simply view Z_p as a subgroup of a larger group, not necessarily abelian, and allow subgroup and coset relations on the larger group. Here we have the following.

Let the general subgroup problem for a finite group G be the constraint-satisfaction problem with template G whose relations are subgroups and cosets of subgroups of G^k. We shall need to bound k by some constant to obtain a finite template, but we will always allow k to be as large as needed for the argument at hand.

THEOREM 33. *The general subgroup problem for a finite group G is polynomially solvable.*

Proof. The result follows immediately from a known algorithm that finds generators for a group obtained by Babai [6] and Furst, Hopcroft, and Luks [21]; see also Theorem II.12 in Hoffmann [31].

The main observation is that, given a group H with known generators and a chain of subgroups $H = H_0 > H_1 > \cdots > H_r = \{1\}$, one can obtain distinct representatives from each coset of each H_i in H_{i-1} as follows. Select two elements x, x' among the generators of H_0 that belong to the same coset of H_1, say, $x' = xy$ with $y \in H_1$; then discard x' and add y to the list of generators. Iterate until there is only one generator in each coset of each H_i in H_{i-1}, and carry out the process for products xy of two current generators as well. The fact that only products of pairs are needed to obtain representatives for all cosets of each H_i in H_{i-1} requires proof; see Theorem II.8 in [31].

In our application, we have n elements that must be assigned values in G, so a solution is an element of $H = G^n$. Each relation in an instance defines a subgroup or more generally a coset a_iJ_i in H, for some subgroup J_i of H, with $1 \le i \le s$ if there are s relations in an instance. Let $H_i = J_1 \cap J_2 \cap \cdots \cap J_i$ for $1 \le i \le s$; then fix each of the n components to 1 successively until $H_r = H_{s+n} = \{1\}$ is obtained. Now obtain representatives for all cosets of each H_i in H_{i-1} using the algorithm above.

To solve the constraint-satisfaction problem, observe that the first relation a_1J_1 is a coset of $H_1 = J_1$ in H_0, so we may select a representative a for this coset from the above representation, and then look for a solution of the form ax with a fixed and x in H_1. Having fixed a, a condition $ax \in a_iJ_i$ now becomes $x \in a^{-1}a_iJ_i = b_iJ_i$. Now we proceed with b_2J_2 and H_2 as we did before for a_1J_1 and H_1. Here it might be that no coset representative b for H_2 in H_1 is in b_2J_2, in which case the problem has no solution. If such a representative b exists, we may again look for a solution of the form by with y in H_2, and proceed as before to $H_3, H_4, \ldots, H_s$. In the end, we just need to select an element of H_s, with no constraints, and we may just take 1.

This gives a polynomial time algorithm because n, r, $|G|$, and $|H_i|/|H_{i-1}|$ are polynomially bounded. □

Labeled graph isomorphism has polynomial time algorithms obtained by the authors mentioned above, and the algorithm described in the preceding theorem is essentially the same as the algorithm of Furst, Hopcroft, and Luks [21] as described in Hoffmann [31]. In labeled graph isomorphism, two graphs have been colored with each color occurring a bounded number of times, and we look for a color-preserving isomorphism. For simplicity, assume that each color occurs in k of the vertices of each of the two graphs. Let G be the group of permutations on $2k$ elements, corresponding to the $2k$ vertices of the same color in the two graphs. The constraint that vertices in one graph map to vertices in the other is a coset in G, while the constraint that adjacent vertices map to adjacent vertices is a subgroup of G for adjacent vertices of the same color, and a subgroup of G^2 for adjacent vertices of different colors.

Thus labeled graph isomorphism can be viewed as a subgroup constraint-satisfaction problem.

Let G be a finite group and consider the general subgroup problem for G. Suppose that we consider adding a nonsubgroup constraint, where we mean a subset of G^k that is neither a subgroup nor a coset. We have stated before that for Z_p, this makes the problem NP-complete. In fact, this is still true for any abelian group G. However, it turns out that for nonabelian groups, it is sometimes possible to include a nonsubgroup constraint and still have a polynomially solvable problem. The key notion turns out to be what we call a *nearsubgroup*; nearsubgroups coincide with subgroups in the abelian case, but not generally for an arbitrary nonabelian group.

Let G be a finite group, and let K be a subset of G such that $1 \in K$. We say that K is a *nearsubgroup* if for all $b \in G$ such that $1 \in bK$, for all subgroups M of G, and for all normal subgroups N of M such that $M^* = M/N$ is abelian, the set $bK^* = \{aN \subseteq M : bK \cap aN \neq \emptyset\}$ is a subgroup of M^*. In other words, the intersections of K with the abelian sections of G form subgroups.

An alternative definition is the following. Let G be a finite group and let K be a subset of G such that $1 \in K$ and if $x, y \in K$, then $xyx \in K$. We call this condition the *cycles condition* because, given that $1 \in K$, it is equivalent to stating that if $b, bx \in K$, then $bx^2 \in K$. Furthermore, this implies that $bx^i \in K$ for all i, thus obtaining a coset $b\langle x\rangle$ of a cyclic group such that $b\langle x\rangle \subseteq K$, where $\langle x\rangle$ denotes the subgroup generated by x. Suppose that K satisfies the cycles condition. If M is a subgroup of G, and N is a normal subgroup of M with M/N isomorphic to $E_4 = Z_2^2$, then there is no $b \in G$ such that $bK \cap M$ meets exactly three of the four cosets of N in M. In this case, K is a nearsubgroup.

It will be useful to consider nearsubgroups with the following stronger 2-*element property*. Here K satisfies the cycles condition, and if S is the set of 2-elements in G, and b is such that $1 \in bK$, then $S \cap \langle S \cap bK\rangle \subseteq bK$. That is, the 2-elements in bK generate a subgroup whose 2-elements are in bK.

Our interest in these notions comes from the following three theorems. A non-nearsubgroup is a bK with $1 \in K$ such that K is not a nearsubgroup.

THEOREM 34. *Let G be a finite group. Consider the general subgroup problem for G. Also include a single subset of G^k for some k that is a non-nearsubgroup. Then the subgroup problem with this additional non-nearsubgroup constraint can simulate one-in-three SAT and is therefore NP-complete.*

Proof. Let bK be the non-nearsubgroup in G^k. We can treat k-tuples as single elements, and so assume that bK is a non-nearsubgroup in G. We can simulate the constraint $x \in K$ by $y = bx$ and $y \in bK$, because $y = bx$ is a coset $(x, y) \in (1, b)H$ where $H = \{(z, z) : z \in G\}$. So K itself is a set in the problem, with $1 \in K$, and not a nearsubgroup.

Suppose that K does not satisfy the cycles condition. If for $y \in K$, we also have $y^{-1} \in K$, then the cycles condition can be restated as $x, y \in K$ implies $xy^{-1}x \in K$. Setting $a = y$ and $az = x$, we have $a, az \in K$ but $az^2 \notin K$; we also have this if $y^{-1} \notin K$ for some $y \in K$, letting $a = y$ and $az = 1$. Furthermore, we can use the set $K' = a^{-1}K$ as a constraint by $x = ay$ and $x \in K$. So we have a constraint set K with $1, z \in K$ but $z^2 \notin K$. We pass to $\langle z\rangle$, the group generated by z, which is Z_n for some n. We wish to obtain a set with just two elements $1, z$ in $Z_k = \langle z\rangle$ for some $k|n$, $k \geq 3$.

We start with K itself and gradually reduce the size of K or the integer n down to a smaller k. If $z^k, z^{k+1} \in K$ for some $k \neq 0$, then $K' = K \cap (z^{-k}K)$ still has

$1, z \in K'$ but with K' strictly contained in K, unless $x \in K$ if and only if $xz^k \in K$, in which case we may pass to the smaller Z_k which is isomorphic to $Z_n/\langle z^k \rangle$. So we may assume that $z^k, z^{k+1} \in K$ only for $k = 0, 1$. If K contains some z^k with $k \neq 0, 1$ relatively prime to n, then we set $K' = K \cap (z^k K^{-1})$, so that $1, z^k \in K'$ but $z \notin K'$, so K' is strictly smaller than K and we may rename z^k as z. Similarly, if K contains some z^k with $k \neq 0, 1$ and $k-1$ relatively prime to n, then we set $K' = K \cap (z^{k+1} K^{-1})$. Therefore $z, z^k \in K'$ but $1 \notin K'$, so K' is strictly smaller than K and we may rename z^k as 1 by a simple transformation. Now, if K contains some z^k with $k \neq 0, 1$ with $k-1$, and k not relatively prime to n, then arguing in the smaller groups generated by z^{k-1} and z^k we may assume that K contains z^{2k} and z^{2k-1}; but this is only possible if $2k - 1 = n$, contrary to the assumption that k is not relatively prime to n.

So we may indeed assume that K contains precisely $1, z$ in $Z_n = \langle z \rangle$ with $n \geq 3$. Now consider $x, y, t \in K \cap Z_n$ with the coset constraint $xyt = z$. Then one of x, y, t is z and the other two are 1, thus defining one-in-three SAT and giving NP-completeness.

For the other case, suppose that K meets exactly three of the four cosets of N in M. We may then pass to $E_4 = M/N = \{1, a, b, ab\}$ and assume $K = \{1, a, b\}$. Then consider $x, y, z \in \{1, a\}$ with $xyz = a$, which are all subgroup and coset constraints; furthermore, add $t \in \{1, b\}$ with $(x, t) \in \{(1, 1), (a, b)\}$, which are still subgroup constraints. Finally, $yt = u$ with $u \in K$. Then one of x, y, z is a and the other two are 1, again defining one-in-three SAT and giving NP-completeness. □

THEOREM 35. *Let G be a finite group. Consider the general subgroup problem for G. Include also any number of subsets bK of G^k, where the sets K are nearsubgroups. Then the subgroup problem with these additional nearsubgroup constraints cannot simulate one-in-three SAT (and is thus unlikely to be NP-complete).*

Proof. Since the intersection of nearsubgroups is a nearsubgroup by a result of Aschbacher [3], it suffices to show that a single one-nearsubgroup K of G^3 cannot represent the one-in-three SAT relation via some cK. Suppose that it does, so that cK contains three elements (a, b, b), (b, a, b), (b, b, a) with $a \neq b$. We may assume $b = 1$, and then multiply by $(a^-1, 1, 1)$, so the three triples are $(1, 1, 1)$, $(a^-1, a, 1)$, and $(a^{-1}, 1, a)$ in K. But the product (a^-2, a, a) of the last two is also in K since the last two also commute, so we have a triple (a^{-1}, a, a) together with $(a, 1, 1)$, $(1, a, 1)$, $(1, 1, a)$. That is, the core will still contain some (x, a, a) for $x = 1$ or $x = a$, which does not define one-in-three SAT. □

Consider now a problem whose constraints are subsets of G^k containing the identity element 1; the only constraints that do not contain 1 are single-element subsets $\{a\} \in G$. If every such a is of odd order, we call this the *odd problem* for G. If every such a is of order a power of 2, we call this the 2-*element problem* for G.

THEOREM 36. *Consider a problem on a finite group G with arbitrary constraints. This problem reduces to the odd problem and the 2-element problem together, with constraints aR corresponding to the constraints R in the original problem. Furthermore,* (1) *the odd problem reduces to the subgroup problem for G if all constraints satisfy the cycles condition*; (2) *the 2-element problem reduces to the subgroup problem for G if all constraints K satisfy $S \cap \langle S \cap K \rangle \subseteq K$, where S is the set of two elements, and also $\langle x \rangle \in K$ for $x \in K$. Therefore, the problem for G with nearsubgroup satisfying the 2-element property reduces to the subgroup problem for G and is thus polynomially solvable.*

Proof. The first step takes arbitrary constraints on G and reduces them to odd problems and 2-element problems. The basic idea is that if we have r constraints,

and s of them already contain the identity element 1, then we shall force one more of these constraints to contain 1. Repeating this step eventually forces all constraints to contain 1, and then 1 is a solution. Let K be the chosen set not containing 1, and ignore the remaining $r-s-1$ constraints. For $K \subseteq G^k$ itself, we may just try each of the possible values for k variables involved. Suppose we just consider them one at a time. We have thus reduced the problem to a problem where all constraints contain 1 except for a single constraint that assigns a value to a single variable. If this constant is of odd order or of order a power of two, we have an odd problem or a 2-element problem, respectively. If this constant is of order rs, with r odd and s a power of 2, then it can be written as (a, b) in $Z_r \times Z_s$, where a generates Z_r and b generates Z_s. We initially replace b with a variable constrained to Z_s, so we only have the odd order constant a, and hence an odd problem. After solving the odd problem, we obtain a solution s, and we may look for a solution of the form sx. This means that in $Z_r \times Z_s$ we want the element $(1, s_i^{-1}b)$, and this is now a 2-element problem.

It remains to reduce the odd problem and the 2-element problem to the subgroup problem in the cases mentioned in the theorem. We first consider the odd problem, where the constraints are subsets K of G^k containing 1 and satisfying the cycles condition. Additional constraints just assign a single odd order value to a variable. We replace each variable with two variables. If K is a subset of G^k, we replace it with the subgroup H of G^{2k} generated by the pairs (x, x^{-1}) in K. If a is a single odd order constant, we replace it by the pair $(a^{\frac{1}{2}}, a^{-\frac{1}{2}})$, where $a^{\frac{1}{2}}$ denotes $a^{\frac{r+1}{2}}$, r being the order of a. After a solution is found for the resulting subgroup problem, we replace each pair (x, y^{-1}) in the solution by the product xy to obtain a solution for the original problem. If the original problem had a solution, we could choose s a power of 2 such that $x^s = x$ for odd order elements x and x^s has odd order for all x. Raising the solution to the power s gives an odd order solution t, and we may use the pair $(t^{\frac{1}{2}}, t^{-\frac{1}{2}})$ as a solution to the new problem. Conversely, if the new problem has a solution, then the odd constant pairs $(a^{\frac{1}{2}}, a^{-\frac{1}{2}})$ give the right product value a. Furthermore, a pair (x, y^{-1}) in H can be written as $(x_1 x_2 \cdots x_k, x_1^{-1} x_2^{-1} \cdots x_k^{-1})$ with the x_i in K, and then $xy = x_1 x_2 \cdots x_k x_k \cdots x_2 x_1$ is in K by the cycles condition, as desired.

For the 2-element problem, the constraints are subsets K of G^k containing 1 and satisfying the condition $S \cap \langle S \cap K \rangle \subseteq K$. Additional constraints just assign a single 2-element value to a variable. The main point is that the set K and the subgroup $\langle S \cap K \rangle$ are indistinguishable as far as their 2-elements are concerned, i.e., $S \cap \langle S \cap K \rangle = S \cap K$. So we replace the set K with the subgroup $\langle S \cap K \rangle$ and insist for either problem that the solution consist of 2-elements. As before, we can choose r odd such that $x^r = x$ for 2-elements x and x^r is a 2-element for all x; raising a solution to either problem to the power r guarantees that the solution is a 2-element, as desired. □

Summarizing, non-nearsubgroups give NP-completeness, nearsubgroups give non-NP-completeness unless the reduction is not a simulation of one-in-three SAT, and nearsubgroups with the 2-element property give polynomiality by a reduction to the subgroup case.

We first showed that nearsubgroups are the same as subgroups in the abelian case. We moved on to the nonabelian case, and still showed that nearsubgroups are the same as subgroups for 2-groups. This led us to consider the case of odd order groups, where we encountered a nearsubgroup that is not a subgroup: the elements are triples from Z_p with product operation $(i, j, k)(i', j', k') = (i + i' + jk', j + j', k + k')$

and the nearsubgroup K consists of the elements of the form $(\frac{1}{2}jk, j, k)$. For odd order groups, nearsubgroups immediately satisfy the 2-element property. From this we inferred that for groups that are the product of a 2-group and an odd order group, in particular for nilpotent groups, nearsubgroups satisfy the 2-element property, and so the constraint-satisfaction problem for nearsubgroups is polynomially solvable.

We then asked whether it might always be the case that nearsubgroups satisfy the 2-element property, so that the constraint-satisfaction problem for nearsubgroups is polynomially solvable. Aschbacher found a counterexample, where $K = I$ is the set of involutions (elements of order 2) plus the identity element 1, in a rank-1 simple Lie group of even characteristic with at least one element of order 4.

Because of this example, we considered the case of groups with no element of order 4 and showed for such groups that when the 2-Sylow subgroups have at most four elements, then nearsubgroups satisfy the 2-element property; Aschbacher [3] proved a general theorem that implies that nearsubgroups satisfy the 2-element property for all groups with no element of order 4. Since all groups where we had previously shown that nearsubgroups satisfy the 2-element property are solvable, and Aschbacher's counterexample is not a solvable group, we asked whether there might be any counterexample for solvable groups. Here we could understand the case of a dihedral group, $\langle a, b\rangle$, with $a^2 = b^n = 1$ and $ab = b^{-1}a$; then the only nearsubgroups that are not necessarily subgroups are given for r, s dividing n by the elements of the form b^{ir} and ab^{js}, provided that the same power of 2 divides r and s. Aschbacher showed that nearsubgroups satisfy the 2-element property for solvable groups. Finally, since it is not possible to simulate one-in-three SAT and obtain NP-completeness with nearsubgroups alone, we asked whether nearsubgroups could give a non-nearsubgroup by intersection. Aschbacher showed that this is not possible—the intersection of nearsubgroups is a nearsubgroup.

Summarizing our findings and those of Aschbacher [3], we state the following theorem.

THEOREM 37. *Let G be a finite group.*

(1) *If G is abelian, or a 2-group, then its nearsubgroups are subgroups.*

(2) *Let I be the involutions plus 1. If G has two involutions whose product has order 4, then I is not a nearsubgroup. Otherwise, I is a nearsubgroup. If in addition I generates no element of order 4, then I satisfies the 2-element property; otherwise it does not. There exist groups G that meet this condition for I being a nearsubgroup but not satisfying the 2-element property.*

(3) *If G has no element of order 4, or if it is solvable, then its nearsubgroups satisfy the 2-element property.*

(4) *The intersection of nearsubgroups is a nearsubgroup.*

The fact that there are finite groups G with nearsubgroups that do not satisfy the 2-element property, such as the ones in (2) above, creates an interesting situation. Consider the nearsubgroup problem for G. We know that this problem cannot simulate one-in-three SAT, and is thus unlikely to be NP-complete. On the other hand, only nearsubgroups with the 2-element property seem to be transformable into subgroups so as to obtain a subgroup problem, so the problem might not be polynomially solvable. This is our best candidate for a constraint-satisfaction problem that might be neither polynomially solvable nor NP-complete.

On the other hand, we have identified a property, the *nearsubgroup intersection property*, that would imply that the problem with nearsubgroups is polynomially solvable. We know of no counterexamples to the nearsubgroup intersection property,

and it is implied by the 2-element property. The nearsubgroup intersection property is as follows. Let G be a finite group with a subgroup H of index 2. Then there exists an element g in $G \setminus H$ such that g belongs to every nearsubgroup K of G such that the 2-elements in the intersection of K^* and $G^* \setminus H^*$ generate G^*. Here $G^* = G/N$, where N is some normal subgroup of G (dependent on K) such that H contains N, kN is contained in K for every k in K, and H^*, K^* are the corresponding induced subsets of G^* obtained from H, K. The *weak nearsubgroup intersection property* considers only nearsubgroups K with $N = 1$. That is, for a finite group G with a subgroup H of index 2, there exists an element g in $G \setminus H$ such that g belongs to every nearsubgroup K of G such that the 2-elements in the intersection of K and $G \setminus H$ generate G. The following theorem can be proved using the weak nearsubgroup intersection property; we have chosen to use the stronger definition because it leads to a simpler proof and algorithm.

THEOREM 38. *Suppose that the nearsubgroup intersection property holds. Then the 2-element problem is polynomially solvable for nearsubgroups, and therefore the problem for nearsubgroups is polynomially solvable.*

Proof. We have already shown that to solve the problem with nearsubgroup constraints, it is sufficient to solve the 2-element problem. In the 2-element problem, we have a space which is the direct product of groups $G_1, G_2, \ldots, G_n$, where $G_1 = \langle r \rangle$ is a group generated by a 2-element r (we would like to obtain this element r in a solution), and all constraints are nearsubgroups of the direct product of a bounded number of G_i. Call G the product of all the G_i, and let H be the subgroup of index 2 obtained by replacing G_1 with $G_1' = \langle r^2 \rangle$. If we can obtain a solution g in $G \setminus H$, this solution will have r^{2i+1} in G_1, and then a solution having r itself can be easily obtained since $r = (r^{2i+1})^j$ for some j. So suppose we are looking for a solution g in $G \setminus H$. Consider the first nearsubgroup constraint K, which constrains, say, the direct product of the first k groups G_i. Call N the product of the remaining G_i. By taking the factor group defined by N, we move to G^* and obtain H^* and K^*. It is then sufficient to look for a solution in the group generated by the 2-elements in the intersection of K^* and $G^* \setminus H^*$. This gives a subgroup of G, and we may obtain generators for this subgroup. By carrying out this process of further restricting G by considering each nearsubgroup constraint K in turn, we obtain a decreasing chain of subgroups (we may need to look at the same nearsubgroup more than once; this process is similar to how the subgroup problem was solved). If at the end of this process, the subgroup H still has index 2 in G, then the nearsubgroup intersection property guarantees the existence of a solution with all nearsubgroup constraints. To actually find such a solution, we may successively fix the values in $G_2, G_3, \ldots, G_n$ and test that H still has index 2, until we find a complete solution. □

7. Conclusions and further directions. Every known polynomially solvable problem in CSP can be explained by a combination of Datalog and group theory. In fact, only three specific cases combine to give all known polynomially solvable problems. The three cases are width 1, bounded strict width, and subgroup problems.

These three cases have something in common, which is best illustrated by the following characterizations, where we also include a fourth case where the template is over the reals.

(1) A problem has width 1 if and only if there is a function f that maps nonempty subsets of the template to elements of the template, such that for every relation R in the template, of arity k, the following holds. Let $S_1, S_2, \ldots, S_k$ be subsets with the property that for every x_i in S_i, there exist x_j in S_j for $j \neq i$ such that $(x_1, x_2, \ldots, x_k)$

is in R. Then $(f(S_1), f(S_2), \ldots, f(S_k))$ is in R. The case of extended width 1 is slightly more general, because it only considers a fraction of the subsets of the template. The existence of solutions can also be described in terms of tree duality.

(2) A problem has strict width l if and only if there is a function g that maps $(l+1)$-tuples from the template to elements of the template, such that for every relation R in the template, of arity k, the following holds. First, if all but at most one of some $l+1$ elements x_i are equal to some specific element x, then $g(x_1, x_2, \ldots, x_{l+1}) = x$. Second, let x_{ij} be elements with $(x_{1j}, x_{2j}, \ldots, x_{kj})$ in R for every j. Then $(g(x_{11}, x_{12}, \ldots, x_{1(l+1)}), g(x_{21}, x_{22}, \ldots, x_{2(l+1)}), \ldots, g(x_{k1}, x_{k2}, \ldots, x_{k(l+1)}))$ is in R. The existence of solutions can also be described in terms of the Helly property.

(3) A problem is a subgroup problem if and only if we can define a group operation on the elements of the template such that for every relation R in the template, of arity k, the following holds. If the three tuples $(b_1, b_2, \ldots, b_k)$, $(b_1x_1, b_2x_2, \ldots, b_kx_k)$, $(b_1y_1, b_2y_2, \ldots, b_ky_k)$ are in R, then the tuple $(b_1x_1y_1, b_2x_2y_2, \ldots, b_kx_ky_k)$ is also in R. This means that R is a coset of a subgroup of G^k. The case of nearsubgroups can be viewed as mapping b, bx, by into $bxyz$, rather than into bxy, where z is an element of the commutator group of the subgroup generated by x and y. The algorithm that finds solutions depends on the fact that when G acts on itself every subgroup defines a partition into cosets; we do not know whether a partition theory can also be developed for nearsubgroups.

(4) Convex programming, a constraint satisfaction problem over the reals solvable in polynomial time with the ellipsoid method, has the property that if $(x_1, \ldots, x_k)$ and $(y_1, \ldots, y_k)$ are both in some convex set, then so is $(h_\alpha(x_1, y_1), \ldots, h_\alpha(x_k, y_k))$, where $h_\alpha(x, y) = \alpha x + (1 - \alpha)y$ and $0 \leq \alpha \leq 1$. Here convex programming duality is of interest in the study of solutions.

These four characterizations are all *closure properties*; i.e., there exists a function (f, g, group operation, or h_α) such that every relation R in the template is closed under componentwise application of the function. Furthermore, there is always an associated structure theory for the study of the existence of solutions (tree duality, Helly property, coset partition, convex programming duality, respectively). Schaefer [48] used such closure properties when he classified the polynomially solvable and NP-complete problems in Boolean CSP. He was thus in a sense showing that Horn clauses, 2SAT, and linear equations modulo 2, the three polynomially solvable cases, are width 1, strict width 2, and subgroup, respectively. We do not know whether polynomial solvability for CSP is always necessarily tied to a closure property.

The algorithmic significance of these closure properties is an interesting question. For problems of width 1, the f mapping is not needed to solve the problem in polynomial time, but can be used to obtain a solution directly once the Datalog program has found nonempty sets associated with each variable. For problems of strict width l, once the Datalog program has found nonempty sets associated with l-tuples of variables satisfying the Helly property, the g mapping can be used to obtain a solution efficiently in parallel. There are, however, problems where the f and g mappings help find fast algorithms without running the Datalog program. Here the following results from Feder and Hell [18] are good examples. The connected list problem for reflexive graphs is polynomially solvable for chordal graphs, and NP-complete otherwise. For chordal graphs, this problem has width 1. By using the perfect elimination ordering for perfect graphs, a fast parallel algorithm can be found for this problem, and here the f mapping is based on the existence of a perfect elimination ordering. Similarly, the arbitrary list problem for reflexive graphs is polynomially solvable for

interval graphs, and NP-complete otherwise. For interval graphs, this problem has strict width 2. By using the interval representation for interval graphs, a reduction to 2SAT can be found for this problem, giving again a fast parallel algorithm. Here the g mapping is based on the existence of an interval representation. Thus the f and g mappings can help understand the structure of a problem and lead to fast parallel algorithms. For subgroup problems, the situation is more drastic: we do not know of any algorithm that does not involve finding generators, and here the group operation is used directly.

This raises an important question. For subgroup problems, the set of solutions has a polynomial number of generators. This means that there exists a polynomial number of solutions such that if we take the closure under the mapping that maps s, sx, sy to sxy, then we obtain all solutions. Now we may ask whether there exist a polynomial number of generators for the width 1 and bounded strict width cases, when closure under f and g mappings is considered. We first study the bounded strict width case. Here there does exist a polynomial number of generators; namely, if a problem has strict width l, then for each choice of l variables $x_1, x_2, \ldots, x_l$, determine the assignments of values to these variables for which a solution exists, and choose one solution for each assignment. This produces at most n^l generators. To see that these are generators, suppose that we have a solution and consider the values it assigns to $l+1$ variables $x_1, \ldots, x_{l+1}$. For each choice of a variable x_i, there exists a generator that assigns the correct value to the remaining x_j, but not necessarily to x_i. Find $l+1$ such generators, one for each i, and then applying the g mapping to them will assign the correct value to the $l+1$ variables. Inductively, we can then obtain the correct assignment for all variables. Now consider the width 1 case. Here we write $f(S) = t$ for a set of solutions S and a solution t if for each variable x_i, letting S_i be the set of values taken by x_i in S, and letting t_i be the value of x_i in t, we have $f(S_i) = t_i$. In general, the number of generators needed for this f mapping is exponential. For Horn clauses, with $f(\{0\}) = 0$, $f(\{1\}) = 1$, and $f(\{0,1\}) = 0$, the mapping $f(S) = t$ corresponds to set intersection. Even for independent set, a special case of Horn clauses, the set of generators with the f mapping must contain at least the maximal number of independent sets, and there can be an exponential number of them. Independent set, on the other hand, is also a special case of 2SAT, and the number of generators with the g mapping is polynomial. For linear programming, the generators with the h_α mappings are the vertices of the polytope, and there can be an exponential number of them. Notice now that Horn clauses and linear programming are in general P-complete, while neither bounded strict width problems nor subgroup problems seem to be P-complete; it might be that P-completeness is tied to the nonexistence of a closure property that allows for a small (polynomial or at least nonexponential) set of generators.

Our attempt to classify the problems in CSP and establish a dichotomy is based on the following two conjectures.

CONJECTURE 1. *A constraint-satisfaction problem is not in Datalog if and only if the associated core T can simulate a core T' consisting of two relations C, Z that give the ability to count. This is equivalent to simulating either Z_p or one-in-three SAT.*

CONJECTURE 2. *A constraint-satisfaction problem is NP-complete if and only if the associated core T can simulate a core T' consisting of the single relation C defining one-in-three SAT.*

The first conjecture indicates a sharp line out of Datalog and into group theory:

since one-in-three SAT can simulate Z_2, and, when the two linear equations $x = 0$, $x + y + z = 1$ modulo p give the ability to count, then every linear equation modulo p can be simulated, it follows that the conjecture basically says that a problem not in Datalog is at least as powerful as the general subgroup problem for Z_p.

So we assume that a template not in Datalog contains at least the general subgroup problem for Z_p. We then move on to the second conjecture, which indicates a sharp line into NP-completeness. We thus try to determine what can make a problem that can simulate Z_p able to simulate one-in-three SAT, and thus NP-complete. We first observe that on the p elements that simulate Z_p, any relation that is not a subgroup or a coset in a power of Z_p makes it possible to simulate one-in-three SAT. It is thus not possible to interfere with Z_p itself and avoid NP-completeness. This suggests that the only way to enlarge the Z_p problem and still obtain a problem that cannot simulate one-in-three SAT may be to view Z_p as a subgroup of a larger, not necessarily abelian, group. It seems that any other approach to extending Z_p would simply combine Z_p with other problems without interfering with Z_p itself, either by taking the product of Z_p with another problem or by encoding Z_p in a special class such as digraph-homomorphism.

Suppose then that we have the general subgroup problem for a finite group, which is still polynomially solvable. It is no longer true that adding a nonsubgroup makes it possible to simulate one-in-three SAT. We can show that adding a non-nearsubgroup makes it possible to simulate one-in-three SAT. So we only allow nearsubgroups. The intersection of nearsubgroups gives nearsubgroups. So nearsubgroups do not make it possible to simulate one-in-three SAT, which by the second conjecture would mean that adding nearsubgroups to a subgroup problem cannot make the problem NP-complete. We can show that if we restrict our attention further to nearsubgroups with the 2-element property, then nearsubgroups can be replaced with related subgroups, and the resulting problem reduces to subgroup problems and is thus polynomially solvable.

But then nearsubgroups have the 2-element property for many groups, including solvable groups and groups with no element of order 4, by results of Aschbacher [3]. It may then be that a subgroup problem always remains polynomially solvable when we add nearsubgroups. However, Aschbacher identifies a group with a nearsubgroup that does not have the 2-element property. It does not seem possible to represent this nearsubgroup as a subgroup, so the problem does not immediately reduce to a subgroup problem, although it still contains the general subgroup problem for the chosen group; this may mean that the problem is not polynomially solvable. On the other hand, the fact that nearsubgroups cannot simulate one-in-three SAT may mean that the problem is not NP-complete.

We thus have our best candidate for a problem in CSP that may be neither polynomially solvable nor NP-complete, which is the following. Consider the general subgroup problem for a finite group and focus on the set of involutions, including the identity, which we wish to add as a new constraint. If some element of order 4 is the product of two involutions, then involutions are not a nearsubgroup and the problem is NP-complete. Otherwise, the involutions are a nearsubgroup. If no element of order 4 is generated by involutions, then the involutions have the 2-element property, and the problem is polynomially solvable. Otherwise, the involutions do not have the 2-element property. Aschbacher observed that there are such finite groups, where the involutions form a nearsubgroup without the 2-element property. Could it be that for such groups the involutions define a problem that is neither polynomially solvable

nor NP-complete? On the other hand, the following may hold.

CONJECTURE 3. *The nearsubgroup intersection property holds for all finite groups. Therefore, the constraint-satisfaction problem with nearsubgroup constraints is polynomially solvable.*

This is the current state of the attempt to classify the problems in CSP and obtain a dichotomy. Considering the three main conjectures, it may be that there is a direct approach towards proving the first one, concerning the ability to count. One might start by showing that if a subgroup problem does not have the ability to count, then it can be solved with Datalog, beginning with abelian groups. Similarly, one could consider linear programming, a constraint-satisfaction problem over the reals that gets the ability to count by representing bipartite matching or linear equations, and show that if a linear programming template does not have the ability to count, then it can be solved with Datalog.

The second conjecture cannot be approached directly, as we cannot show non-NP-completeness without showing P≠NP. It might still be possible to show that if a CSP problem cannot simulate one-in-three SAT, then it belongs to a class that is unlikely to contain NP-complete problems, such as co-NP; this would establish a dichotomy, namely, NP-complete versus in NP∩co-NP. An approach along these lines was successful in showing that graph isomorphism is not NP-complete unless the polynomial hierarchy collapses. The approach there was via interactive proofs for graph nonisomorphism; interactive proofs have already been shown to be relevant in the study of constraint satisfaction problems; see [20, 7].

It might be possible to answer the third conjecture with the theory of nearsubgroups as studied by Aschbacher [3].

A basic question remains open. Find a deterministic construction of small graphs of high chromatic number and high girth, in particular with the size of the graph polynomial in the chromatic number, with the girth lower-bounded by a constant. This would help derandomize the reduction between equivalent MMSNP and CSP problems.

The class NP consists of all problems expressible by an existential second-order sentence with an arbitrary first-order part (see [14]; the first-order part need not be universal, as for SNP). What is the complexity of problems in *MMNP*, the class *monotone monadic NP without equality or inequality*?

Appendix: Constraint satisfaction and link systems. In his paper "Process, system, causality, and quantum mechanics" [12], Etter proposes link systems as a unifying framework for i-o systems, Markov processes, and quantum mechanics; he also suggests that link systems provide an interpretation for relativity and for relational databases. In this paper, we have proposed constraint satisfaction as a unifying framework for problems that can be solved with Datalog, such as Horn clauses and 2-satisfiability; for NP-complete problems, such as 3-coloring and one-in-three-satisfiability; for problems from group theory, such as systems of linear equations and labeled graph isomorphism; for linear programming, graph matching, and a family of matroid parity problems; and for network stability and stable matching.

In this Appendix we intend to expose a direct connection between constraint satisfaction and link systems, so that this work and the work of Etter may benefit from each other.

A *link system* is a stochastic process on a set of random variables $x_1, x_2, \ldots, x_n$, together with a set of *links*. A link is a condition of the form $x = y$ on two variables x and y of a stochastic process, thereby creating a new process in which the uncondi-

tional probability $p(E)$ of any event E is $p(E|x = y)$ in the old process. The duplicate variable y is then dropped.

A *proper link system* is a link system in which all links are between different independent parts of the stochastic process, and no two links involve the same variable. Etter is primarily interested in proper link systems.

A *white variable* is a random variable with a uniform distribution. An *i-o system* is a link system where every link is of the form $x = w$ and where w is a white variable, with x and w independent. Notice that when the process is conditioned on $x = w$, so that x and w are linked and w is dropped, the distribution of x in the resulting stochastic process is the same as in the original process, because w is white.

Suppose that x and y are independent random variables. Then the probability that x and y will both have some value k is the product of the probabilities that they will have value k separately, i.e., $p(x = y = k) = p(x = k)p(y = k)$. Let's call $p(x = k)$ and $p(y = k)$ the *unlinked* probabilities of x being k and of y being k. Now suppose we impose the link condition $x = y$. The *linked* probability is then the probability, conditioned by $x = y$, that both $x = k$ and $y = k$. The probability that x is k, as a function of k, is proportional to $p(x = k)p(y = k)$.

In short, linked probabilities are always quadratic in unlinked probabilities. If the distributions of x and y are identical, which is the quantum situation, then linked probabilities are the squares of unlinked probabilities. That is, quantum amplitudes are simply unlinked probabilities, which explains the square root law. The main additional element that appears in quantum mechanics is that probability theory must be extended by allowing cases to count negatively. Much of the trouble that we have when looking for an interpretation of quantum mechanics is that we are looking for an i-o system, where one of the two linked variables is white, when instead it is the case that the two variables are identically distributed.

This is in essence a summary of some of the basic ideas in Etter's paper. He also shows that the velocity law of relativity, which adds velocities as $(v + v')/(1 + vv')$, where the speed of light is 1, can be viewed as linking two variables x and x' corresponding to coin tosses, where the velocity is the probability of heads minus the probability of tails.

In a second paper, "Quantum mechanics as a branch of mereology" [13] Etter proposes an approach to link systems from relational databases instead of probability theory. The word *mereology* means the mathematical theory of parts and wholes. The idea is to use the record count for database tables instead of probabilities. Linking is then defined in the usual sense for combining tables in relational databases. For example, let A be a two-field table with record count $A(x, y)$ and let B be another two-field table with record count $B(y, z)$. When A and B are linked at y to obtain a three-field table, the record counts multiply, giving record count $A(x, y)B(y, z)$ for the entry given by the triple x, y, z. When we hide the linked field y, we obtain a two-field table C, adding the record counts over y, so the record count is $C(x, z) = \sum_y A(x, y)B(y, z)$. In short, when A, B, C are viewed as matrices, we have $C = AB$. That is, linking is intimately tied to matrix multiplication. Etter uses this connection to examine the two core laws of quantum mechanics, the Born probability rule and the unitary dynamical law (whose best-known form is Schrödinger's equation), formulated by von Neumann in the language of Hilbert space in terms of matrices as $\text{prob}(P) = \text{trace}(PS)$ and $S'T = TS$, respectively. Etter observes that the algebraic forms of these two core laws occur as completely general theorems about links.

We now turn to the connection with constraint satisfaction. Given an instance

I of constraint satisfaction, suppose we create an auxiliary instance I' where every variable in I has been replaced by many variables, so that every variable occurs in precisely one constraint. The instance I' always has a solution, namely, satisfying each constraint separately. Now view I' as a stochastic process, by requiring for each constraint $(x_1, \ldots, x_k) \in R$ that the variables x_i be distributed so as to give a uniform distribution over the satisfying assignments for R. Now I can be viewed as the system obtained from I' by linking the many variables that came from the same variable. The resulting distribution is then the uniform distribution over the set of satisfying assignments to the instance I.

This is in essence the connection between constraint satisfaction and link systems. We may then ask what part of the framework of Etter makes sense in the context of constraint satisfaction. A first observation is that the use of Datalog allows for rules such as $C(x, z)$:–$A(x, y), B(y, z)$. When conjunctions and disjunctions are replaced by multiplication and addition, respectively, the rule becomes $C(x, z) = \sum_y A(x, y)B(y, z)$, which we saw arises in link systems.

Consider next the network stability problem. Here the constraints are gates; i.e., they involve $k+l$ variables via a functional constraint $f(x_1, x_2, \ldots, x_k) = (y_1, y_2, \ldots, y_l)$. In the instance I', prior to the linking, the x_i variables are independently and uniformly distributed. All the links are of the form $y = x$, matching an output y to an input x, where the input x is uniformly distributed, and hence a white variable. Therefore, network stability is just an i-o system, in Etter's terminology. Stable matching is known to be the special case of network stability where all variables are Boolean and all gates are the X gate $X(x_1, x_2) = (x_1\overline{x_2}, x_2\overline{x_1})$, in addition to the constant generating gates with $k = 0$ and $l = 1$, as well as the absorption gate with $k = 1$ and $l = 0$. For every network there is an underlying digraph, namely, the digraph with a vertex for every constraint C, and a directed edge from C to C' if an output of C is an input to C'. This digraph need not be acyclic in general and, for example, stable matching problems give digraphs that are usually not acyclic. Then the network stability may not have any solutions or more than one solution. When the digraph is acyclic, the network is called a circuit, and there is then a unique solution obtained by evaluating the gates in the order in which they appear in the digraph.

Notice that in the case of network stability, variables occur in disjoint linked pairs, as in Etter's proper linked systems. In general, we have not made the assumption that every variable occurs in precisely two constraints in this paper. Without this assumption, we have conjectured that the constraint-satisfaction problems are either solved with Datalog or have the ability to count; in the latter case, we either have linear equations or one-in-three satisfiability. Suppose we make the same conjecture in the case where every variable occurs in precisely two constraints; then one-in-three satisfibility becomes a polynomially solvable problem, namely, graph matching.

In fact, graph matching generalizes to a family of matroid parity problems as a constraint-satisfaction problem with the property that every variable occurs in only two constraints. To see this assume that each relation in the template is a matroid. Then all relations occurring in an instance form a single matroid that decomposes into a collection of small matroids—one for each relation. The constraints that pair up the variables define a matroid parity problem. Notice that the bases exchange property acts once again as a closure property. By limiting the number of relations involving a single variable, we are in essence limiting fanout. In the case of network stability, every variable occurs in one relation as an input and in one relation as an output. This limits fanout unless we have a copy gate $f(x) = (x, x)$ that creates fanout. Mayr

and Subramanian [39] observed that in the Boolean case, forbidding such a copy gate gives rise precisely to the adjacency-preserving case. In the case where the relations are arbitrary, not necessarily gates, the constraint that each variable appear in only two constraints limits fanout unless we have a copy relation $R(x, x, x)$ that creates fanout. Here we observe that in the Boolean case, if we consider only relations where all tuples have the same number of zeros and the same number of ones, then forbidding such a copy relation gives rise precisely to the matroid parity problems just described. It is also interesting whether limiting fanout has relevance in the quantum context, as suggested by applications in quantum cryptography.

Now let us look at the definition of the ability to count in detail. This definition involves a system I' consisting of two disjoint parts A and B that do not share any variables; and the links that produce I involve a pair of variables, one from A and one from B, with each variable participating in exactly one pair. This is precisely the main case of interest for Etter, where removing the links disconnects the system into two independent parts A and B.

Finally, when we looked at group-theoretic problems, specifically in the odd order problem, we saw that to turn the nearsubgroup problem into a subgroup problem, each variable x had to be replaced by a pair $(x^{1/2}, x^{-1/2})$. That is, just as in quantum mechanics, we have a square root law. It would be interesting to see whether the fact that a square root law arises in group theory is not a coincidence, since group theory has central importance in quantum mechanics. In fact, could it be that nearsubgroups have some interpretation in quantum physics?

Etter has observed that, given that constraint satisfaction is the problem of mapping a structure S to a fixed template T, it makes sense to exchange the roles of S and T and to look for a homomorphism from T to S. He has observed that this duality has something in quantum mehanics that seems related to this strange reversal, which is the Foch-space representation of the wave-particle duality that reverses the role of vector and tensor components in Hilbert space. He suspects that duality will turn out to be a key concept for unifying quantum mechanics with relativity. In the context of constraint satisfaction, the simplest example of this form of duality is the case where the template is a clique K_r on r vertices. Mapping S to K_r is the problem of coloring a graph with r colors, while mapping K_r to S is the problem of whether a graph has a clique of size r.

Acknowledgments. We benefitted greatly from early discussions with Yatin Saraiya and with Peter Winkler. Christos Papadimitriou suggested the connection between constraint satisfaction and two problems with a fixed structure that were previously studied, graph-homomorphism and the Boolean domain case. Milena Mihail suggested that quasi-random graphs may derandomize the representation of monotone monadic SNP in CSP. Tom McFarlane showed us the paper of Tom Etter on link systems. We also had very valuable conversations with Miki Ajtai, Michael Aschbacher, Yossi Azar, Laszlo Babai, Ron Fagin, Jim Hafner, Pavol Hell, Rajeev Motwani, and Moni Naor. The comments of two anonymous referees were very helpful in the final writing of this paper.

REFERENCES

[1] S. Arnborg, J. Lagergren, and D. Seese, *Easy problems for tree-decomposable graphs*, J. Algorithms, 12 (1991), pp. 308–340.

[2] M. Aschbacher, *Finite Group Theory*, Cambridge Stud. Adv. Math. 10, Cambridge University Press, Cambridge, UK, 1986.

[3] M. Aschbacher, *Near Subgroups of Finite Groups*, manuscript.
[4] F. Afrati and S. S. Cosmadakis, *Expressiveness of restricted recursive queries*, in Proc. 21st ACM Symp. on Theory of Computing, ACM, New York, 1989, pp. 113–126.
[5] F. Afrati, S. S. Cosmadakis, and M. Yannakakis, *On Datalog vs. polynomial time*, in Proc. 10th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, ACM, New York, 1991, pp. 13–25.
[6] L. Babai, *Monte Carlo Algorithms in Graph Isomorphism Testing*, manuscript, 1979.
[7] R. Bačík and S. Mahajan, *Semidefinite Programming and Its Applications to NP Problems*, manuscript, 1995.
[8] J. Bang-Jensen and P. Hell, *The effect of two cycles on the complexity of colourings by directed graphs*, Discrete Appl. Math., 26 (1990), pp. 1–23.
[9] H. L. Bodlaender, *Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees*, J. Algorithms, 11 (1990), pp. 631–643.
[10] R. Dechter, *Constraint networks*, in Encyclopedia of Artificial Intelligence, 1992, pp. 276–285.
[11] P. Erdős, *Graph theory and probability*, Canad. J. Math., 11 (1959), pp. 34–38.
[12] T. Etter, *Process, system, causality, and quantum mechanics – a psychoanalysis of animal faith*, Internat. J. General Systems (Special Issue on General Systems and the Emergence of Physical Structure from Information Theory).
[13] T. Etter, *Quantum mechanics as a branch of mereology*, extended abstract, Phys. Comp., 116 (1995).
[14] R. Fagin, *Generalized first-order spectra, and polynomial-time recognizable sets*, in Complexity of Computations, R. Karp, ed., AMS, Providence, RI, 1974.
[15] T. Feder, *Stable Networks and Product Graphs*, Ph.d. Thesis, Stanford University, Stanford, CA, 1991; also Mem. Amer. Math. Soc. 555, AMS, Providence, RI, 1995.
[16] T. Feder, *Removing Inequalities and Negation for Homomorphism-Closed Problems*, manuscript.
[17] T. Feder, *Classification of Homomorphisms to Oriented Cycles and of k-Partite Satisfiability Problems*, manuscript.
[18] T. Feder and P. Hell, *List Problems for Reflexive Graphs*, manuscript.
[19] T. Feder and P. Hell, *Homomorphism Problems on Graphs with Some Self-Loops*, manuscript.
[20] U. Feige and L. Lovász, *Two-prover one-round proof systems: Their power and their problems*, 24th Annual ACM Symp. on Theory of Computing, ACM, New York, 1994, pp. 422–431.
[21] M. Furst, J. E. Hopcroft, and E. Luks, *Polynomial-time algorithms for permutation groups*, in Proc. 21st IEEE Symp. on Found. of Comp. Sci., IEEE, Piscataway, NJ, 1980, pp. 36–41.
[22] D. M. Goldschmidt, *2-fusion in finite groups*, Ann. Math., 99 (1974), pp. 70–117.
[23] W. Gutjahr, E. Welzl, and G. Woeginger, *Polynomial graph colourings*, Discrete Appl. Math., 35 (1992), pp. 29–46.
[24] P. Hell and J. Nešetřil, *On the complexity of H-coloring*, J. Combin. Theory Ser. B, 48 (1990), pp. 92–110.
[25] P. Hell, J. Nešetřil, and X. Zhu, *Duality and polynomial testing of tree homomorphisms*, Trans. Amer. Math. Soc.
[26] P. Hell, J. Nešetřil, and X. Zhu, *Complexity of tree homomorphisms*, Discrete Appl. Math.
[27] P. Hell, J. Nešetřil, and X. Zhu, *Duality of graph homomorphisms*, in Combinatorics, Paul Erdos is Eighty, Bolyai Soc. Math. Stud. 2, 1995.
[28] P. Hell and X. Zhu, *Homomorphisms to oriented paths*, Discrete Math., 132 (1994), pp. 107–114.
[29] P. Hell and X. Zhu, *The existence of homomorphisms to oriented cycles*, SIAM J. Discrete Math., 8 (1995), pp. 208–222.
[30] G. G. Hillebrand, P. C. Kanellakis, H. G. Mairson, and M. Y. Vardi, *Tools for Datalog boundedness*, in Proc. 10th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, ACM, New York, 1991, pp. 1–12.
[31] C. M. Hoffmann, *Group-Theoretic Algorithms and Graph Isomorphism*, Lecture Notes in Comput. Sci. 136 Springer-Verlag, New York, 1982.
[32] P. G. Kolaitis and M. Y. Vardi, *The decision problem for the probabilities of higher-order properties*, in Proc. 19th ACM Symp. on Theory of Computing, 1987, pp. 425–435.
[33] P. G. Kolaitis and M. Y. Vardi, *On the expressive power of Datalog: Tools and a case study*, in Proc. 9th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, ACM, New York, 1990, pp. 61–71.
[34] V. Kumar, *Algorithms for constraint-satisfaction problems*, AI Magazine, 13 (1992), pp. 32–44.
[35] R. E. Ladner, *On the structure of polynomial time reducibility*, J. Assoc. Comput. Mach., 22 (1975), pp. 155–171.

[36] V. S. Lakshmanan and A. O. Mendelzon, *Inductive pebble games and the expressive power of Datalog*, in Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, ACM, New York, 1989, pp. 301–310.
[37] A. Lubiw, *Some NP-complete problems similar to graph isomorphism*, SIAM J. Comput., 10 (1981), pp. 11–21.
[38] J. Matoušek and R. Thomas, *Algorithms finding tree-decompositions of graphs*, J. Algorithms, 12 (1991), pp. 1–22.
[39] E. Mayr and A. Subramanian, *The complexity of circuit value and network stability*, J. Comput. System Sci., 44 (1992), pp. 302–323.
[40] P. Meseguer, *Constraint satisfaction problem: An overview*, AICOM, 2 (1989), pp. 3–16.
[41] J. C. Mitchell, *Coercion and type inference (summary)*, in Conf. Rec. 11th ACM Symp. on Principles of Programming Languages, ACM, New York, 1984, pp. 175–185.
[42] P. Lincoln and J. C. Mitchell, *Algorithmic aspects of type inference with subtypes*, in Conf. Rec. 19th ACM Symp. on Principles of Programming Languages, ACM, New York, 1992, pp. 293–304.
[43] M. Wand and P. M. O'Keefe, *On the complexity of type inference with coercion*, in Conf. on Functional Programming Languages and Computer Architecture, 1989.
[44] C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.
[45] V. Pratt and J. Tiuryn, *Satisfiability of Inequalities in a Poset*, manuscript.
[46] A. A. Razborov, *Lower bounds on monotone complexity of the logical permanent*, Math. Notes Acad. Sci. USSR, 37 (1985), pp. 485–493.
[47] N. Robertson and P. Seymour, *Graph minors.* II. *Algorithmic aspects of tree-width*, J. Algorithms, 7 (1985), pp. 309–322.
[48] T. J. Schaefer, *The complexity of satisfiability problems*, in Proc. 10th ACM Symp. on Theory of Computing, ACM, New York, 1978, 216–226.
[49] E. Tardos, *The gap between monotone and non-monotone circuit complexity is exponential*, Combinatorica, 7–4 (1987), pp. 141–142.
[50] J. D. Ullman, *Principles of Database and Knowledge-Base Systems*, Vol. I, Computer Science Press, Rockville, MD, 1989.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 105–136

ASYMPTOTICALLY TIGHT BOUNDS FOR PERFORMING BMMC PERMUTATIONS ON PARALLEL DISK SYSTEMS*

THOMAS H. CORMEN†, THOMAS SUNDQUIST‡, AND LEONARD F. WISNIEWSKI§

Abstract. This paper presents asymptotically equal lower and upper bounds for the number of parallel I/O operations required to perform bit-matrix-multiply/complement (BMMC) permutations on the Parallel Disk Model proposed by Vitter and Shriver. A BMMC permutation maps a source index to a target index by an affine transformation over $GF(2)$, where the source and target indices are treated as bit vectors. The class of BMMC permutations includes many common permutations, such as matrix transposition (when dimensions are powers of 2), bit-reversal permutations, vector-reversal permutations, hypercube permutations, matrix reblocking, Gray-code permutations, and inverse Gray-code permutations. The upper bound improves upon the asymptotic bound in the previous best known BMMC algorithm and upon the constant factor in the previous best known bit-permute/complement (BPC) permutation algorithm. The algorithm achieving the upper bound uses basic linear-algebra techniques to factor the characteristic matrix for the BMMC permutation into a product of factors, each of which characterizes a permutation that can be performed in one pass over the data.

The factoring uses new subclasses of BMMC permutations: memoryload-dispersal (MLD) permutations and their inverses. These subclasses extend the catalog of one-pass permutations.

Although many BMMC permutations of practical interest fall into subclasses that might be explicitly invoked within the source code, this paper shows how to quickly detect whether a given vector of target addresses specifies a BMMC permutation. Thus, one can determine efficiently at run time whether a permutation to be performed is BMMC and then avoid the general-permutation algorithm and save parallel I/Os by using the BMMC permutation algorithm herein.

Key words. bit-defined permutations, BMMC permutations, matrix factoring, parallel disk systems, parallel I/O, potential functions, universal lower bounds

AMS subject classifications. 15A03, 15A23, 68Q05, 68Q22, 68Q25

PII. S0097539795283681

1. Introduction.

From both the theoretical and practical points of view, permuting is an interesting and important problem when the data reside on disk. As one of the most basic data-movement operations, permuting is central to the theory of I/O complexity. The problems that we attack with supercomputers are ever-increasing in size, and in several applications matrices and vectors exceed the memory provided by even the largest supercomputers. (Such applications include seismic problems, compu-

*Received by the editors March 18, 1995; accepted for publication (in revised form) November 1, 1996; published electronically June 15, 1998. An extended abstract of this paper appeared in the Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures.

http://www.siam.org/journals/sicomp/28-1/28368.html

†Department of Computer Science, Dartmouth College, Hanover, NH 03755 (thc@cs.dartmouth.edu). Portions of this research were performed while at the MIT Laboratory for Computer Science and appear in [9] and were supported in part by the Defense Advanced Research Projects Agency under grant N00014-91-J-1698 during that time. Other portions of this research were performed while at Dartmouth College and were supported in part by funds from Dartmouth College and in part by the National Science Foundation under grant CCR-9308667.

‡Department of Mathematics, Dartmouth College, Hanover, NH 03755. Current address: Secure Computing Corporation, Roseville, MN 55113 (sundquis@sctc.com). This research was supported in part by funds from Dartmouth College.

§Department of Computer Science, Dartmouth College, Hanover, NH 03755. Current address: HPC Group, Sun Microsystems Computer Company, Chelmsford, MA 01824 (lenbo@east.sun.com). This research was supported in part by INFOSEC grant 3-56666, by the National Science Foundation under grant CCR-9308667, and by a Dartmouth Graduate Fellowship.

tational fluid dynamics, and processing large images. For a list of "grand challenge" applications with huge I/O requirements, see the list compiled by del Rosario and Choudhary [14].) One solution is to store large matrices and vectors on parallel disk systems. The high latency of disk accesses makes it essential to minimize the number of disk I/O operations. Permuting the elements of a matrix or vector is a common operation, particularly in the data-parallel style of computing, and good permutation algorithms can provide significant savings in disk-access costs over poor ones when the data reside on parallel disk systems.

This paper examines the class of bit-matrix-multiply/complement (BMMC) permutations for parallel disk systems and derives four important results:

1. a universal lower bound for BMMC permutations,
2. an algorithm for performing BMMC permutations whose I/O complexity asymptotically matches the lower bound, thus making it asymptotically optimal,
3. an efficient method for determining at run time whether a given permutation is BMMC, thus allowing us to use the BMMC algorithm if it is, and
4. two new subclasses of BMMC permutations, memoryload-dispersal (MLD) permutations and their inverses, which we show how to perform in one pass.

Depending on the exact BMMC permutation, our asymptotically optimal bound may be significantly lower than the asymptotically optimal bound proven for general permutations. Moreover, the low constant factor in our algorithm makes it very practical.

Model and previous results. We use the Parallel Disk Model first proposed by Vitter and Shriver [24], who also gave asymptotically optimal algorithms for several problems including sorting and general permutations. In the Parallel Disk Model, N *records* are stored on D *disks* $\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_{D-1}$, with N/D records stored on each disk. The records on each disk are partitioned into *blocks* of B records each. When a disk is read from or written to, an entire block of records is transferred. Disk I/O transfers records between the disks and a *random-access memory* (which we shall refer to simply as "memory") capable of holding M records. Each *parallel I/O operation* transfers up to D blocks between the disks and memory, with at most one block transferred per disk, for a total of up to BD records transferred. We assume *independent I/O*, in which the blocks accessed in a single parallel I/O may be at any locations on their respective disks, as opposed to *striped I/O*, which has the restriction that the blocks accessed in a given operation must be at the same location on each disk.

We measure an algorithm's efficiency by the number of parallel I/O operations it requires. Although this cost model does not account for the variation in disk access times caused by head movement and rotational latency, programmers often have no control over these factors. The number of disk accesses, however, can be minimized by carefully designed algorithms. Optimal algorithms have appeared in the literature for fundamental problems such as sorting [3, 6, 21, 22, 24], general permutations [24], and structured permutations [9, 10, 26], as well as higher-level domains such as fast Fourier transform (FFT) [24], matrix-matrix multiplication [24], LUP decomposition [27], computational geometry problems [5, 18], graph algorithms [8], and boolean function manipulation [4].

For convenience, we use the following notation extensively:

$$b = \lg B\ , \quad d = \lg D\ , \quad m = \lg M\ , \quad n = \lg N\ .$$

We shall assume that b, d, m, and n are nonnegative integers, which implies that B, D, M, and N are exact powers of 2. In order for the memory to accomodate

	$\mathcal{D}_0$		$\mathcal{D}_1$		$\mathcal{D}_2$		$\mathcal{D}_3$		$\mathcal{D}_4$		$\mathcal{D}_5$		$\mathcal{D}_6$		$\mathcal{D}_7$	
stripe 0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
stripe 1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
stripe 2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
stripe 3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

FIG. 1. *The layout of $N = 64$ records in a parallel disk system with $B = 2$ and $D = 8$. Each box represents one block. The number of stripes is $N/BD = 4$. Numbers indicate record indices.*

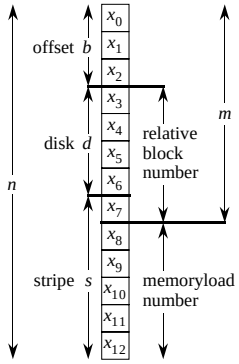


FIG. 2. *Parsing the address $x = (x_0, x_1, \ldots, x_{n-1})$ of a record on a parallel disk system. Here, $n = 13$, $b = 3$, $d = 4$, $m = 8$, and $s = 6$. The least significant b bits contain the offset of a record within its block, the next d bits contain the disk number, and the most significant s bits contain the stripe number. The most significant $n - m$ bits form the record's memoryload number, and bits $b, b+1, \ldots, m-1$ form the relative block number, used in section 4.*

the records transferred in a parallel I/O operation to all D disks, we require that $BD \leq M$. Also, we assume that $M < N$, since otherwise we can just perform all operations in memory. These two requirements imply that $b + d \leq m < n$.

The Parallel Disk Model lays out data on a parallel disk system as shown in Fig. 1. A *stripe* consists of the D blocks at the same location on all D disks. We indicate the *address*, or *index*, of a record as an n-bit vector x with the least significant bit first: $x = (x_0, x_1, \ldots, x_{n-1})$. Record indices vary most rapidly within a block, then among disks, and finally among stripes. As Fig. 2 shows, the offset within the block is given by the least significant b bits $x_0, x_1, \ldots, x_{b-1}$, the disk number by the next d bits $x_b, x_{b+1}, \ldots, x_{b+d-1}$, and the stripe number by the $s = n-(b+d)$ most significant bits $x_{b+d}, x_{b+d+1}, \ldots, x_{n-1}$.

Since each parallel I/O operation accesses at most BD records, any algorithm that must access all N records requires $\Omega(N/BD)$ parallel I/Os, and so $O(N/BD)$ parallel I/Os is the analogue of linear time in sequential computing. Vitter and Shriver showed an upper bound of $\Theta(\min(\frac{N}{D}, \frac{N}{BD}\frac{\lg(N/B)}{\lg(M/B)}))$ parallel I/Os for general permutations, that is, for arbitrary mappings $\pi : \{0, 1, \ldots, N-1\} \overset{1\text{-}1}{\rightarrow} \{0, 1, \ldots, N-1\}$. The first term comes into play when the block size B is small, and the second term is the sorting bound $\Theta(\frac{N}{BD}\frac{\lg(N/B)}{\lg(M/B)})$, which was shown by Vitter and Shriver for randomized sorting and subsequently by Nodine and Vitter [22] and others [3, 6, 21] for deterministic sorting. These bounds are asymptotically tight, because they match the lower bounds proven earlier by Aggarwal and Vitter [2] using a model with one disk and D independent read/write heads, which is at least as powerful as the Parallel Disk Model.

TABLE 1

Classes of permutations, their characteristic matrices, and upper bounds shown in [10] *on the number of passes needed to perform them. A pass consists of reading and writing each record exactly once and therefore uses exactly* $2N/BD$ *parallel I/Os. For MRC permutations, submatrix dimensions are shown on matrix borders. For BMMC permutations,* r *is the rank of the leading* $\lg M \times \lg M$ *submatrix of* A*, and the function* $H(N, M, B)$ *is given by equation* (1)*. For BPC permutations, the function* $\rho(A)$ *is defined in equation* (3)*.*

Permutation	Characteristic matrix	Number of passes
BMMC (bit-matrix-multiply/ complement)	Nonsingular matrix A	$2\left\lceil \frac{\lg M - r}{\lg(M/B)} \right\rceil + H(N, M, B)$
BPC (bit-permute/ complement)	Permutation matrix A	$2\left\lceil \frac{\rho(A)}{\lg(M/B)} \right\rceil + 1$
MRC (memory-rearrangement/ complement)	$\begin{array}{c} \begin{array}{cc} m & n-m \end{array} \\ \left[\begin{array}{c|c} \text{nonsingular} & \text{arbitrary} \\ \hline 0 & \text{nonsingular} \end{array}\right] \begin{array}{l} m \\ n-m \end{array} \end{array}$	1

Specific classes of permutations sometimes require fewer parallel I/Os than general permutations. Vitter and Shriver showed how to transpose an $R \times S$ matrix ($N = RS$) with only $\Theta(\frac{N}{BD}(1 + \frac{\lg \min(B,R,S,N/B)}{\lg(M/B)}))$ parallel I/Os. Subsequently, Cormen [10] studied several classes of bit-defined permutations that include matrix transposition as a special case. Table 1 shows some of the classes of permutations examined and the corresponding upper bounds derived in [10].

BMMC permutations. The most general class considered in [10] is that of BMMC permutations.[1] A BMMC permutation is specified by an $n \times n$ *characteristic matrix* $A = (a_{ij})$ whose entries are drawn from $\{0, 1\}$ and is nonsingular (i.e., invertible) over $GF(2)$.[2] The specification also includes a *complement vector* $c = (c_0, c_1, \ldots, c_{n-1})$ of length n. Treating a *source address* x as an n-bit vector, we perform matrix-vector multiplication over $GF(2)$ and then form the corresponding n-bit *target address* y by complementing some subset of the resulting bits: $y = A\,x \oplus c$, or

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1,n-1} \\ a_{20} & a_{21} & a_{22} & \cdots & a_{2,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix} \oplus \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix}.$$

Because we require the characteristic matrix A to be nonsingular, the mapping of source addresses to target addresses is one-to-one. (This property is a consequence of Lemma 3 in section 2.)

We shall generally focus on the matrix multiplication portion of BMMC permutations rather than on the complement vector. The permutation π_A characterized by

[1] Edelman, Heller, and Johnsson [15] call BMMC permutations *affine transformations* or, if there is no complementing, *linear transformations*.

[2] Matrix multiplication over $GF(2)$ is like standard matrix multiplication over the reals but with all arithmetic performed modulo 2. Equivalently, multiplication is replaced by logical-and, and addition is replaced by exclusive-or.

a matrix A is the permutation for which $\pi_A(x) = A\,x$ for all source addresses x.

The following lemma shows the equivalence of multiplying characteristic matrices and composing permutations when the complement vectors are zero. For permutations π_Y and π_Z, the *composition* $\pi_Z \circ \pi_Y$ is defined by $(\pi_Z \circ \pi_Y)(x) = \pi_Z(\pi_Y(x))$ for all x in the domain of π_Y.

LEMMA 1. *Let Z and Y be nonsingular $n \times n$ matrices and let π_Z and π_Y be the permutations characterized by Z and Y, respectively. Then the matrix product $Z\,Y$ characterizes the composition $\pi_Z \circ \pi_Y$.*

Proof. For any source address x, we have

$$\begin{aligned}(\pi_Z \circ \pi_Y)(x) &= \pi_Z(\pi_Y(x)) \\ &= \pi_Z(Y\,x) \\ &= Z(Y\,x) \\ &= (Z\,Y)x\ ,\end{aligned}$$

and so the matrix product $Z\,Y$ characterizes the composition $\pi_Z \circ \pi_Y$. ∎

When we factor a characteristic matrix A into the product of several nonsingular matrices, each factor characterizes a BMMC permutation. The following corollary describes the order in which we perform these permutations to effect the permutation characterized by A.

COROLLARY 2. *Let the $n \times n$ characteristic matrix A be factored as $A = A^{(k)}\,A^{(k-1)}\,A^{(k-2)} \cdots A^{(1)}$, where each factor $A^{(i)}$ is a nonsingular $n \times n$ matrix. Then we can perform the BMMC permutation characterized by A by performing, in order, the BMMC permutations characterized by $A^{(1)}, A^{(2)}, \ldots, A^{(k)}$. That is, we perform the permutations characterized by the factors of a matrix from right to left.*

Proof. The proof is a simple induction, using Lemma 1. ∎

The BMMC algorithm in [10] exploits Corollary 2 to factor a characteristic matrix into a product of other characteristic matrices, performing the permutations given by the factors right to left. It uses

$$\frac{2N}{BD}\left(2\left\lceil\frac{\lg M - r}{\lg(M/B)}\right\rceil + H(N,M,B)\right)$$

parallel I/Os, where r is the rank of the leading $\lg M \times \lg M$ submatrix of the characteristic matrix and

$$H(N,M,B) = \begin{cases} 4\left\lceil\dfrac{\lg B}{\lg(M/B)}\right\rceil + 9 & \text{if } M \le \sqrt{N}\ , \\ 4\left\lceil\dfrac{\lg(N/B)}{\lg(M/B)}\right\rceil + 1 & \text{if } \sqrt{N} < M < \sqrt{NB}\ , \\ 5 & \text{if } \sqrt{NB} \le M\ . \end{cases} \tag{1}$$

One can adapt the lower bound proven in this paper to show that $\Omega\left(\frac{N}{BD}\frac{\lg M - r}{\lg(M/B)}\right)$ parallel I/Os are necessary (see section 2.8 of [9]), but so far it has been unknown whether the $\Theta\left(\frac{N}{BD}\,H(N,M,B)\right)$ term is necessary in all cases. This paper shows that it is not.

BPC permutations. By restricting the characteristic matrix A of a BMMC permutation to be a permutation matrix—having exactly one 1 in each row and each

column—we obtain the class of *bit-permute/complement*, or *BPC*, permutations.[3] One can think of a BPC permutation as forming each target address by applying a fixed permutation to the source-address bits and then complementing a subset of the resulting bits. The class of BPC permutations includes many common permutations such as matrix transposition (when dimensions are powers of 2), bit-reversal permutations (used in performing FFTs), vector-reversal permutations, hypercube permutations, and matrix reblocking.

Previous work [10] expressed the I/O complexity of BPC permutations in terms of cross-ranks. For any $n \times n$ permutation matrix A and for any $k = 0, 1, \ldots, n-1$, the *k-cross-rank* of A is

$$\rho_k(A) = \operatorname{rank} A_{k..n-1,0..k-1} = \operatorname{rank} A_{0..k-1,k..n-1} \ , \tag{2}$$

where, for example, $A_{k..n-1,0..k-1}$ denotes the submatrix of A consisting of the intersection of rows $k, k+1, \ldots, n-1$ and columns $0, 1, \ldots, k-1$. The *cross-rank* of A is the maximum of the b- and m-cross-ranks:

$$\rho(A) = \max(\rho_b(A), \rho_m(A)) \ . \tag{3}$$

The BPC algorithm in [10] uses at most

$$\frac{2N}{BD}\left(2\left\lceil\frac{\rho(A)}{\lg(M/B)}\right\rceil + 1\right)$$

parallel I/Os. One can adapt the lower bound we prove in section 3 for BMMC permutations to show that this BPC algorithm is asymptotically optimal. The BMMC algorithm in section 6, however, is asymptotically optimal for all BMMC permutations—including those that are BPC—and it reduces the innermost factor of 2 in the above bound to a factor of 1. Not only is the BPC algorithm in [10] improved upon by the results in this paper, but the notion of cross-rank appears to be obviated as well.

MRC permutations. *Memory-rearrangement/complement*, or *MRC*, permutations are BMMC permutations with the additional restrictions shown in Table 1: both the leading $m \times m$ and trailing $(n-m) \times (n-m)$ submatrices of the characteristic matrix are nonsingular, the upper right $m \times (n-m)$ submatrix can contain any 0-1 values at all, and the lower left $(n-m) \times m$ submatrix is all 0. Cormen [10] shows that any MRC permutation requires only one pass of N/BD parallel reads and N/BD parallel writes. If we partition the N records into N/M consecutive sets of M records each, we call each set a *memoryload*. Each memoryload consists of M/BD consecutive stripes in which all addresses have the same value in the most significant $n-m$ bits, as Fig. 2 shows. Any MRC permutation can be performed by reading in a memoryload, permuting its records in memory, and writing them out to a (possibly different) memoryload number. Because a memoryload may be read and written with striped I/Os, any MRC permutation may be performed with striped reads and striped writes. The class of MRC permutations includes those characterized by unit upper-triangular matrices. As [10] shows, both the standard binary-reflected Gray code and its inverse have characteristic matrices of this form, and so they are MRC permutations.

[3] Johnsson and Ho [19] call BPC permutations *dimension permutations*, and Aggarwal, Chandra, and Snir [1] call BPC permutations without complementing *rational permutations*.

MLD permutations. We define here a new BMMC permutation subclass, which we shall use in our asymptotically optimal BMMC algorithm. To define this subclass, we first need the standard linear-algebraic notion of a kernel. The *kernel* of any $p \times q$ matrix A is the set of q-vectors that map to 0 when multiplied by A. That is,

$$\ker A = \{x : Ax = 0\} .$$

An MLD permutation has a characteristic matrix that is nonsingular and of the following form:

$$\begin{array}{c} \begin{array}{cc} \multicolumn{1}{c}{m} & \multicolumn{1}{c}{n-m} \end{array} \\ \left[\begin{array}{c|c} \text{arbitrary} & \\ \hline \lambda & \text{arbitrary} \\ \hline \mu & \end{array}\right] \end{array} \begin{array}{l} b \\ m-b \\ n-m \end{array} ,$$

subject to the *kernel condition*

$$\ker \lambda \subseteq \ker \mu; \tag{4}$$

or, equivalently, $\lambda x = 0$ implies $\mu x = 0$.

As we shall see in section 4, the kernel condition implies that we can perform any MLD permutation in one pass by reading in each source memoryload, permuting its records in memory, and writing these records out to M/BD blocks on each disk. Although the blocks read from each memoryload must come from M/BD consecutive stripes, the blocks written may go to any locations at all, as long as M/BD blocks are written to each disk. That is, MLD permutations use striped reads and independent writes. We shall also see in section 4 that we can perform the inverse of an MLD permutation in one pass with independent reads and striped writes.

Outline. The remainder of this paper is organized as follows. Section 2 reviews some fundamental linear-algebraic notions and proves some properties that we shall use in later sections. Section 3 states and proves the lower bound for BMMC permutations. Section 4 shows how to perform any MLD permutation in one pass and gives some additional properties of MLD permutations and their inverses. Section 5 previews several of the matrix forms used in section 6, which presents an algorithm for BMMC permutations whose I/O complexity asymptotically matches the lower bound. Section 7 shows how to detect at run time whether a vector of target addresses describes a BMMC permutation, thus enabling us to determine whether the BMMC algorithm is applicable; this section also presents an easy method for determining whether a nonsingular matrix satisfies the kernel condition (4) and therefore characterizes an MLD permutation. Finally, section 8 contains some concluding remarks.

The algorithms for MLD and BMMC permutations in sections 4 and 6 take little computation time and space. (They do, however, require permutations to be performed in memory, and various architectures may differ in how efficiently they do so.) The data structures are vectors of length $\lg N$ or matrices of size at most $\lg N \times \lg N$. Even sequential algorithms for the harder computations (e.g., finding a maximal set of linearly independent columns of a bit matrix) take time polynomial in $\lg N$, in fact, $O(\lg^3 N)$.

We shall not concern ourselves with memory issues when manipulating characteristic matrices. That is, we assume throughout this paper that $\lg^2 N \ll M$, since as a practical matter, the size of any characteristic matrix is much smaller than memory.

Consider, for example, a problem with $N = 2^{60}$ records, or about one quintillion. (This problem is much larger than any problem that one is likely to see for a long time. If each record were only one byte long, such a data set would occupy one billion gigabytes.) A 60×60 characteristic matrix for a problem this large would require 3600 bits, or 120 words of 32 bits if each column is packed into two words. This amount is insignificant compared to memory sizes of even modest computer systems. Consequently, we shall think of the M-record memory as holding only records and not the characteristic matrix or complement vector.

We shall use several notational conventions in this paper, as in equation (2). Matrix row and column numbers are indexed from 0 starting from the upper left. Vectors are indexed from 0, too. We index rows and columns by sets to indicate submatrices, using ".." notation to indicate sets of contiguous numbers. When a matrix is indexed by just one set rather than two, the set indexes column numbers; the submatrix consists of entire columns. When a submatrix index is a singleton set, we shall often omit the enclosing braces. We denote an identity matrix by I and a matrix whose entries are all 0s by 0; the dimensions of such matrices will be clear from their contexts. All matrix and vector elements are drawn from $\{0, 1\}$, and all matrix and vector arithmetic is over $GF(2)$. When convenient, we interpret bit vectors as the integers they represent in binary. Vectors are treated as 1-column matrices in context.

Some readers familiar with linear algebra may notice that a few of the lemmas in this paper are special cases of standard linear-algebra properties restricted to $GF(2)$. We include the proofs here for completeness.

2. Linear-algebraic fundamentals. This section reviews some standard linear-algebraic terms and proves a few simple properties that we shall use later on. It also shows how to find a maximal set of linearly independent columns of a bit matrix.

Ranges and preimages. For a $p \times q$ matrix A with 0-1 entries, we define the *range* of A by

$$\mathcal{R}(A) = \{y : y = Ax \text{ for some } x \in \{0, 1, \ldots, 2^q - 1\}\} ;$$

that is, $\mathcal{R}(A)$ is the set of p-vectors that can be produced by multiplying all q-vectors with 0-1 entries (interpreted as integers in $\{0, 1, \ldots, 2^q - 1\}$) by A over $GF(2)$. We also adopt the notation

$$\mathcal{R}(A) \oplus c = \{z : z = y \oplus c \text{ for some } y \in \mathcal{R}(A)\} ;$$

that is, $\mathcal{R}(A) \oplus c$ is the exclusive-or of the range of A and a fixed vector c.

LEMMA 3. *Let A be a $p \times q$ matrix whose entries are drawn from $\{0, 1\}$, let c be any p-vector whose entries are drawn from $\{0, 1\}$, and let $r = \operatorname{rank} A$. Then $|\mathcal{R}(A) \oplus c| = 2^r$.*

Proof. Let S index a maximal set of linearly independent columns of A, so that $S \subseteq \{0, 1, \ldots, q - 1\}$, $|S| = r$, the columns of the submatrix A_S are linearly independent, and for any column number $j \notin S$, the column A_j is linearly dependent on the columns of A_S. We claim that $\mathcal{R}(A) = \mathcal{R}(A_S)$. Clearly, $\mathcal{R}(A_S) \subseteq \mathcal{R}(A)$, since $\mathcal{R}(A)$ includes the sum (over $GF(2)$) of each subset of columns of A. To see that $\mathcal{R}(A) \subseteq \mathcal{R}(A_S)$, consider any q-vector $y \in \mathcal{R}(A)$. There is some set T of column indices such that $y = \bigoplus_{j \in T} A_j$. For each column index $j \in T - S$, let $S_j \subseteq S$ index

the columns of A_S that A_j depends on: $A_j = \bigoplus_{k \in S_j} A_k$. Then we have

$$\begin{aligned} y &= \bigoplus_{j \in T} A_j \\ &= \left(\bigoplus_{j \in T \cap S} A_j \right) \oplus \left(\bigoplus_{j \in T - S} A_j \right) \\ &= \left(\bigoplus_{j \in T \cap S} A_j \right) \oplus \left(\bigoplus_{j \in T - S} \left(\bigoplus_{k \in S_j} A_k \right) \right) , \end{aligned}$$

and so y is a linear combination of columns of A_S. Thus, $y \in \mathcal{R}(A_S)$, which in turn proves that $\mathcal{R}(A) \subseteq \mathcal{R}(A_S)$, and consequently $\mathcal{R}(A) = \mathcal{R}(A_S)$.

We have $|\mathcal{R}(A_S)| = 2^{|S|} = 2^r$, since each vector in $\mathcal{R}(A_S)$ is the sum of a unique subset of the columns of S and each column index in S may or may not be included in a sum of the columns. Thus, $|\mathcal{R}(A)| = 2^r$. Exclusive-oring the result of the matrix multiplication by a constant p-vector does not change the cardinality of the range. Therefore, $|\mathcal{R}(A) \oplus c| = |\mathcal{R}(A)| = 2^r$. □

For a $p \times q$ matrix A and a p-vector $y \in \mathcal{R}(A)$, we define the *preimage* of y under A by

$$\mathrm{Pre}(A, y) = \{x : Ax = y\} .$$

That is, $\mathrm{Pre}(A, y)$ is the set of q-vectors x that map to y when multiplied by A.

LEMMA 4. *Let A be a $p \times q$ matrix whose entries are drawn from $\{0, 1\}$, let y be any p-vector in $\mathcal{R}(A)$, and let $r = \operatorname{rank} A$. Then $|\mathrm{Pre}(A, y)| = 2^{q-r}$.*

Proof. Let S index a maximal set of linearly independent columns of A, so that $S \subseteq \{0, 1, \ldots, q-1\}$, $|S| = r$, the columns of the submatrix A_S are linearly independent, and for any column number $j \notin S$, the column A_j is linearly dependent on the columns of A_S. Let $S' = \{0, 1, \ldots, q-1\} - S$.

We claim that for any value $i \in \{0, 1, \ldots, 2^{q-r} - 1\}$, there is a unique q-vector $x^{(i)}$ for which $x^{(i)}_{S'}$ is the binary representation of i and $y = A\,x^{(i)}$. Why? We have $y = A_S\, x^{(i)}_S \oplus A_{S'}\, x^{(i)}_{S'}$ or, equivalently,

$$y \oplus A_{S'} x^{(i)}_{S'} = A_S\, x^{(i)}_S \ . \tag{5}$$

The columns of A_S span $\mathcal{R}(A)$, which implies that for all $z \in \mathcal{R}(A)$, there is a unique r-vector w such that $z = A_S\, w$. Letting $z = y \oplus A_{S'}\, x^{(i)}_{S'}$, we see that there is a unique r-vector $x^{(i)}_S$ that satisfies equation (5), which proves the claim.

Thus, we have shown that $|\mathrm{Pre}(A, y)| \geq 2^{q-r}$. If we had $|\mathrm{Pre}(A, y)| > 2^{q-r}$, then because y is arbitrarily chosen from $\mathcal{R}(A)$, we would have that $\sum_{y' \in \mathcal{R}(A)} |\mathrm{Pre}(A, y')| > 2^q$. But this inequality contradicts there being only 2^q possible preimage vectors. We conclude that $|\mathrm{Pre}(A, y)| = 2^{q-r}$. □

Row spaces. The *row space* of a matrix A, written $\operatorname{row} A$, is the span of the rows of A. We prove the following lemma about the relationship between kernels and row spaces, which we shall use later to prove properties resulting from the kernel condition of MLD permutations and to check that the kernel condition holds.

LEMMA 5. *Let K and L be q-column matrices. Then $\ker K \subseteq \ker L$ if and only if $\operatorname{row} L \subseteq \operatorname{row} K$.*

Proof. For any vector space X, the *orthogonal space* of X, written $X^\perp$, is the set of vectors Y such that for all $x \in X$ and all $y \in Y$, the inner product $x \cdot y$ is 0. We use the following well-known facts from linear algebra (see Strang [23, pp. 138–139], for example).

1. The row space and the kernel are orthogonal spaces of each other. Thus, $(\operatorname{row} K)^\perp = \ker K$ and $(\operatorname{row} L)^\perp = \ker L$.
2. For any vector spaces X and Y, $X \subseteq Y$ implies $Y^\perp \subseteq X^\perp$.
3. For any vector space X, $(X^\perp)^\perp = X$.[4]

The latter two properties imply that if $Y^\perp \subseteq X^\perp$, then $X \subseteq Y$. Thus we have

$$\begin{aligned} \ker K \subseteq \ker L \ &\text{iff}\ (\operatorname{row} K)^\perp \subseteq (\operatorname{row} L)^\perp \\ &\text{iff}\ \operatorname{row} L \subseteq \operatorname{row} K\ , \end{aligned}$$

which proves the lemma. □

Finding a maximal set of linearly independent columns. We conclude this section with a simple sequential algorithm to find a maximal set S of linearly independent columns of a $p \times q$ matrix K. We shall use this technique several times in this paper.

We use the following pseudocode:

```
1  S ← ∅
2  for each row index i ← 0 to p − 1 do
3     if there exists some column index j for which K_ij = 1
4        then for each column index j′ such that K_ij′ = 1 do
5                 add column j to column j′
6                 S ← S ∪ {j}
```

At the completion of this algorithm, the set S contains the indices for a maximal set of linearly independent columns of K. Lines 4–5 zero out any column in the set. Each iteration of the outer loop zeros out the next row. By the end of the algorithm, every column gets zeroed out as a column in S or by the addition of some subset of columns in S.

This algorithm takes $O(p^2 q)$ time on a sequential machine. In our applications of this algorithm, p and q are at most $\lg N$, and so the sequential time will always be $O(\lg^3 N)$.

3. A universal lower bound for BMMC permutations. In this section, we state and prove the lower bound for BMMC permutations. After stating the lower bound, we briefly discuss its significance before presenting the full proof. The lower bound is given by the following theorem.

THEOREM 6. *Any algorithm that performs a nonidentity BMMC permutation with characteristic matrix A requires*

$$\Omega\left(\frac{N}{BD}\left(1 + \frac{\operatorname{rank}\gamma}{\lg(M/B)}\right)\right)$$

parallel I/Os, where γ is the submatrix $A_{b..n-1,0..b-1}$ of size $\lg(N/B) \times \lg B$. □

This lower bound is *universal* in the sense that it applies to all inputs other than the identity permutation, which of course requires no data movement at all. In

[4] The proof that this property holds over $GF(2)$ is not as straightforward as the conventional proof that it holds over $\mathbb{R}^n$. Lang [20, p. 131] contains a proof for $GF(2)$.

contrast, lower bounds such as the standard $\Omega(N \lg N)$ lower bound for sorting N items on a sequential machine are *existential*: they apply to worst-case inputs, but for some inputs an algorithm may be able to do better.

Section 6 presents an algorithm that achieves the bound given by Theorem 6, and so this algorithm is asymptotically optimal.

Technique. To prove Theorem 6, we rely heavily on the technique used by Aggarwal and Vitter [2] to prove a lower bound on I/Os for matrix transposition; their proof is based in turn on a method by Floyd [16]. We prove the lower bound for the case in which $D = 1$; the general case follows by dividing by D. We consider only I/Os that are *simple*. An input is simple if each record read is removed from the disk and moved into an empty location in memory. An output is simple if the records are removed from the memory and written to empty locations on the disk. When all I/Os are simple, exactly one copy of each record exists at any time during the execution of an algorithm. The following lemma, proven by Aggarwal and Vitter, allows us to consider only simple I/Os when proving lower bounds.

LEMMA 7. *For each computation that implements a permutation of records, there is a corresponding computation strategy involving only simple I/Os such that the total number of I/Os is no greater.* □

The basic scheme of the proof of Theorem 6 uses a potential-function argument. *Time* q is the time interval starting when the qth I/O completes and ending just before the $(q+1)$st I/O starts. We define a potential function Φ so that $\Phi(q)$ is the *potential* at time q. This potential measures how close the current record ordering is to the desired permutation order. Higher potentials indicate that the current ordering is closer to the desired permutation. We compute the initial and final potentials and bound the amount that the potential can increase in each I/O operation. The lower bound then follows.

To be more precise, we start with some definitions. For $i = 0, 1, \ldots, N/B - 1$, we define the ith *target group* to be the set of records that belong in block i according to the given BMMC permutation. We denote by $g_{\text{block}}(i, k, q)$ the number of records in the ith target group that are in block k on disk at time q, and $g_{\text{mem}}(i, q)$ denotes the number of records in the ith target group that are in memory at time q. We define the continuous function

$$f(x) = \begin{cases} x \lg x & \text{if } x > 0 \ , \\ 0 & \text{if } x = 0 \ , \end{cases}$$

and we define *togetherness functions*

$$G_{\text{block}}(k, q) = \sum_{i=0}^{N/B-1} f(g_{\text{block}}(i, k, q))$$

for each block k at time q and

$$G_{\text{mem}}(q) = \sum_{i=0}^{N/B-1} f(g_{\text{mem}}(i, q))$$

for memory at time q. Finally, we define the *potential* at time q, denoted $\Phi(q)$, as the sum of the togetherness functions:

$$\Phi(q) = G_{\text{mem}}(q) + \sum_{k=0}^{N/B-1} G_{\text{block}}(k, q) \ .$$

Aggarwal and Vitter embed the following lemmas in their lower-bound argument. The first lemma is based on the observation that the number of parallel I/Os needed is at least the total increase in potential over all parallel I/Os divided by the maximum increase in potential (denoted $\Delta\Phi_{\max}$) in any single parallel I/O.

LEMMA 8. *Let $D = 1$, and consider any algorithm that uses t parallel I/Os to perform a permutation. Then $t = \Omega\left(\frac{\Phi(t)-\Phi(0)}{\Delta\Phi_{\max}}\right)$.* □

LEMMA 9. *Let $D = 1$, and consider any permutation that can be performed with t parallel I/Os. Then $\Phi(t) = N \lg B$ and $\Delta\Phi_{\max} = O(B \lg(M/B))$. Therefore, any algorithm that performs a permutation uses $\Omega\left(\frac{N \lg B - \Phi(0)}{B \lg(M/B)}\right)$ parallel I/Os.* □

Observe that these lemmas imply lower bounds that are universal. No matter what permutation is being performed, the initial potential is $\Phi(0)$, the final potential is $\Phi(t)$, the increase in potential per parallel I/O is at most $\Delta\Phi_{\max}$, and so $\Omega\left(\frac{\Phi(t)-\Phi(0)}{\Delta\Phi_{\max}}\right)$ parallel I/Os are required.

We can now show a trivial lower bound for all nonidentity BMMC permutations.

LEMMA 10. *If $D = 1$, any algorithm that performs a nonidentity BMMC permutation requires $\Omega(N/B)$ parallel I/Os.*

Proof. Consider a BMMC permutation with characteristic matrix A and complement vector c. It is the identity permutation if and only if $A = I$ and $c = 0$, so we shall assume that either $A \neq I$ or $c \neq 0$.

A *fixed point* of the BMMC permutation is a source address x for which

$$A\,x \oplus c = x \ . \tag{6}$$

If a record's source address is not a fixed point, its source block must be read and its target block must be written. We shall show that for any nonidentity BMMC permutation, at least $N/2$ addresses are not fixed points. Even if these records are clustered into as few source blocks as possible, then at least half the source blocks, or $N/2B$, must be read. The lemma then follows.

To show that at least $N/2$ addresses are not fixed points, we shall show that at most $N/2$ addresses are. Rewriting equation (6) as $(A \oplus I)x = c$, we see that we wish to bound the size of $\mathrm{Pre}(A \oplus I, c)$. If $c \notin \mathcal{R}(A \oplus I)$, then this size is 0. Otherwise, by Lemma 4, this size is $2^{n-\mathrm{rank}(A \oplus I)}$. If $A \neq I$, then $\mathrm{rank}(A \oplus I) \geq 1$, which implies that $|\mathrm{Pre}(A \oplus I, c)| \leq 2^{n-1} = N/2$. If $A = I$, then $A \oplus I$ is the 0 matrix, and the only vector in its range is 0. But $A = I$ and $c = 0$ yields the identity permutation, which we specifically disallow. □

Proof of Theorem 6. Recall that we shall prove Theorem 6 by proving the lower bound for the case in which $D = 1$; the general case follows by dividing by D. We work with characteristic matrix A and complement vector c. We assume that all I/Os are simple and transfer exactly B records, some possibly empty. Since all records start on disk and I/Os are simple, memory is initially empty.

We need to compute the initial potential in order to apply Lemma 9. The initial potential depends on the number of records that start in the same source block and are in the same target group. A record with source address $x = (x_0, x_1, \ldots, x_{n-1})$ is in source block k if and only if

$$k = x_{b..n-1} \ , \tag{7}$$

interpreting k as an $(n-b)$-bit binary number with the least signficant bit first. This

record maps to target block i if and only if

$$\begin{aligned} i &= A_{b..n-1,0..n-1}\, x_{0..n-1} \oplus c_{b..n-1} \\ &= A_{b..n-1,0..b-1}\, x_{0..b-1} \oplus A_{b..n-1,b..n-1}\, x_{b..n-1} \oplus c_{b..n-1} \ , \end{aligned} \tag{8}$$

also interpreting i as an $(n-b)$-bit binary number. The following lemma gives the exact number of records that start in each source block and are in the same target group.

LEMMA 11. *Let $r = \operatorname{rank} A_{b..n-1,0..b-1}$, and consider any source block k. There are exactly 2^r distinct target blocks that some record in source block k maps to, and for each such target block, exactly $B/2^r$ records in source block k map to it.*

Proof. For a given source block k, all source addresses fulfill condition (7), and so they map to target block numbers given by condition (8) but with $x_{b..n-1}$ fixed at k. The range of target block numbers is thus $\mathcal{R}(A_{b..n-1,0..b-1}) \oplus (A_{b..n-1,b..n-1}\, k \oplus c_{b..n-1})$ which, by Lemma 3, has cardinality 2^r.

Now we determine the set of source addresses in source block k that map to a particular target block i in $\mathcal{R}(A_{b..n-1,0..b-1}) \oplus (A_{b..n-1,b..n-1}\, k \oplus c_{b..n-1})$. Again fixing $x_{b..n-1} = k$ in condition (8) and exclusive-oring both sides by $A_{b..n-1,b..n-1}\, k \oplus c_{b..n-1}$, we see that this set is precisely $\operatorname{Pre}(A_{b..n-1,0..b-1}, i \oplus A_{b..n-1,b..n-1}\, k \oplus c_{b..n-1})$. By Lemma 4, this set has cardinality exactly 2^{b-r}, which equals $B/2^r$. ∎

We can interpret Lemma 11 as follows. Let $r = \operatorname{rank} A_{b..n-1,0..b-1}$, and consider a particular source block k. Then there are exactly 2^r target blocks i for which $g_{\text{block}}(i,k,0)$ is nonzero, and for each such nonzero target block, we have $g_{\text{block}}(i,k,0) = B/2^r$.

Now we can compute $\Phi(0)$. Since memory is initially empty, $g_{\text{mem}}(i,0) = 0$ for all blocks i, which implies that $G_{\text{mem}}(0) = 0$. We have

$$\begin{aligned} \Phi(0) &= G_{\text{mem}}(0) + \sum_{k=0}^{N/B-1} G_{\text{block}}(k,0) \\ &= 0 + \sum_{k=0}^{N/B-1} \sum_{i=0}^{N/B-1} f(g_{\text{block}}(i,k,0)) \\ &= \sum_{k=0}^{N/B-1} 2^r \frac{B}{2^r} \lg \frac{B}{2^r} \qquad \text{(by Lemma 11)} \\ &= \frac{N}{B} B \lg \frac{B}{2^r} \\ &= N(\lg B - r) \ . \end{aligned} \tag{9}$$

Combining Lemmas 9 and 10 with equation (9), we get a lower bound of

$$\Omega\left(\frac{N}{B} + \frac{N \lg B - N(\lg B - r)}{B \lg(M/B)}\right) = \Omega\left(\frac{N}{B}\left(1 + \frac{\operatorname{rank} A_{b..n-1,0..b-1}}{\lg(M/B)}\right)\right)$$

parallel I/Os. Dividing through by D yields a lower bound of

$$\Omega\left(\frac{N}{BD}\left(1 + \frac{\operatorname{rank} A_{b..n-1,0..b-1}}{\lg(M/B)}\right)\right) \ ,$$

which completes the proof of Theorem 6.

4. MLD permutations. In this section, we describe how to perform any MLD permutation in only one pass. This section also discusses additional properties of MLD and MRC permutations and concludes with a discussion of MLD^{-1} permutations, which are permutations whose inverses are MLD permutations. Section 7 shows how to determine whether a given matrix characterizes an MLD permutation.

How the kernel condition implies a one-pass permutation. We shall show in three steps that the kernel condition implies that, for a given source memoryload, the source records are permuted into full target blocks spread evenly across the disks. To do so, we first need to define the notion of relative block number, as shown in Fig. 2. For a given n-bit record address $x_{0..n-1}$, the *relative block number* of x is the $m-b$ bits $x_{b..m-1}$. The relative block number ranges from 0 to $M/B-1$ and determines the number of a block within its memoryload. Recall that the memoryload number is the $n-m$ bits $x_{m..n-1}$. We shall prove that *for a given source memoryload*, the following properties hold.

1. Its records map to all M/B relative block numbers, and each relative block number has exactly B records mapping to it.
2. Records that map to the same relative block number map to the same target memoryload number as well.

The first two properties imply that the records of each source memoryload map to exactly M/B target blocks and that each such target block is full.

3. These M/B target blocks are distributed evenly among the disks, with M/BD mapping to each disk.

Given these properties, we can perform an MLD permutation in one pass. Like the other one-pass permutations described in [10], we allow the permutation to map records from one set of N/BD stripes (the "source portion" of the parallel disk system) to a different set of N/BD stripes (the "target portion"). One can think of addresses as relative to the beginning of the appropriate portion. In this way, we need not be concerned with overwriting source records before we get a chance to read them. Note that when we chain passes together, as in the BMMC algorithm of section 6 and the BPC algorithm of [10], we can avoid allocating a new target portion in each pass by reversing the roles of the source and target portions between passes, and so the total disk space used is $2N$ records.

We perform an MLD permutation by processing source memoryload numbers from 0 to $N/M-1$. For each source memoryload, we first read into memory its M/BD consecutive stripes from the source portion. We then permute its records in memory, clustering them into M/B full target blocks that are distributed evenly among the disks. We then write out these target blocks using M/BD independent writes to the target portion. After processing all N/M source memoryloads, we have read each record from the source portion and written it to where it belongs in the target portion. Thus, we have performed the MLD permutation in one pass.

The following lemma gives an important consequence of the kernel condition.

LEMMA 12. *If the matrix A characterizes an MLD permutation, then the submatrix λ has rank $m-b$.*

Proof. We shall prove that all rows of the leading $m \times m$ submatrix of A are linearly independent. The lemma then follows because λ is a subset of these rows.

Because A is nonsingular, the rank of its leftmost m columns (i.e., the submatrix $A_{0..n-1,0..m-1}$) is m. The row rank of any matrix equals the column rank, and so there are m linearly independent rows in $A_{0..n-1,0..m-1}$.

Since $\ker \lambda \subseteq \ker \mu$, Lemma 5 implies that $\operatorname{row} \mu \subseteq \operatorname{row} \lambda$. Thus, every row of μ is linearly dependent on some rows of λ and hence on some rows of the leading $m \times m$ submatrix of A. Since there are m linearly independent rows in $A_{0..n-1,0..m-1}$, all rows of the leading $m \times m$ submatrix must be linearly independent. □

We now prove property 1.

LEMMA 13. *The records of each source memoryload in an MLD permutation map to exactly M/B relative block numbers. Moreover, for a given source memoryload, each relative block number has exactly B records mapping to it.*

Proof. Let A characterize an MLD permutation with complement vector c. By Lemma 12, $\operatorname{rank} A_{b..m-1,0..m-1} = m - b$. The target relative block number $y_{b..m-1}$ corresponding to a source address x is given by the equation

$$y_{b..m-1} = A_{b..m-1,0..m-1}\, x_{0..m-1} \oplus A_{b..m-1,m..n-1}\, x_{m..n-1} \oplus c_{b..m-1} \ . \tag{10}$$

The value of $x_{m..n-1}$ is fixed for a given source memoryload, and so the $(m-b)$-vector $A_{b..m-1,m..n-1}\, x_{m..n-1} \oplus c_{b..m-1}$ has the same value for all records. By Lemma 3, $y_{b..m-1}$ takes on $2^{\operatorname{rank} A_{b..m-1,0..m-1}} = 2^{m-b} = M/B$ different values for the M different values of $x_{0..m-1}$. That is, the records of each source memoryload map to exactly M/B different relative block numbers.

Now consider some relative block number $y_{b..m-1}$ that some source address in a memoryload maps to. Using equation (10), the number of source addresses $x_{0..m-1}$ within that memoryload that map to $y_{b..m-1}$ is equal to $|\mathrm{Pre}(A_{b..m-1,0..m-1}, y_{b..m-1} \oplus A_{b..m-1,m..n-1}\, x_{m..n-1} \oplus c_{b..m-1})|$. By Lemma 4, this number is equal to

$$2^{m - \operatorname{rank} A_{b..m-1,0..m-1}} = 2^{m-(m-b)} = B \ . \qquad \square$$

Property 2 follows directly from the kernel condition. Although we use kernel notation for its simplicity of expression, the following lemma shows that the kernel condition is equivalent to requiring that, for a given source memoryload, every record destined for a particular relative block number must also be destined for the same target memoryload.

LEMMA 14. *Let K and L be matrices with q columns. Then $\ker K \subseteq \ker L$ if and only if for all q-vectors x and y, $Kx = Ky$ implies $Lx = Ly$.*

Proof. Suppose that $\ker K \subseteq \ker L$ and $Kx = Ky$. Then $K(x \oplus y) = 0$, which implies that $L(x \oplus y) = 0$, which in turn implies $Lx = Ly$.

Conversely, suppose that $Kx = Ky$ implies $Lx = Ly$ for all q-vectors x and y, and consider any q-vector $z \in \ker K$. We have $Kz = 0 = K \cdot 0$, which implies $Lz = L \cdot 0 = 0$. Thus, $z \in \ker L$. □

For an MLD permutation, since $\ker \lambda \subseteq \ker \mu$, we apply Lemma 14 with $K = \lambda$ and $L = \mu$. Thus, any two source records x and y from the same source memoryload that are mapped to relative block number $\lambda\, x_{0..m-1}$ are also mapped to the same target memoryload $\mu\, x_{0..m-1}$.

Property 3 follows from property 1. Each source memoryload maps to relative block numbers $0, 1, \ldots, M/B-1$. As Fig. 2 shows, the number of the disk that a block resides on is encoded in the least significant d bits of its relative block number. The M/B relative block numbers, therefore, are evenly distributed among the D disks, with M/BD residing on each disk.

Thus, we have the following theorem.

THEOREM 15. *Any MLD permutation can be performed in one pass with striped reads and independent writes.*

Proof. The above argument demonstrates that we can perform any MLD permutation in one pass with independent writes. Because a memoryload can be read with M/BD striped reads and the above method for performing MLD permutations reads full memoryloads, it uses striped reads. ∎

Additional properties of MLD and MRC permutations. We now examine some additional properties of MLD permutations. We shall use these properties primarily to combine matrix factors in the BMMC algorithm, thus reducing the number of passes. The first property bounds the rank of the submatrix μ as another consequence of the kernel condition.

LEMMA 16. *In the characteristic matrix for an MLD permutation, the submatrix μ has rank at most $m - b$.*

Proof. By Lemma 5 and the kernel condition, $\operatorname{row}\mu \subseteq \operatorname{row}\lambda$, which in turn implies that $\dim(\operatorname{row}\mu) \le \dim(\operatorname{row}\lambda)$, where the dimension of a vector space is the size of any basis for it. Applying Lemma 12, we have that $\operatorname{rank}\mu \le \operatorname{rank}\lambda = m - b$. ∎

Thus, if the lower left $(n-m) \times m$ submatrix of a characteristic matrix has rank more than $m - b$, the matrix cannot characterize an MLD permutation.

THEOREM 17. *Let the matrix Y characterize an MLD permutation, and let the matrix X characterize an MRC permutation. Then the matrix product YX characterizes an MLD permutation.*

Proof. Write the nonsingular matrix Y as

$$Y = \begin{array}{c} \begin{matrix} m & n-m \end{matrix} \\ \left[\begin{array}{c|c} \alpha & \beta \\ \hline \gamma & \delta \end{array}\right] \begin{matrix} m \\ n-m \end{matrix} \end{array},$$

where

$$\ker \alpha_{b..m-1,0..m-1} \subseteq \ker \gamma \ . \tag{11}$$

Write the nonsingular matrix X as

$$X = \begin{array}{c} \begin{matrix} m & n-m \end{matrix} \\ \left[\begin{array}{c|c} \phi & \sigma \\ \hline 0 & \nu \end{array}\right] \begin{matrix} m \\ n-m \end{matrix} \end{array},$$

where ϕ and ν are nonsingular. We now show that the product

$$YX = \begin{array}{c} \begin{matrix} m & n-m \end{matrix} \\ \left[\begin{array}{c|c} \alpha\phi & \alpha\sigma \oplus \beta\nu \\ \hline \gamma\phi & \gamma\sigma \oplus \delta\nu \end{array}\right] \begin{matrix} m \\ n-m \end{matrix} \end{array}$$

characterizes an MLD permutation. Observe that the product YX is nonsingular because Y and X are each nonsingular.

We must also prove that the kernel condition (4) holds for the product, i.e., that $\ker(\alpha\phi)_{b..m-1,0..m-1} \subseteq \ker(\gamma\phi)$. For an $m \times m$ matrix τ, note that $\tau_{b..m-1,0..m-1} = I_{b..m-1,0..m-1}\,\tau$, where I is the usual $m \times m$ identity matrix. We have that $x \in \ker(\alpha\phi)_{b..m-1,0..m-1}$ implies $(I_{b..m-1,0..m-1}\alpha\phi)x = 0$ (taking $\alpha\phi$ as τ), which in turn implies that $\phi x \in \ker(I_{b..m-1,0..m-1}\alpha) = \ker \alpha_{b..m-1,0..m-1} \subseteq \ker\gamma$, by property (11). Thus, $\gamma\phi x = 0$, and so $x \in \ker(\gamma\phi)$. We conclude that $\ker(\alpha\phi)_{b..m-1,0..m-1} \subseteq \ker(\gamma\phi)$, which completes the proof. ∎

Theorem 17 shows that the composition of an MLD permutation with an MRC permutation is an MLD permutation. Since we have seen how to perform MLD permutations, we can gain an intuitive understanding of why Theorem 17 holds. An MRC permutation permutes memoryload numbers, with records that start together within a source memoryload remaining together in a target memoryload. We perform an MLD permutation by reading in entire memoryloads. Thus, to perform the composition as an MLD permutation, we only have to remap the source memoryload numbers and adjust the in-memory permutations accordingly. Furthermore, as the following lemma shows, the composition of two MRC permutations is merely the composition of their memoryload mappings with the in-memory permutations adjusted accordingly.

THEOREM 18. *The class of MRC permutations is closed under composition and inversion. That is, if a matrix A characterizes an MRC permutation, then so does the matrix A^{-1}, and if $A^{(1)}$ and $A^{(2)}$ characterize MRC permutations, then so does the product $A^{(1)}\,A^{(2)}$.*

Proof. We first show that MRC permutations are closed under inverse. Let the matrix

$$A = \left[\begin{array}{c|c} \overset{m}{\alpha} & \overset{n-m}{\beta} \\ \hline 0 & \delta \end{array}\right] \begin{matrix} m \\ n-m \end{matrix}$$

characterize an MRC permutation, so that the leading submatrix α and trailing submatrix δ are nonsingular. The inverse of this matrix is

$$A^{-1} = \left[\begin{array}{c|c} \overset{m}{\alpha^{-1}} & \overset{n-m}{\alpha^{-1}\beta\delta^{-1}} \\ \hline 0 & \delta^{-1} \end{array}\right] \begin{matrix} m \\ n-m \end{matrix},$$

where the leading $m \times m$ submatrix α^{-1} and trailing $(n-m) \times (n-m)$ submatrix δ^{-1} are nonsingular. Thus, the matrix A^{-1} characterizes an MRC permutation.

We now show that MRC permutations are closed under composition. Consider MRC characteristic matrices

$$A^{(1)} = \left[\begin{array}{c|c} \overset{m}{\alpha^{(1)}} & \overset{n-m}{\beta^{(1)}} \\ \hline 0 & \delta^{(1)} \end{array}\right] \begin{matrix} m \\ n-m \end{matrix},$$

$$A^{(2)} = \left[\begin{array}{c|c} \overset{m}{\alpha^{(2)}} & \overset{n-m}{\beta^{(2)}} \\ \hline 0 & \delta^{(2)} \end{array}\right] \begin{matrix} m \\ n-m \end{matrix},$$

where the submatrices $\alpha^{(1)}$, $\alpha^{(2)}$, $\delta^{(1)}$, and $\delta^{(2)}$ are nonsingular. Then their product is

$$A^{(1)}\,A^{(2)} = \left[\begin{array}{c|c} \overset{m}{\alpha^{(1)}\,\alpha^{(2)}} & \overset{n-m}{\alpha^{(1)}\,\beta^{(2)} \oplus \beta^{(1)}\,\delta^{(2)}} \\ \hline 0 & \delta^{(1)}\,\delta^{(2)} \end{array}\right] \begin{matrix} m \\ n-m \end{matrix}.$$

Because $\alpha^{(1)}$ and $\alpha^{(2)}$ are nonsingular, so is their product $\alpha^{(1)}\,\alpha^{(2)}$. Similarly, the product $\delta^{(1)}\,\delta^{(2)}$ is nonsingular. The product $A^{(1)}\,A^{(2)}$, therefore, characterizes an MRC permutation. □

On the other hand, the composition of two MLD permutations is not necessarily an MLD permutation. We can see this fact in two ways. First, since we perform an MLD permutation by reading in entire memoryloads but writing blocks independently, it may not be possible to remap the source memoryload numbers. Second, consider the product of two matrices, each of which characterizes an MLD permutation. Although the rank of the lower left $(n-m) \times m$ submatrix of each factor is at most $m-b$, it may be the case that the rank of the lower left $(n-m) \times m$ submatrix of the product exceeds $m-b$. If so, then by Lemma 16, the product cannot characterize an MLD permutation.

Moreover, the composition of an MRC permutation with an MLD permutation (that is, reversing the order of the factors in Theorem 17) is not necessarily an MLD permutation. A simple example is the product

$$\begin{array}{ccc} \begin{array}{c} \scriptstyle b \quad m-b \quad n-m \\ \left[\begin{array}{c|c|c} 0 & I & 0 \\ \hline I & 0 & 0 \\ \hline 0 & 0 & I \end{array}\right] \end{array} & \begin{array}{c} \scriptstyle b \quad m-b \quad n-m \\ \left[\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & I & 0 \\ \hline 0 & I & I \end{array}\right] \end{array} & = \begin{array}{c} \scriptstyle b \quad m-b \quad n-m \\ \left[\begin{array}{c|c|c} 0 & I & 0 \\ \hline I & 0 & 0 \\ \hline 0 & I & I \end{array}\right] \end{array} \begin{array}{l} \scriptstyle b \\ \scriptstyle m-b \\ \scriptstyle n-m \end{array} , \\ \text{MRC} & \text{MLD} & \text{not MLD} \end{array}$$

with $b = m - b = n - m$. This product is not MLD since an m-vector with 0s in the first b positions and 1s in the last $m-b$ positions is a vector in $\ker \lambda$, but it is not a vector in $\ker \mu$.

Finally, we note that any MRC permutation is an MLD permutation. Observe that the lower left $(n-m) \times m$ submatrix of an MRC permutation must be 0, which implies that its kernel is the set of all m-vectors. No matter what $\ker \lambda$ is, it is a subset of this set.

Inverses of MLD permutations. The first BMMC algorithm we shall see works by factoring the BMMC characteristic matrix into matrices that characterize MRC and MLD permutations. In some settings, it may be easier to perform a permutation whose inverse is MLD (we call this class MLD^{-1}) than to perform an MLD permutation. We shall see an alternative way to factor BMMC characteristic matrices—into MRC and MLD^{-1} characteristic matrices—so that the resulting algorithm takes the same number of parallel I/Os as the original factorization into MRC and MLD permutations. In the remainder of this section, we examine the properties of MLD^{-1} permutations that enable this alternative factorization.

Striped writes may be valuable when redundant data is maintained on a parallel disk system to reduce the chance of data loss due to a failed device. Many common parallel-disk organizations fall under the heading of RAID (redundant array of inexpensive disks) [7, 17], which is organized into "levels" of redundancy. In RAID levels 3 and 4, an additional disk is added to the disk array to store redundancy. Each block of this *parity disk* contains the bitwise exclusive-or of the contents of the corresponding blocks of the other D data disks. If any one data disk fails, its contents are easily reconstructed from the contents of the $D-1$ remaining data disks and the parity disk. If an entire stripe is written to the disk array, it is easy to compute the corresponding parity information at the same time and write it to the parity disk in parallel with the data being written to the data disks. On the other hand, when less than a full stripe of data is being written to a given stripe of the disk array, updating the parity disk is harder. For each partial stripe being written, the old data and parity information must be read and the new data and parity information must be written. If k different

stripes are being written, accessing the parity disk may become a severe bottleneck since k different blocks of the parity disk must be read and rewritten. Because an independent write may update individual blocks in several different stripes, in a RAID 3 or 4 organization, algorithms that use striped writes are preferable to those that use independent writes.

With this motivation, we begin our investigation of MLD^{-1} permutations with a property that pertains to all one-pass permutations.

LEMMA 19. *If a permutation Π is a one-pass permutation, then its inverse permutation Π^{-1} is also a one-pass permutation. Moreover, if we perform Π using striped reads (respectively, writes), then we can perform Π^{-1} using striped writes (respectively, reads).*

Proof. Consider a one-pass algorithm to perform the permutation Π. It repeatedly reads a set of blocks, permutes their records in memory, and writes the records as full blocks. The one-pass algorithm reads and writes each record once. To perform the inverse permutation Π^{-1}, we invert each read-permute-write step in the algorithm for Π. In each step, we read the blocks that were written in the corresponding step for Π, we perform the inverse in-memory permutation, and we write the blocks that were read in the corresponding step for Π. Each record is still read and written once, and thus Π^{-1} is also a one-pass permutation. Note that if a read (respectively, write) for Π is striped, then the corresponding write (respectively, read) for Π^{-1} is also striped. □

The following corollary follows directly from Theorem 15 and Lemma 19.

COROLLARY 20. *Any MLD^{-1} permutation can be performed in one pass with independent reads and striped writes.* □

Our final property of MLD^{-1} permutations is analogous to Theorem 17.

LEMMA 21. *Let the matrix X characterize an MRC permutation, and let the matrix Y characterize an MLD^{-1} permutation. Then the matrix product XY characterizes an MLD^{-1} permutation.*

Proof. Let $Z = XY$, so that $Z^{-1} = Y^{-1}X^{-1}$. Since the matrix Y characterizes an MLD^{-1} permutation, the matrix Y^{-1} characterizes an MLD permutation. By Theorem 18, the matrix X^{-1} characterizes an MRC permutation. By Theorem 17, therefore, the matrix Z^{-1} characterizes an MLD permutation. We conclude that the matrix Z characterizes an MLD^{-1} permutation. □

5. Matrix-column operations. In this section, we classify forms of matrices which, as factors, have the effect of adding columns of other matrices to yield a product. We shall use matrices of this form in section 6 to transform the characteristic matrix for any BMMC permutation into a characteristic matrix for an MRC permutation. This section shows the structure and useful properties of specific characteristic matrix forms we shall use.

Column additions. We define a *column-addition* matrix as a matrix Q such that the product $A' = AQ$ is a modified form of A in which specified columns of A have been added into others. Denoting the kth column of A by A_k, we define the matrix $Q = (q_{ij})$ by

$$q_{ij} = \begin{cases} 1 & \text{if } i = j\ , \\ 1 & \text{if column } A_i \text{ is added into column } A_j\ , \\ 0 & \text{otherwise}\ . \end{cases}$$

For example,

$$\underbrace{\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}}_{A} \quad \underbrace{\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}}_{Q} = \left[\begin{array}{c|c|c|c} A_0 & A_0 \oplus A_1 \oplus A_3 & A_0 \oplus A_2 & A_3 \end{array}\right]$$

$$= \underbrace{\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{A'} .$$

Column-addition matrices are also subject to a *dependency restriction* that if column i is added into column j, then column j cannot be added into any other column. That is, if $q_{ij} = 1$, then $q_{jk} = 0$ for all $k \neq j$. The following lemma shows that any column-addition matrix is the product of two nonsingular matrices, and so any column-addition matrix is also nonsingular.

LEMMA 22. *Any column-addition matrix is nonsingular.*

Proof. We shall prove by induction on the matrix size n that any column-addition matrix Q is the product of two nonsingular matrices L and U. Thus, the matrix Q is also nonsingular.

For the basis, when $n = 2$, the only column-addition matrices are $\left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$, $\left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right]$, and $\left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$. Since each of these matrices is nonsingular, each of them is the product of itself and the identity matrix.

For the inductive step, we assume that every $(n-1) \times (n-1)$ column-addition matrix is the product of two nonsingular matrices. We partition an arbitrary $n \times n$ column-addition matrix Q as

$$Q = \begin{array}{c} \begin{array}{cc} 1 & n-1 \end{array} \\ \left[\begin{array}{c|c} 1 & \psi \\ \hline \theta & \chi \end{array}\right] \begin{array}{c} 1 \\ n-1 \end{array} \end{array} .$$

The trailing $(n-1) \times (n-1)$ submatrix χ is a column-addition matrix because all of its diagonal elements are 1s and, as a submatrix of Q, it obeys the dependency restriction. By our inductive hypothesis, therefore, the submatrix χ is the product of two $(n-1) \times (n-1)$ nonsingular matrices, say ν and η. By the dependency restriction, if there are any 1s in θ, then there cannot be any 1s in ψ. Therefore, either ψ or θ is a zero submatrix, and consequently we can factor Q as

$$Q = \underbrace{\begin{array}{c} \begin{array}{cc} 1 & n-1 \end{array} \\ \left[\begin{array}{c|c} 1 & 0 \\ \hline \theta & \nu \end{array}\right] \end{array}}_{L} \quad \underbrace{\begin{array}{c} \begin{array}{cc} 1 & n-1 \end{array} \\ \left[\begin{array}{c|c} 1 & \psi \\ \hline 0 & \eta \end{array}\right] \begin{array}{c} 1 \\ n-1 \end{array} \end{array}}_{U} .$$

The rightmost $n-1$ columns of L are linearly independent since the submatrix ν is nonsingular and the upper right $1 \times (n-1)$ submatrix is 0. The leftmost column is linearly independent of the rightmost $n-1$ columns since its top entry is 1 and the top entry of each of the rightmost $n-1$ columns is 0. Thus, L is nonsingular. Similarly,

because the submatrix η is nonsingular and the lower left $(n-1) \times 1$ submatrix of U is 0, the matrix U is nonsingular. Thus, any column-addition matrix is the product of two nonsingular matrices, and therefore is also nonsingular. □

In fact, the factors L and U in the proof of Lemma 22 are unit lower-triangular and unit upper-triangular matrices, respectively. Thus, we can factor the example above as

$$Q = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = L\,U\ .$$

Partitioning the matrix. In section 6, we shall factor nonsingular matrices into column-addition matrices and matrices that characterize MRC permutations. These matrices will be of various block forms, and to classify these forms, we use the following block representation. We partition a matrix into three sections: left, middle, and right. The *left section* includes the leftmost b columns, the *middle section* includes the middle $m-b$ columns, and the *right section* includes the rightmost $n-m$ columns. When the form of a particular submatrix is known, we label that block accordingly. Otherwise, we place an asterisk (*) in blocks whose contents are not of any particular form.

For column-addition operations, the characteristic matrix has the following form. Every entry on the diagonal is 1. We place an asterisk in each submatrix that contains any nondiagonal 1s as defined by the operation. Returning to the example above, if $b = 1$ and $m = 2$, the form of Q is

$$Q = \begin{array}{c} \begin{array}{ccc} b=1 & m-b=1 & n-m=2 \end{array} \\ \left[\begin{array}{c|c|c} I & * & * \\ \hline 0 & I & 0 \\ \hline 0 & * & I \end{array}\right] \end{array} \begin{array}{l} b=1 \\ m-b=1 \\ n-m=2 \end{array} .$$

We define several column-addition operations and MRC permutations by the form of their characteristic matrices. Each of these forms is nonsingular and characterizes a one-pass permutation. We shall show that the inverse of each of these one-pass permutations falls into a specific class of one-pass permutations.

Trailer matrix form. In section 6, we shall need to transform a nonsingular matrix into one that has a nonsingular trailing $(n-m) \times (n-m)$ submatrix. We shall create the nonsingular trailing submatrix by adding some columns from the left and middle sections to the right section. We define a *trailer matrix* as a column-addition matrix that adds some columns from the left and middle sections into the columns of the right section. The matrix T for this operation is of the form

$$T = \begin{array}{c} \begin{array}{ccc} b & m-b & n-m \end{array} \\ \left[\begin{array}{c|c|c} I & 0 & * \\ \hline 0 & I & * \\ \hline 0 & 0 & I \end{array}\right] \end{array} \begin{array}{l} b \\ m-b \\ n-m \end{array} .$$

The trailer matrix form characterizes an MRC permutation.

Reducer matrix form. Once we have a matrix with a nonsingular trailing submatrix, we need an operation that puts the matrix into "reduced form." (We shall define reduced form precisely in section 6.) We convert a matrix into reduced form by adding columns from the left and middle sections into other columns in the left and middle sections while respecting the dependency restriction. Thus, a *reducer matrix* R is a column-addition matrix of the form

$$R = \begin{array}{c} \begin{array}{ccc} b & m-b & n-m \end{array} \\ \left[\begin{array}{c|c|c} * & * & 0 \\ \hline * & * & 0 \\ \hline 0 & 0 & I \end{array}\right] \end{array} \begin{array}{l} b \\ m-b \\ n-m \end{array} .$$

Since the dependency restriction is obeyed, the leading $m \times m$ submatrix of R is nonsingular. Thus, the matrix R characterizes an MRC permutation.

We can multiply the forms T and R to create another matrix form that also characterizes a one-pass permutation. The product $T\,R$ results in a matrix of the form

$$P = \begin{array}{c} \begin{array}{ccc} b & m-b & n-m \end{array} \\ \left[\begin{array}{c|c|c} * & * & * \\ \hline * & * & * \\ \hline 0 & 0 & I \end{array}\right] \end{array} \begin{array}{l} b \\ m-b \\ n-m \end{array} .$$

Since both of the matrix forms T and R characterize MRC permutations, by Theorem 18, so does the matrix form P and its inverse.

Swapper matrix form. We shall also need to transform the columns in the lower left and lower middle submatrices into columns of zeros. To do so, we must move the nonzero columns in the lower left submatrix into the lower middle submatrix positions by swapping at most $m - b$ columns at a time from the left section with those in the middle section. Thus, the swap operation is a permutation of the leftmost m columns. A *swapper matrix* is of the form

$$S = \begin{array}{c} \begin{array}{cc} m & n-m \end{array} \\ \left[\begin{array}{c|c} \text{permutation} & 0 \\ \hline 0 & I \end{array}\right] \end{array} \begin{array}{l} m \\ n-m \end{array} ,$$

so that the leading $m \times m$ submatrix is a permutation matrix, which dictates the permutation of the leftmost m columns. The matrix form S characterizes an MRC permutation and, by Theorem 18, so does its inverse.

Erasure matrix form. The last operation used in section 6 is an erasure operation to zero out columns in the lower middle submatrix. To perform this operation, we add columns from the right section into columns in the middle section. Thus, an *erasure* matrix form is defined as

$$E = \begin{array}{c} \begin{array}{ccc} b & m-b & n-m \end{array} \\ \left[\begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & I & 0 \\ \hline 0 & * & I \end{array}\right] \end{array} \begin{array}{l} b \\ m-b \\ n-m \end{array} .$$

This matrix form characterizes an MLD permutation because $E_{b..m-1,0..m-1}$'s kernel includes only those m-vectors x with $x_{b..m-1} = 0$, and each such vector is also

in the kernel of $E_{m..n-1,0..m-1}$. Moreover, observe that any matrix of this form is its own inverse. Consequently, the inverse of such a matrix characterizes an MLD permutation.

6. An asymptotically optimal BMMC algorithm. In this section, we present an algorithm to perform any BMMC permutation by factoring its characteristic matrix into matrices which characterize one-pass permutations. We assume that the BMMC permutation is given by an $n \times n$ characteristic matrix A and a complement vector c of length n. We show that the number of parallel I/Os to perform any BMMC permutation is at most $\frac{2N}{BD}(\lceil \frac{\operatorname{rank}\gamma}{\lg(M/B)} \rceil + 2)$ parallel I/Os, where γ is the submatrix $A_{b..n-1,0..b-1}$, which appears in the lower bound given by Theorem 6.

Our strategy is to factor the matrix A into a product of matrices, each of which characterizes an MRC or MLD permutation. For now, we ignore the complement vector c. According to Corollary 2, we read the factors right to left to determine the order in which to perform the permutations.

To obtain the factorization for A, we multiply A by matrices of the forms described in Section 5. By applying these matrix-column operations, we transform the matrix A into a matrix F that characterizes an MRC permutation. Multiplying F by the inverse of each of the matrix-column factors yields the factorization.

Creating a nonsingular trailing submatrix. We start to transform the characteristic matrix A by creating a nonsingular matrix $A^{(1)}$ which has a nonsingular trailing $(n-m) \times (n-m)$ submatrix. We represent the matrix A as

$$A = \begin{array}{c} \begin{array}{cc} m & n-m \end{array} \\ \left[\begin{array}{c|c} \alpha & \beta \\ \hline \phi & \delta \end{array}\right] \begin{array}{c} m \\ n-m \end{array} \end{array} .$$

Our algorithm depends on the structure of ϕ rather than γ. The following lemma allows us to consider $\operatorname{rank}\phi$ instead of $\operatorname{rank}\gamma$ with only a minor difference.

LEMMA 23. *For any matrix A,*

$$\operatorname{rank} A_{b..n-1,0..b-1} - \lg(M/B) \le \operatorname{rank} A_{m..n-1,0..m-1} \le \operatorname{rank} A_{b..n-1,0..b-1} + \lg(M/B) .$$

Proof. Because the rank of a submatrix is the maximum number of linearly independent rows or columns, we have

$$\begin{aligned} \operatorname{rank} A_{m..n-1,0..b-1} &\le \operatorname{rank} A_{b..n-1,0..b-1} \\ &\le \operatorname{rank} A_{m..n-1,0..b-1} + \lg(M/B) , \end{aligned} \tag{12}$$

$$\begin{aligned} \operatorname{rank} A_{m..n-1,0..b-1} &\le \operatorname{rank} A_{m..n-1,0..m-1} \\ &\le \operatorname{rank} A_{m..n-1,0..b-1} + \lg(M/B) . \end{aligned} \tag{13}$$

Subtracting $\lg(M/B)$ from the right-hand inequality of (12) and combining the result with the left-hand inequality of (13) yields

$$\begin{aligned} \operatorname{rank} A_{b..n-1,0..b-1} - \lg(M/B) &\le \operatorname{rank} A_{m..n-1,0..b-1} \\ &\le \operatorname{rank} A_{m..n-1,0..m-1} . \end{aligned} \tag{14}$$

Adding $\lg(M/B)$ to the left-hand inequality of (12) and combining the result with the right-hand inequality of (13) yields

$$\begin{aligned} \operatorname{rank} A_{m..n-1,0..m-1} &\le \operatorname{rank} A_{m..n-1,0..b-1} + \lg(M/B) \\ &\le \operatorname{rank} A_{b..n-1,0..b-1} + \lg(M/B) . \end{aligned} \tag{15}$$

Combining inequalities (14) and (15) proves the lemma. □

By Lemma 23, therefore,

$$\operatorname{rank}\phi \le \operatorname{rank}\gamma + \lg(M/B)\ . \tag{16}$$

We shall use this fact later in the analysis of the algorithm to express the bound in terms of $\operatorname{rank}\gamma$.

We make the trailing $(n-m)\times(n-m)$ submatrix nonsingular by adding columns in ϕ into those in δ. Consider δ as a set of $n-m$ columns and ϕ as a set of m columns. Because A is nonsingular, the submatrix of A consisting of the bottom $n-m$ rows (i.e., submatrices ϕ and δ) has rank $n-m$. Hence, there exists a set of $n-m$ linearly independent columns in the bottom $n-m$ rows of A. We use the method described in section 2 to determine a maximal set V of $\operatorname{rank}\delta$ linearly independent columns in δ and a set W of $n-m-\operatorname{rank}\delta$ columns in ϕ that, along with V, comprise a set of $n-m$ linearly independent columns. Denoting by $\overline{V}$ the $n-m-\operatorname{rank}\delta$ columns of δ not in V, we make the trailing submatrix of A nonsingular by pairing up columns of W with columns of $\overline{V}$ and adding each column in W into its corresponding column in $\overline{V}$. Because V is a maximal set of linearly independent columns in δ, the columns of $\overline{V}$ depend only on columns of V and not on columns of W. Adding a column of W into a column of $\overline{V}$ must produce a column that is linearly independent of those in V. Because each column of $\overline{V}$ has a different column of W added in, the resulting columns are linearly independent of each other, too.

We must express the above transformation as a column-addition operation. Although we focused above on adding columns of ϕ to columns of δ, column-addition operations add entire columns, and so we must also add the corresponding columns of α to the corresponding columns of β. Since we add columns from the leftmost m columns of A to the rightmost $n-m$ columns, the characteristic matrix of this operation has the trailer matrix form T described in section 5. The matrix product is now

$$A^{(1)} \;=\; AT \;=\; \begin{array}{c} \begin{array}{cc} m & n-m \end{array} \\ \left[\begin{array}{c|c} \alpha & \widehat{\beta} \\ \hline \phi & \widehat{\delta} \end{array}\right] \end{array} \begin{array}{l} m \\ n-m \end{array},$$

where $\widehat{\delta}$ is nonsingular. Since the matrices A and T are nonsingular, the matrix $A^{(1)}$ is nonsingular.

Transforming the matrix into reduced form. The next step is to transform the matrix $A^{(1)}$ into reduced form. For our purposes, a matrix is in *reduced form* when there are $\operatorname{rank}\phi$ linearly independent columns and $m-\operatorname{rank}\phi$ columns of zeros in the lower left $(n-m)\times m$ submatrix, and the trailing $(n-m)\times(n-m)$ submatrix is nonsingular. Once again, we use the method of section 2 to determine a set U that indexes $\operatorname{rank}\phi$ linearly independent columns of ϕ. To perform the reduction, we determine for each linearly dependent column ϕ_j a set of column indices $U_j \subseteq U$ such that $\phi_j = \oplus_{k\in U_j}\phi_k$. Adding the set of columns of ϕ indexed by U_j into ϕ_j zeros it out. We add linearly independent columns from the left and middle sections into the linearly dependent columns of these sections. Since we never add a linearly dependent column into any other column and there are no column additions into the linearly independent columns, we respect the dependency restriction. The matrix R that reduces the matrix $A^{(1)}$ has the reducer matrix form described in section 5.

Thus, the matrix product $T\,R$ is of the form P also described in section 5, and it characterizes an MRC permutation. We now have the product

$$A^{(2)} \;=\; A^{(1)}\,R \;=\; A\,T\,R \;=\; A\,P \;=\; \begin{array}{c} \begin{array}{cc} m & n-m \end{array} \\ \left[\begin{array}{c|c} \widehat{\alpha} & \widehat{\beta} \\ \hline \widehat{\phi} & \widehat{\delta} \end{array}\right] \begin{array}{c} m \\ n-m \end{array} \end{array},$$

with a nonsingular trailing submatrix $\widehat{\delta}$ and a lower left submatrix $\widehat{\phi}$ in reduced form. Since A and P are nonsingular, the matrix $A^{(2)}$ is also nonsingular.

Zeroing out the lower left submatrix. Our eventual goal is to transform the original matrix A into a matrix F that characterizes an MRC permutation. At this point, the matrix A has been transformed into the nonsingular matrix $A^{(2)}$. Thus, our final task is to multiply $A^{(2)}$ by a series of matrices that transform the rank ϕ nonzero columns in the lower left $(n-m) \times m$ submatrix $\widehat{\phi}$ into columns of zeros. We do so by multiplying $A^{(2)}$ by matrices of the swapper and erasure matrix forms described in section 5. Let us further partition the leftmost m columns of $A^{(2)}$ into the leftmost b columns and the middle $m-b$ columns:

$$A^{(2)} \;=\; \begin{array}{c} \begin{array}{ccc} b & m-b & n-m \end{array} \\ \left[\begin{array}{c|c|c} \widehat{\alpha}' & \widehat{\alpha}'' & \widehat{\beta} \\ \hline \widehat{\phi}' & \widehat{\phi}'' & \widehat{\delta} \end{array}\right] \begin{array}{c} m \\ n-m \end{array} \end{array}.$$

Our strategy is to repeatedly use swapper matrix forms to move at most $m-b$ columns from the left section into the middle section and then zero out those columns using erasure matrix forms.

We begin by swapping $m - b - \operatorname{rank}\widehat{\phi}''$ columns from $\widehat{\phi}'$ with the zero columns of $\widehat{\phi}''$. Multiplying the matrix $A^{(2)}$ by a nonsingular matrix S_1 of the swapper matrix form described in section 5, we swap at most $m-b$ columns from the left section with the appropriate columns in the middle section. After performing the swap operation on the matrix $A^{(2)}$, the lower left submatrix has $m - b - \operatorname{rank}\widehat{\phi}''$ additional zero columns and the lower middle submatrix has full rank. The above discussion assumes that $\operatorname{rank}\widehat{\phi}' \geq m - b - \operatorname{rank}\widehat{\phi}''$; if the opposite holds, we swap $\operatorname{rank}\widehat{\phi}'$ columns and the lower left submatrix becomes all zeros.

Our next step is to transform the $m - b$ columns in the lower middle submatrix into columns of zeros. Since the nonsingular trailing $(n-m) \times (n-m)$ submatrix $\widehat{\delta}$ forms a basis for the columns of the lower $n - m$ rows of matrix $A^{(2)}$, we zero out each nonzero column in the lower middle submatrix by adding columns of $\widehat{\delta}$. Since we add columns from the rightmost $n - m$ columns into the middle $m - b$ columns, we perform the matrix-column operation characterized by a nonsingular matrix E_1 of the erasure matrix form described in section 5. After multiplying by the erasure matrix E_1, the original matrix is transformed into a nonsingular matrix

$$A^{(3)} = A\,P\,S_1\,E_1\ ,$$

which has zero columns in the lower middle $(n-m) \times (m-b)$ submatrix and possibly some more nonzero columns in the lower left $(n-m) \times b$ submatrix.

If there are still nonzero columns in the lower left $(n-m) \times b$ submatrix of the matrix $A^{(3)}$, then those columns must also be swapped into the lower middle $(n-m) \times (m-b)$ submatrix by a swapper matrix and transformed into zero columns by an erasure matrix. We repeatedly swap in up to $m - b$ nonzero columns of the

lower left submatrix into the lower middle submatrix. Each time we perform a swap operation, we multiply the current product by a matrix S_i of the swapper matrix form. Note that we swap entire columns here, not just the portions in the lower submatrices. After we perform each matrix-column operation S_i, we zero out the lower middle submatrix by multiplying the current product by a matrix E_i of the erasure matrix form.

After repeatedly swapping and erasing each of the nonzero columns in the lower left $(n-m) \times m$ submatrix, the lower left submatrix will contain only zero columns. This matrix is the matrix F mentioned at the beginning of this section. Since the matrix $A^{(2)}$ is in reduced form, there are at most $\operatorname{rank} \phi$ columns in the submatrix $\widehat{\phi}$ that need to be transformed into zero columns. Thus, at most

$$g = \left\lceil \frac{\operatorname{rank} \phi}{m-b} \right\rceil \tag{17}$$

pairs of swap and erasure operations transform all the columns in the lower left $(n-m) \times m$ submatrix into zero columns. Since each matrix-column operation that we performed on the original matrix A to transform it into F is nonsingular, the resulting matrix product

$$F = A\,P\,S_1\,E_1\,S_2\,E_2\,\cdots S_g\,E_g$$

is a nonsingular matrix that characterizes an MRC permutation. Multiplying both sides by the inverses of the factors yields the desired factorization of A:

$$A = F\,E_g^{-1}\,S_g^{-1}\,E_{g-1}^{-1}\,S_{g-1}^{-1}\,\cdots\,E_1^{-1}\,S_1^{-1}P^{-1}\ . \tag{18}$$

Analysis. We now apply several properties that we have gathered to complete the analysis of our BMMC permutation factoring method.

THEOREM 24. *We can perform any BMMC permutation with characteristic matrix A and complement vector c in at most*

$$\frac{2N}{BD}\left(\left\lceil \frac{\operatorname{rank} \gamma}{\lg(M/B)} \right\rceil + 2\right)$$

parallel I/Os, where $\gamma = A_{b..n-1,0..b-1}$, using striped reads, independent writes, and $2N$ records of disk space.

Proof. Ignore the complement vector c for the moment. In the factorization (18) of A, both factors S_1^{-1} and P^{-1} characterize MRC permutations. By Theorem 18, therefore, so does the product $S_1^{-1}\,P^{-1}$. As we saw in section 5, each factor E_i^{-1} characterizes an MLD permutation. Applying Theorem 17, each grouping of factors $E_1^{-1}\,S_1^{-1}\,P^{-1}$ and $E_i^{-1}\,S_i^{-1}$, for $i = 2, 3, \ldots, g$, characterizes an MLD permutation. By Theorem 15, and adding in one more pass for the MRC permutation characterized by F, we can perform A with $g+1$ passes.

If the complement vector c is nonzero, we include it as part of the MRC permutation characterized by the leftmost factor F. See [10] for details.

Regardless of the complement vector, therefore, we can perform the BMMC permutation with $g+1$ passes. Combining equation (17) and inequality (16), we obtain a bound of

$$g + 1 = \left\lceil \frac{\operatorname{rank} \phi}{\lg(M/B)} \right\rceil + 1$$

$$\leq \left\lceil \frac{\operatorname{rank}\gamma + \lg(M/B)}{\lg(M/B)} \right\rceil + 1$$
$$= \left\lceil \frac{\operatorname{rank}\gamma}{\lg(M/B)} \right\rceil + 2$$

passes for a total of at most

$$\frac{2N}{BD}\left(\left\lceil \frac{\operatorname{rank}\gamma}{\lg(M/B)} \right\rceil + 2\right)$$

parallel I/Os.

Because each factor characterizes either an MRC permutation (performed with striped reads and writes) or an MLD permutation (performed with striped reads and independent writes), and striped I/O is a special case of independent I/O, the method as a whole uses striped reads and independent writes. The method uses $2N$ records of disk space by reversing the roles of the source and target portions between passes. That is, the target portion written to in one pass becomes the source portion read from in the next pass. □

Performing BMMC permutations with striped writes. Here we describe another way to compose the factors from the product of equation (18) into $g + 1$ factors, such that each factor characterizes either an MRC or MLD^{-1} permutation. Both MRC and MLD^{-1} permutations can be performed with striped writes. As mentioned in section 4, striped writes may have advantages in parallel disk systems organized as RAID levels 3 or 4.

In our alternative factorization, we start by noting that because any erasure matrix is its own inverse, not only does each factor E_i^{-1} in the factorization (18) characterize an MLD permutation, it also characterizes an MLD^{-1} permutation. Instead of grouping the factors $E_1^{-1} S_1^{-1} P^{-1}$ and $E_i^{-1} S_i^{-1}$ for $i = 2, 3, \ldots, g$, here we group by $F E_g^{-1}$ and $S_i^{-1} E_{i-1}^{-1}$, for $i = 2, 3, \ldots, g$. By Lemma 21, each such grouping of factors characterizes an MLD^{-1} permutation. Thus, the resulting factorization of A has g MLD^{-1} factors and the MRC product $S_1^{-1} P^{-1}$. This alternative factorization of A has the same number of one-pass factors as our previous grouping, but it uses only MRC and MLD^{-1} permutations as its factors. Thus, we have proven the following.

THEOREM 25. *We can perform any BMMC permutation with characteristic matrix A and complement vector c in at most*

$$\frac{2N}{BD}\left(\left\lceil \frac{\operatorname{rank}\gamma}{\lg(M/B)} \right\rceil + 2\right)$$

parallel I/Os, where $\gamma = A_{b..n-1,0..b-1}$, using independent reads, striped writes, and $2N$ records of disk space. □

7. Detecting BMMC permutations at run time. In practice, we wish to run the BMMC algorithm of section 6 whenever possible to reap the savings over having run the more costly algorithm for general permutations. For that matter, we wish to run even faster algorithms for any of the special cases of BMMC permutations (MRC, MLD, MLD^{-1}, or block BMMC [10]) whenever possible as well. We must know the characteristic matrix A and complement vector c, however, to run any of these algorithms. If A and c are specified in the source code, before running the algorithm we only need to check that A is of the correct form, e.g., that it is nonsingular for a BMMC permutation, of the MLD or MRC form, etc. Later in this section, we show

how to check the kernel condition for MLD permutations. If instead the permutation is given by a vector of N target addresses, we can detect at run time whether it is a BMMC permutation by the following procedure.

1. Check that N is a power of 2.

2. Form a candidate characteristic matrix A and complement vector c such that if the permutation is BMMC, then A and c must be the correct characterizations. This section shows how to do so with only $\lceil \frac{\lg(N/B)+1}{D} \rceil$ parallel reads.

3. Check that the characteristic matrix is of the correct form. That is, check that it is nonsingular, which is easily done by the method of section 2. If further structure is desired, e.g., MRC or MLD forms, check further for the desired form.

4. Verify that all N target addresses are described by the candidate characteristic matrix and complement vector. If for any source address x and its corresponding target address y we have $y \neq A\,x \oplus c$, the permutation is not BMMC and we can terminate verification. If $y = A\,x \oplus c$ for all N source-target pairs, the permutation is BMMC. Verification uses at most N/BD parallel reads, since we need to read each target address at most once. Source addresses are generated implicitly, and so they do not entail any I/O cost.

The total number of parallel I/Os is at most

$$\frac{N}{BD} + \left\lceil \frac{\lg(N/B)+1}{D} \right\rceil ,$$

all of which are reads, and it is usually far fewer when the permutation turns out not to be BMMC.

One benefit of run-time BMMC detection is that the programmer might not realize that the desired permutation is BMMC. For example, as noted in section 1, the standard binary reflected Gray code and its inverse are both MRC permutations. Yet the programmer might not know to call a special MRC or BMMC routine. Even if the system provides an entry point to perform the standard Gray code permutation and this routine invokes the MRC algorithm, variations on the standard Gray code may foil this approach. For example, a standard Gray code with all bits permuted the same (i.e., a characteristic matrix of $\Pi\,G$, where Π is a permutation matrix and G is the MRC matrix that characterizes the standard Gray code) is BMMC but not necessarily MRC. It might not be obvious enough that the permutation characterized by $\Pi\,G$ is BMMC for the programmer to invoke the BMMC algorithm explicitly.

Forming the candidate characteristic matrix and complement vector. The method for forming the candidate characteristic matrix A and candidate complement vector c is based on two observations. First, if the permutation is BMMC, then the complement vector c must be the target address corresponding to source address 0. This relationship holds because $x = 0$ and $y = A\,x \oplus c$ imply that $y = c$.

The second observation is as follows. Consider a source address $x = (x_0, x_1, \ldots, x_{n-1})$, and suppose that bit position k holds a 1, i.e., $x_k = 1$. Let us denote the jth column for matrix A by A_j. Also, let S_k denote the set of bit positions in x other than k that hold a 1: $S_k = \{j : j \neq k \text{ and } x_j = 1\}$. If $y = A\,x \oplus c$, then we have

$$y = \left(\bigoplus_{j \in S_k} A_j \right) \oplus A_k \oplus c \ , \tag{19}$$

since only the bit positions j for which $x_j = 1$ contribute a column of A to the sum of columns that forms the matrix-vector product. If we know the target address y, the

complement vector c, and the columns A_j for all $j \neq k$, we can rewrite equation (19) to yield the kth column of A:

$$A_k = y \oplus \left(\bigoplus_{j \in S_k} A_j \right) \oplus c \ . \tag{20}$$

We shall compute the complement vector c first and then the columns of the characteristic matrix A one at a time, from A_0 up to A_{n-1}. When computing A_k, we will have already computed $A_0, A_1, \ldots, A_{k-1}$, and these will be the only columns we need in order to apply equation (20). In other words, $S_k \subseteq \{0, 1, \ldots, k-1\}$. Recall that as Fig. 2 shows, the lower b bits of a record's address give the record's offset within its block, the middle d bits give the disk number, and the upper $s = n-(b+d)$ bits give the stripe number.

From equation (20), it would be easy to compute A_k if S_k were empty. The set S_k is empty if the source address is a unit vector, with its only 1 in position k. If we look at these addresses, however, we find that the target addresses for a disproportionate number—all but d of them—reside on disk $\mathcal{D}_0$. The block whose disk and stripe fields are all zero contains b such addresses, so they can be fetched in one disk read. A problem arises for the s source addresses with one 1 in the stripe field: their target addresses all reside on different blocks of disk $\mathcal{D}_0$. If we use this method, each of these blocks must be fetched in a separate read. The total number of parallel reads to fetch all the target addresses corresponding to all unit-vector source addresses is $s + 1 = \lg(N/BD) + 1$.

To achieve only $\lceil \frac{\lg(N/B)+1}{D} \rceil$ parallel reads, each read fetches one block from each of the D disks. The first parallel read determines the complement vector, the first $b + d$ columns, and the next $D - d - 1$ columns. Each subsequent read determines another D columns, until all n columns have been determined.

In the first parallel read, we do the same as above for the first $b + d$ bits. That is, we fetch blocks containing target addresses whose corresponding source addresses are unit vectors with one 1 in the first $b + d$ positions. As before, b of them are in the same block on disk $\mathcal{D}_0$. This block also contains address 0, which we need to compute the complement vector. The remaining d are in stripe number 0 of disks $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_4, \mathcal{D}_8, \ldots, \mathcal{D}_{D/2}$. Having fetched the corresponding target addresses, we have all the information we need to compute the complement vector c and columns $A_0, A_1, \ldots, A_{b+d-1}$.

The columns we have yet to compute correspond to bit positions in the stripe field. If we were to compute these columns in the same fashion as the first $b + d$, we would again encounter the problem that all the blocks we need to read are on disk $\mathcal{D}_0$. In the first parallel read, the only unused disks remaining are those whose numbers are not a power of 2 ($\mathcal{D}_3, \mathcal{D}_5, \mathcal{D}_6, \mathcal{D}_7, \mathcal{D}_9, \ldots$). The key observation is that we have already computed all d columns corresponding to the disk field, and we can thus apply equation (20). For example, let us compute column A_{b+d}, which corresponds to the first bit of the stripe number. We read stripe 1 on disk $\mathcal{D}_3$ and find the first target address y in this block. Disk number 3 corresponds to the first two disk-number columns, A_b and A_{b+1}. Applying equation (20) with $S_{b+d} = \{b, b+1\}$, we compute $A_{b+d} = y \oplus A_b \oplus A_{b+1} \oplus c$. The next column we compute is A_{b+d+1}. Reading the block at stripe 2 on disk $\mathcal{D}_5$, we fetch a target address y and then compute $A_{b+d+1} = y \oplus A_b \oplus A_{b+2} \oplus c$. Continuing on in this fashion, we compute a total of $D - d - 1$ stripe-bit columns from the first parallel read.

The remaining parallel reads compute the remaining stripe-bit columns. We follow the stripe-bit pattern of the first read, but we use all disks, not just those whose disk numbers are not powers of 2. Each block read fetches a target address y, which we exclusive-or with a set of columns from the disk field and with the complement vector to compute a new column from the stripe field. The first parallel read computes $b + D - 1$ columns, and all subsequent parallel reads compute D columns. The total number of parallel reads is thus

$$\begin{aligned} 1 + \left\lceil \frac{n - (b + D - 1)}{D} \right\rceil &= 1 + \left\lceil \frac{\lg(N/B) - D + 1}{D} \right\rceil \\ &= \left\lceil \frac{\lg(N/B) + 1}{D} \right\rceil . \end{aligned}$$

Checking the kernel condition for MLD permutations. In practice, we would like a simple procedure to verify that a given matrix characterizes an MLD permutation. By the method of section 2, it is easy to verify that a candidate matrix A is nonsingular. It may not be obvious how to verify that $\ker \lambda \subseteq \ker \mu$ when the matrix is blocked into λ and μ. Instead of determining directly whether $\ker \lambda \subseteq \ker \mu$, we check whether $\operatorname{row} \mu \subseteq \operatorname{row} \lambda$. By Lemma 5, these conditions are equivalent.

Checking whether $\operatorname{row} \mu \subseteq \operatorname{row} \lambda$ is easy. Note that the rows of the submatrix $A_{b..n-1,0..m-1}$ consist of the union of the rows of λ and μ, and observe that $\operatorname{row} \mu \subseteq \operatorname{row} \lambda$ if and only if $\operatorname{row} \lambda = \operatorname{row} A_{b..n-1,0..m-1}$. That is, if including the rows of μ adds no new vectors to the row space of λ, then the row space of μ must be a subset of the row space of λ. The condition $\operatorname{row} \lambda = \operatorname{row} A_{b..n-1,0..m-1}$ is equivalent to $\dim \operatorname{row} \lambda = \dim \operatorname{row} A_{b..n-1,0..m-1}$, which is in turn equivalent to $\operatorname{rank} \lambda = \operatorname{rank} A_{b..n-1,0..m-1}$. We can check this last condition easily by using the method of section 2.

8. Conclusions. This paper has shown an asymptotically tight bound on the number of parallel I/Os required to perform BMMC permutations on parallel disk systems. It is particularly satisfying that the tight bound was achieved not by raising the lower bound proven here and in [9], but by decreasing the upper bound in [10]. (After all, we would rather perform BMMC permutations with fewer parallel I/Os.) The multiplicative and additive constants in the I/O complexity of our algorithm are small, which is especially fortunate in light of the expense of disk accesses. Our algorithm has been implemented on a DEC 2100 server with eight disk drives [13]. This implementation uses asynchronous independent reads and asynchronous striped writes, so that when performing each MRC or MLD^{-1} permutation, it overlaps prefetching the next memoryload, writing the previous memoryload, and permuting in memory the current memoryload. A later implementation that runs on either the DEC 2100 server or a network of workstations [11] uses asynchronous striped reads and asynchronous independent writes; it is a key subroutine in an efficient out-of-core FFT implementation [12].

One can adapt the proof by Aggarwal and Vitter [2] of Lemma 9 to bound $\Delta\Phi_{\max}$ precisely, rather than just asymptotically. In particular, it is a straightforward exercise to derive the bound

$$\Delta\Phi_{\max} \leq B \left(\frac{2}{e \ln 2} + \lg(M/B) \right) .$$

Moreover, the potential change is at most zero for write operations, and so the potential increases only during read operations. If all I/Os are simple, then the total

number of blocks read equals the total number of blocks written. Therefore, we can modify the lower bound of Lemma 8 to $2\,\frac{\Phi(t)-\Phi(0)}{\Delta\Phi_{\max}}$, with which we can derive a lower bound of

$$\frac{2N}{BD}\,\frac{\operatorname{rank}\gamma}{\frac{2}{e\ln 2}+\lg(M/B)}$$

parallel I/Os for any BMMC permutation. Since the quantity $2/(e\ln 2)$ is approximately 1.06, this lower bound is quite close to the exact upper bound given by Theorem 24.

We have also shown how to detect BMMC permutations at run time, given a vector of target addresses. Detection is inexpensive and, when successful, permits the execution of our BMMC algorithm or possibly a faster algorithm for a more restricted permutation class.

Wisniewski [25] uses the linear-algebraic technique of performing column additions and row additions to derive BMMC-permutation algorithms on distributed-memory models and main-memory/cache models. On what other memory models can we use this technique to efficiently perform BMMC permutations?

What other permutations can be performed quickly? Several $O(1)$-pass permutation classes appear in [9], and this paper has added two more (MLD and MLD^{-1} permutations in section 4). We have shown that the inverse of any one-pass permutation is a one-pass permutation. One can also show that the composition of an MLD permutation with an MLD^{-1} permutation is a one-pass permutation. What other useful permutation classes can we show to be BMMC?

Finally, is the lower bound of $\Omega(\frac{\Phi(t)-\Phi(0)}{\Delta\Phi_{\max}})$ parallel I/Os universally tight for all permutations, not just those that are BMMC? One possible approach is to design an algorithm that explicitly manages the potential. If each pass increases the potential by $\Theta\left(\frac{N}{BD}(\Delta\Phi_{\max})\right)$, the algorithm's I/O count would match the lower bound.

Acknowledgments. Thanks to C. Esther Jeserum, Michael Klugerman, and the anonymous referees for their helpful suggestions.

REFERENCES

[1] A. Aggarwal, A. K. Chandra, and M. Snir, *Hierarchical memory with block transfer*, in Proceedings of the 28th Annual Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1987, pp. 204–216.

[2] A. Aggarwal and J. S. Vitter, *The input/output complexity of sorting and related problems*, Comm. ACM, 31 (1988), pp. 1116–1127.

[3] L. Arge, *The buffer tree: A new technique for optimal I/O-algorithms*, in 4th International Workshop on Algorithms and Data Structures (WADS), Lecture Notes in Computer Science 955, Springer-Verlag, New York, 1995, pp. 334–345.

[4] L. Arge, *The I/O-complexity of ordered binary-decision diagram manipulation*, in Proceedings of the 6th International Symposium on Algorithms and Computations (ISAAC '95), Lecture Notes in Computer Science 1004, J. Staples, P. Eades, N. Katoh, and A. Moffat, eds., Springer-Verlag, New York, 1995, pp. 82–91.

[5] L. Arge, D. E. Vengroff, and J. S. Vitter, *External-memory algorithms for processing line segments in geographic information systems*, in Proceedings of the Third Annual European Symposium on Algorithms (ESA '95), Lecture Notes in Computer Science 979, P. Spirakis, ed., Springer-Verlag, New York, 1995, pp. 295–310.

[6] R. D. Barve, E. F. Grove, and J. S. Vitter, *Simple randomized mergesort for parallel disks*, in Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1996, pp. 109–118.

[7] P. Chen, G. Gibson, R. H. Katz, D. A. Patterson, and M. Schulze, *Two Papers on RAIDs*, Tech. Report UCB/CSD 88/479, Computer Science Division (EECS), University of California, Berkeley, 1988.

[8] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter, *External-memory graph algorithms*, in Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1995, pp. 139–149.

[9] T. H. Cormen, *Virtual Memory for Data-Parallel Computing*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1992; also available as Technical Report MIT/LCS/TR-559.

[10] T. H. Cormen, *Fast permuting in disk arrays*, J. Parallel Distrib. Computing, 17 (1993), pp. 41–57.

[11] T. H. Cormen and M. Hirschl, *Early experiences in evaluating the Parallel Disk Model with the ViC* implementation*, Parallel Comput., 23 (1997), pp. 571–600.

[12] T. H. Cormen and D. M. Nicol, *Performing out-of-core FFTs on parallel disk systems*, Tech. Report PCS-TR96-294, Department of Computer Science, Dartmouth College, Hanover, NH, 1996; Parallel Comput., to appear.

[13] S. R. Cushman, *A Multiple Discrete Pass Algorithm on a DEC Alpha* 2100, Tech. Report PCS-TR95-259, Department of Computer Science, Dartmouth College, Hanover, NH, 1995.

[14] J. M. del Rosario and A. Choudhary, *High-performance I/O for massively parallel computers*, IEEE Computer, (1994), pp. 59–67.

[15] A. Edelman, S. Heller, and S. L. Johnsson, *Index transformation algorithms in a linear algebra framework*, IEEE Trans. Parallel Distributed Systems, 5 (1994), pp. 1302–1309.

[16] R. W. Floyd, *Permuting information in idealized two-level storage*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 105–109.

[17] G. A. Gibson, *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*, The MIT Press, Cambridge, MA, 1992; also available as Tech. Report UCB/CSD 91/613, Computer Science Division (EECS), University of California, Berkeley, 1991.

[18] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter, *External-memory computational geometry*, in Proceedings of the 34th Annual Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1993, pp. 714–723.

[19] S. L. Johnsson and C.-T. Ho, *Generalized shuffle permutations on boolean cubes*, J. Parallel Distrib. Computing, 16 (1992), pp. 1–14.

[20] S. Lang, *Linear Algebra*, 3rd ed., Springer-Verlag, New York, 1987.

[21] M. H. Nodine and J. S. Vitter, *Deterministic distribution sort in shared and distributed memory multiprocessors*, in Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1993, pp. 120–129.

[22] M. H. Nodine and J. S. Vitter, *Greed sort: Optimal deterministic sorting on parallel disks*, J. Assoc. Comput. Mach., 42 (1995), pp. 919–933.

[23] G. Strang, *Linear Algebra and Its Applications*, 3rd ed. Harcourt Brace Jovanovich, San Diego, CA, 1988.

[24] J. S. Vitter and E. A. M. Shriver, *Algorithms for parallel memory* I: *Two-level memories*, Algorithmica, 12 (1994), pp. 110–147.

[25] L. F. Wisniewski, *Efficient Design and Implementation of Permutation Algorithms on the Memory Hierarchy*, Ph.D. thesis, Department of Computer Science, Dartmouth College, Hanover, NH, 1996.

[26] L. F. Wisniewski, *Structured permuting in place on parallel disk systems*, in Proceedings of the Fourth Annual Workshop on I/O in Parallel and Distributed Systems (IOPADS), ACM, New York, 1996, pp. 128–139.

[27] D. Womble, D. Greenberg, S. Wheat, and R. Riesen, *Beyond core: Making parallel computer I/O practical*, in DAGS '93, Dartmouth Institute for Advanced Graduate Studies, Hanover, NH, 1993.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 137–151

L-PRINTABLE SETS*

LANCE FORTNOW†, JUDY GOLDSMITH‡, MATTHEW A. LEVY‡, AND STEPHEN MAHANEY§

Abstract. A language is L-printable if there is a logspace algorithm which, on input 1^n, prints all members in the language of length n. Following the work of Allender and Rubinstein [*SIAM J. Comput.*, 17 (1988), pp. 1193–1202] on P-printable sets, we present some simple properties of the L-printable sets. This definition of "L-printable" is robust and allows us to give alternate characterizations of the L-printable sets in terms of tally sets and Kolmogorov complexity. In addition, we show that a regular or context-free language is L-printable if and only if it is sparse, and we investigate the relationship between L-printable sets, L-rankable sets (i.e., sets A having a logspace algorithm that, on input x, outputs the number of elements of A that precede x in the standard lexicographic ordering of strings), and the sparse sets in L. We prove that under reasonable complexity-theoretic assumptions, these three classes of sets are all different. We also show that the class of sets of small generalized Kolmogorov space complexity is exactly the class of sets that are L-isomorphic to tally languages.

Key words. sparse sets, logspace, L-isomorphisms, Kolmogorov complexity, computational complexity, ranking, regular languages, context-free languages

AMS subject classifications. 68Q15, 68Q30, 68Q05, 68Q68, 03D05, 03D15, 03D30

PII. S0097539796300441

1. Introduction.

What is an easy set? Typically, complexity theorists view easy sets as those with easy membership tests. An even stronger requirement might be that there is an easy algorithm to print all the elements of a given length. These "printable" sets are easy enough that we can efficiently retrieve all of the information we might need about them.

Hartmanis and Yesha first defined P-printable sets in 1984 [HY84]. A set A is P-printable if there is a polynomial-time algorithm that on input 1^n outputs all of the elements of A of length n. Any P-printable set must lie in P and be sparse; i.e., the number of strings of each length is bounded by a fixed polynomial of that length. Allender and Rubinstein [AR88] give an in-depth analysis of the complexity of the P-printable sets.

Once P-printability has been defined, it is natural to consider the analogous notion of logspace-printability. Since it is not known whether or not L = P, an obvious question to ask is: do the L-printable sets behave differently than the P-printable sets? In this paper, we are able to answer this question in the affirmative, at least under plausible complexity-theoretic assumptions. Jenner and Kirsig [JK89] define L-printability as the logspace computable version of P-printability. Because L-printability implies P-printability, every L-printable set must be sparse and lie in L. In this paper we give

*Received by the editors February 12, 1996; accepted for publication (in revised form) November 15, 1996; published electronically June 15, 1998.
http://www.siam.org/journals/sicomp/28-1/30044.html

†Department of Computer Science, University of Chicago, Chicago, IL 60637. The work of this author was supported in part by NSF grant CCR-9253582.

‡Department of Computer Science, University of Kentucky, Lexington, KY 40506-0046. The work of these authors was supported in part by NSF grant CCR-9315354. The work of the third author was also supported in part by a University of Kentucky Presidential Fellowship.

§DIMACS Center, Rutgers University, Piscataway, NJ 08855. The work of this author was supported by NSF cooperative agreement CCR-9119999 and a grant from the New Jersey Commission on Science and Technology.

the first in-depth analysis of the complexity of L-printable sets. (Jenner and Kirsig focused only one chapter on printability, and most of their printability results concern NL-printable sets.)

Whenever a new class of sets is analyzed, it is natural to wonder about the structure of those sets. Hence, we examine the regular and context-free L-printable sets. Using characterizations of the sparse regular and context-free languages, we show in section 4 that every sparse regular or context-free language is L-printable. (Although the regular sets are a special case of the context-free sets, we include the results for the regular languages because our characterization of the sparse regular languages is simple and intuitive.)

We might expect many of the properties of P-printable sets to have logspace analogues, and, in fact, this is the case. In section 5 we show that L-printable sets (like their polynomial-time counterparts) are closely related to tally sets in L and to sets in L with low generalized *space-bounded* Kolmogorov complexity.

A set is said to have *small generalized Kolmogorov complexity* if all of its strings are highly compressible and easily restorable. Generalized time-bounded Kolmogorov complexity and generalized space-bounded Kolmogorov complexity are introduced in [Har83] and [Sip83]. Several researchers [Rub86, BB86, HH88] show that P-printable sets are exactly the sets in P with small generalized time-bounded Kolmogorov complexity. [AR88] show that a set has small generalized time-bounded Kolmogorov complexity if and only if it is P-isomorphic to a tally set. Using similar techniques, we show in section 5 that the L-printable sets are exactly the sets in L with small generalized space-bounded Kolmogorov complexity. We also prove that a set has small generalized space-bounded Kolmogorov complexity if and only if it is L-isomorphic to a tally set.

In section 6, we note that sets that can be ranked in logspace (i.e., given a string x, a logspace algorithm can determine the number of elements in the set $\leq x$) seem different from the L-printable sets. For sparse sets, P-rankability is equivalent to P-printability. We show a somewhat surprising result in section 6, namely, that the sparse L-rankable sets and the L-printable sets are the same if and only if there are no tally sets in $\mathrm{P} - \mathrm{L}$ if and only if LinearSPACE = E.

Are all sparse sets in L either L-printable or L-rankable? Allender and Rubinstein [AR88] show that every sparse set in P is P-printable if and only if there are no sparse sets in $\mathrm{FewP} - \mathrm{P}$. In section 6, we similarly show a stronger collapse: every sparse set in L is L-printable if and only if every sparse set in L is L-rankable if and only if there are no sparse sets in $\mathrm{FewP} - \mathrm{L}$ if and only if LinearSPACE = FewE.

Unlike L-printable sets, L-rankable sets may have exponential density. Blum (see [GS91]) shows that every set in P is P-rankable if and only if every #P function is computable in polynomial time. In section 6, we also show that every set in L is L-rankable if and only if every #P function is computable in logarithmic space.

2. Definitions. We assume a basic familiarity with Turing machines and Turing machine complexity. For more information on complexity theory, we suggest either [BDG88] or [Pap94]. We also assume a familiarity with regular languages and expressions and context-free languages as found in [Mar91]. We denote the characteristic function of A by χ_A. We use the standard lexicographic ordering on strings and let $|w|$ be the length of the string w. (Recall that $w \leq_{lex} v$ iff $|w| < |v|$ or $|w| = |v|$ and, if i is the position of the leftmost bit where w and v differ, $w_i < v_i$.) The alphabet $\Sigma = \{0, 1\}$, and all strings are elements of Σ^*. We denote the complement of A by $\overline{A}$.

The class P is deterministic polynomial time, and L is deterministic logarithmic space; remember that in calculating space complexity, the machine is assumed to have separate tapes for input, computation, and output. The space restriction applies only to the work tape. It is known that $\mathrm{L} \subseteq \mathrm{P}$, but it is not known whether the two classes are equal. The class E is deterministic time $2^{O(n)}$, and LinearSPACE is deterministic space $O(n)$.

DEFINITION 2.1. *A set A is in the class* PP *if there is a polynomial-time nondeterministic Turing machine that, on input x, accepts with more than half its computations if and only if $x \in A$. A function f is in #P if there is a polynomial-time nondeterministic Turing machine M such that for all x, $f(x)$ is the number of accepting computations of $M(x)$.*

Allender[All86] defined the class FewP. FewE is defined analogously.

DEFINITION 2.2 (see [All86]). *A set A is in the class* FewP *if there is a polynomial-time nondeterministic Turing machine M and a polynomial p such that on all inputs x, M accepts x on at most $p(|x|)$ paths. A set A is in the class* FewE *if there is an exponential-time nondeterministic Turing machine M and a constant c such that on all inputs x, M accepts x on at most 2^{cn} paths. (Note that this is small compared to the double exponential number of paths of an exponential-time nondeterministic Turing machine.)*

DEFINITION 2.3. *A set S is* sparse *if there is some polynomial $p(n)$ such that for all n, the number of strings in S of length n is bounded by $p(n)$ (i.e., $|S^{=n}| \leq p(n)$).*

A set T over alphabet Σ is a tally *set if $T \subseteq \{\sigma\}^*$, for some character $\sigma \in \Sigma$.*

The work here describes certain enumeration properties of sparse sets in L. There are two notions of enumeration that are considered: rankability and printability.

DEFINITION 2.4. *If $\mathcal{C}$ is a complexity class, then a set A is* $\mathcal{C}$-printable *if and only if there is a function computable in $\mathcal{C}$ that, on any input of length n, outputs all the strings of length n in A.*

Note that P-printable sets are necessarily in P and are sparse, since all of the strings of length n must be printed in time polynomial in n. Since every logspace computable function is also computable in polynomial time, L-printable sets are also P-printable, and thus are also sparse.

DEFINITION 2.5. *If $\mathcal{C}$ is a complexity class, then a set, A, is* $\mathcal{C}$-rankable *if and only if there is a function r_A computable in $\mathcal{C}$ such that $r_A(x) = |\{y \leq_{lex} x : y \in A\}|$. (In other words, $r_A(x)$ gives the lexicographic rank of x in A.) The function r_A is called the* ranking function *for A.*

Note that P-rankable sets are necessarily in P but are not necessarily sparse. Furthermore, a set is P-rankable if and only if its complement is P-rankable. Finally, note that any P-printable set is P-rankable.

DEFINITION 2.6. *If $\mathcal{C}$ is a complexity class, then two sets, A and B, are* $\mathcal{C}$-isomorphic *($A \cong_{\mathcal{C}} B$) if there are total functions f and g computable in $\mathcal{C}$ that are both one–one and onto, such that $f(g(x)) = x$ and $g(f(y)) = y$, f is a reduction from A to B, and g is a reduction from B to A.*

In order for two sets to be P-isomorphic, their density functions must be close to each other: if one set is sparse and the other is not, then any one–one reduction from the sparse set to the dense set must have superpolynomial growth rate. By the same argument, if one has a superpolynomial gap, the other must have a similar gap.

A lexicographic (or order-preserving) isomorphism from A to B is, informally, a bijection that maps the ith element of A to the ith element of B and maps the ith element of $\overline{A}$ to the ith element of $\overline{B}$. Note that in the definition of similar densities,

the isomorphisms need not be computable in any particular complexity class. This merely provides the necessary condition on densities in order for the two sets to be P-isomorphic or L-isomorphic.

DEFINITION 2.7. *Two sets, A and B, have* similar densities *if the lexicographic isomorphisms from A to B and from B to A are polynomial-size bounded.*

The notion of printability, or of ranking on sparse sets, can be considered a form of compression. Another approach to compression is found in the study of Kolmogorov complexity; a string is said to have "low information content" if it has low Kolmogorov complexity. We are interested in the space-bounded Kolmogorov complexity class defined by Hartmanis [Har83].

DEFINITION 2.8. *Let M_v be a Turing machine, and let f and s be functions on the natural numbers. Then we define*

$$KS_v[f(n), s(n)] = \{w : |w| = n \text{ and } \exists y(|y| \leq f(n) \text{ and } M_v(y) = w \text{ and } M_v \text{ uses } s(n) \text{ space})\}.$$

Following the notation of [AR88], we refer to y as the *compressed string*, $f(n)$ as the *compression*, and $s(n)$ as the *restoration space*. Hartmanis [Har83] shows that there exists a universal machine M_u such that for all v, there exists a constant c such that $KS_v[f(n), s(n)] \subseteq KS_u[f(n) + c, cs(n) + c]$. We will drop the subscript and let $KS[f(n), s(n)] = KS_u[f(n), s(n)]$.

3. Basic results. We begin by formalizing some observations from the previous section.

OBSERVATION 3.1. *If A is* L*-printable, then A has polynomially bounded density, i.e., A is sparse.*

This follows immediately from the fact that logspace computable functions are P-time computable (i.e., L-printability implies P-printability), and from the observations on P-printable sets.

PROPOSITION 3.2 (see [JK89]). *If A is* L*-printable, then $A \in$ L.*

Proof. To decide $x \in A$, simulate the L-printing function for A with input $1^{|x|}$. As each $y \in A$ is "printed," compare it, bit by bit, with x. If $y = x$, accept. Because the comparisons can be done using $O(1)$ space, and the L-printing function takes $O(\log |x|)$ space, this is a logspace procedure. □

PROPOSITION 3.3. *If A is* L*-rankable, then $A \in$ L.*

Proof. Note that the function $x - 1$ (the lexicographic predecessor of x) can be computed (though not written) in space logarithmic in $|x|$. Since logspace computable functions are closed under composition, $r_A(x - 1)$ can be computed in logspace, as can $r_A(x) - r_A(x - 1) = \chi_A(x)$. □

PROPOSITION 3.4. *If A is* L*-printable, then A is* L*-rankable.*

Proof. To compute the rank of x, we print the strings of A up to $|x|$ and count the ones that are lexicographically smaller than x. Since A is sparse, by Observation 3.1, we can store this counter in logspace. □

We can now prove the following, first shown by [JK89] with a different proof.

PROPOSITION 3.5 (see [JK89]). *If A is* L*-printable, then A is* L*-printable in lexicographically increasing order.*

Proof. To prove this, we use a variation on selection sort. Suppose the logspace machine M L-prints A. Then we can construct another machine, N, to L-print A in lexicographically increasing order. Note that it is possible to store an instantaneous description of a logspace machine, i.e., the position of the input head, the state, the contents of the worktape, and the character just output, in $O(\log |x|)$ space.

The basic idea is that we store, during the computation, enough information to produce three strings: the most recently printed string (in the lexicographically ordered printing), the current candidate for the next string to be printed, and the current contender. We can certainly store three IDs for M in logspace. Each ID describes the state of M immediately prior to printing the desired string.

In addition to storing the IDs, we must simulate M on these three computations in parallel, so that we can compare the resulting strings bit by bit. If the contender string is greater than the last string output (so it has not already been output) and less than the candidate, it becomes the new candidate. Otherwise, the final ID of the computation becomes the new contender. These simulated computations do not produce output for N; when the next string is found for N to print, its initial ID is available, and the simulation is repeated, with output. □

Using the same technique as in the previous proof, one can easily show the following.

PROPOSITION 3.6. *If A is* L*-printable, and $A \cong_{log} B$, then B is* L*-printable as well.* □

4. L-printable sets. We begin this section with a very simple example of a class of L-printable sets.

PROPOSITION 4.1 (see [JK89]). *The tally sets in* L *are* L*-printable.*

Proof. On input of length n, decide whether $1^n \in A$. If so, print it. □

One may ask: are all of the L-printable sets as trivial as Proposition 4.1? We demonstrate in the following sections that every regular language or context-free language that is sparse is also L-printable (see Theorem 4.8 and Corollary 4.14). We also give an L-printable set that is neither regular nor context-free (see Proposition 4.15).

4.1. Sparse regular languages. We show that the sparse regular languages are L-printable. In order to do so, we give some preliminary results about regular expressions.

DEFINITION 4.2 (see [BEGO71]). *Let r be a regular expression. We say r is* unambiguous *if every string has at most one derivation from r.*

THEOREM 4.3 (see [BEGO71]). *For every regular language L, there exists an unambiguous regular expression r such that $L(r) = L$.*

Proof (sketch). Represent L as the union of disjoint languages whose deterministic finite automatons (DFAs) have a unique final state. Using the standard union construction of a nondeterministic finite automaton (NFA) from a DFA, we get an NFA with the property that each string has a unique accepting path. Now, using state elimination to construct a regular expression from this NFA, the unique path for each string becomes a unique derivation from the regular expression. □

We should note that even though removal of ambiguity from a regular expression is, in general, PSPACE-complete [SH85], this does not concern us. Theorem 4.3 guarantees the existence of an unambiguous regular expression, corresponding to every regular language, that is sufficient for our needs.

We now define a restricted form of regular expression that will generate precisely the sparse regular languages. (Note that a similar, although more involved, characterization was given in [SSYZ92]. They give characterizations for a variety of densities, whereas we are only concerned with sparse sets.)

DEFINITION 4.4. *We define a* static regular expression (SRE) *on an alphabet Σ inductively, as follows.*

1. *The empty expression is an SRE, and defines $\emptyset$, the empty set.*
2. *If $x \in \Sigma$ or $x = \lambda$ (the empty string), then x is an SRE.*

3. *If s and t are SREs, then st, the concatenation of s and t, is an SRE.*
4. *If s and t are SREs, then $s + t$, the union of s and t, is an SRE.*
5. *If s is an SRE, then s^* is an SRE if and only if:*
   a) *s does not contain a union of two SREs and*
   b) *s does not contain any use of the $*$ operator.*

Note the restriction of the $*$ operator in the above definition. That is, $*$ can only be applied to a string. This is the only difference between SREs and standard regular expressions.

We can alternatively define an SRE as a regular expression that is the sum of terms, each of which is a concatenation of letters and starred strings.

THEOREM 4.5. *Let R be an unambiguous regular expression. Then $L(R)$ is sparse if and only if R is static.*

Proof. We first prove two lemmas about "forbidden" subexpressions.

LEMMA 4.6. *Let α, β, S be nonempty regular expressions such that $S = (\alpha + \beta)^*$ and S is unambiguous. Then there is a constant $k > 0$ such that, for infinitely many n, $L(S)$ contains $2^{\frac{n}{k}}$ strings of length n.*

Proof. Let $u, v \in \Sigma^*$ such that $u \in L(\alpha)$ and $v \in L(\beta)$. Let $k = |u| \cdot |v|$. Because S is unambiguous, there must be at least two strings of length k in $L(S)$, namely, $u^{|v|}$ and $v^{|u|}$. So, for any length n such that $n = ik$, $i \geq 1$, there are at least $2^i = 2^{i \cdot \frac{k}{k}} = 2^{\frac{n}{k}}$ strings of length n in $L(S)$. □

LEMMA 4.7. *Let α, β, S be nonempty regular expressions such that S is unambiguous, where S is either of the form $(\alpha^*\beta)^*$ or of the form $(\alpha\beta^*)^*$. Then, there is a constant k such that, for infinitely many n, $L(S)$ contains $2^{\frac{n}{k}}$ strings of length n.*

Proof. Let $u, v \in \Sigma^*$ such that $u \in L(\alpha)$ and $v \in L(\beta)$. Suppose $S = (\alpha^*\beta)^*$. Let $k = |u| \cdot |v| + |v|$. If S is unambiguous, there are at least two distinct strings of length k in $L(S)$, namely, $u^{|v|}v$ and $v^{|u|+1}$. So, for any length n such that $n = ik$, $i \geq 1$, there are at least $2^i = 2^{\frac{n}{k}}$ strings of length n in $L(S)$.

The proof is very similar if $S = (\alpha\beta^*)^*$ is unambiguous. □

It is clear that unambiguity is necessary for both lemmas. For example, the expression $(a + a)^*$ is not static, but $L((a + a)^*) = L(a^*)$, which is sparse.

Note that if R is the empty expression, the theorem is true, since R is static, and $L(R) = \emptyset$, which is certainly sparse. So, for the rest of the proof, we will assume that R is nonempty.

To show one direction of Theorem 4.5, suppose R is not static. Then it contains a subexpression that is either of the form $(\gamma_0(\alpha + \beta)\gamma_1)^*$ or of the form $(\gamma_0\alpha^*\gamma_1)^*$. In the first case, by a small modification to the proof of Lemma 4.6, $L(R)$ is not sparse. In the second case, by a similar modification to the proof of Lemma 4.7, $L(R)$ cannot be sparse.

Now, suppose R is static. If $R = x$, $x \in \Sigma$, $L(R)$ contains only the string x. If $R = r^*$, where r is either a string of characters or a single character, $L(R)$ can have at most one string of any length.

Suppose $R = r + s$, where r and s are SREs. Let $p_r(n)$ and $p_s(n)$ bound the number of strings in $L(r)$ and $L(s)$, respectively. Then there are at most $p_r(n) + p_s(n)$ strings of length n.

Finally, suppose $R = rs$, where r and s are SREs. Let $p_r(n)$ and $p_s(n)$ bound the number of strings in $L(r)$ and $L(s)$, respectively. Then, the number of strings of length n is:

$$q(n) \leq \sum_{i=0}^{n} p_r(i) \cdot p_s(n - i).$$

The degree of q is bounded by $1 + \text{degree}(p_r(n)) + \text{degree}(p_s(n))$. By induction on the complexity of R, $L(R)$ is sparse. □

Note that the second half of the proof does not use unambiguity. Hence, any static regular expression generates a sparse regular language.

THEOREM 4.8. *Let R be an SRE. Then $L(R)$ is* L*-printable.*

Proof. Basically, we divide R into terms that are either starred expressions or nonstarred expressions. For example, we would divide $0(1+0)10(11)^*00(0+11)$ into three parts: $0(1+0)10$, $(11)^*$, and $00(0+11)$. Then, we internally L-print each term independently, and check to see if the strings generated have the correct length. In our example, to print strings of length 9, we might generate 0110, 11, and 0011, respectively, and check that the combined string is in fact 9 characters long. (In this case, the string is too long and is not printed.)

Let k be the number of stars that appear in R. Partition R into at most $2k+1$ subexpressions, k with stars, and the others containing no stars.

The machine to L-print $L(R)$ has two types of counters. For each starred subexpression, the machine counts how many times that subexpression has been used. For a string of length n, no starred subexpression can be used more than n times. Each counter for a starred subexpression only needs to count up to n.

Each nonstarred subexpression generates only a constant number of strings. Thus, up to $k+1$ additional counters, each with a constant bound, are needed. (Note that the production may intermix the two types of counters, for instance, if $(x^* + y^*)$ occurs.)

The machine uses two passes for each potential string. First, the machine generates a current string, counting its length. If the string is the correct length, it regenerates the string and prints it out. Otherwise, it increments the set of counters and continues. In this way, all strings of lengths $\leq n$ are generated, and all strings of length n are printed.

Lastly, we need to argue that this procedure can be done by a logspace machine. Each of the at most $2k+1$ counters must count up to n (for n sufficiently large, say, larger than $|R|$). Thus, the counting can be done in $\log n$ space. In addition, the actual production of a string requires an additional counter, to store a loop variable. The rest of the computation can be handled in $O(1)$ space, using the states of the machine. Thus, $L(R)$ is L-printable.

Note that this L-printing algorithm may generate some strings in $L(R)$ more than once. To get a nonredundant L-printer, simply modify the program to output the strings in lexicographic order, as in Proposition 3.5, or use an unambiguous SRE for $L(R)$. □

Theorem 4.8 does not characterize the L-printable sets, as we see below.

PROPOSITION 4.9. *There exists a set S such that S is* L*-printable and not regular.*

Proof. The language $S = \{0^k 1^k : k \in \mathcal{N}\}$ is L-printable (for any n, we print out $0^{\frac{n}{2}}1^{\frac{n}{2}}$ only if n is even), but not regular. □

4.2. Sparse context-free languages. Using the theory of bounded context-free languages we can also show that every sparse context-free language is L-printable.

DEFINITION 4.10. *A set A is* bounded *if there exist strings $w_1, \ldots, w_k$ such that*

$$A \subseteq (w_1)^* \cdots (w_k)^*.$$

Note the similarity between bounded languages and languages generated by SREs. Note also that every bounded language is sparse.

Ibarra and Ravikumar [IR86] prove the following.

THEOREM 4.11 (see [IR86]). *If A is a context-free language then A is sparse if and only if A is bounded.* □

Ginsburg [Gin66, p. 158] gives the following characterization of bounded context-free languages.

THEOREM 4.12 (see [Gin66]). *The class of bounded context-free languages is the smallest class consisting of the finite sets and fulfilling the following properties.*

1. *If A and B are bounded context-free languages then $A \cup B$ is also a bounded context-free language.*
2. *If A and B are bounded context-free languages then $AB = \{xy \mid x \in A \text{ and } y \in B\}$ is also a bounded context-free language.*
3. *If A is a bounded context-free language and x and y are fixed strings then the following set is also a bounded context-free language:*

$$\{x^n a y^n : \ a \in A \text{ and } n \in \mathcal{N}\}.$$ □

COROLLARY 4.13. *Every bounded context-free language is* L*-printable.*

Proof. Every finite set is L-printable. The L-printable sets are closed under the three properties in Theorem 4.12. □

COROLLARY 4.14. *Every sparse context-free language is* L*-printable.* □

This completely characterizes the L-printable context-free languages. However, the sparse context-free languages do not characterize the L-printable languages.

PROPOSITION 4.15. *There exists an* L*-printable set S such that S is not context-free.*

Proof. The language $S = \{0^n 1^n 0^n : n \in \mathcal{N}\}$ is L-printable, but it is not context-free. □

5. L-isomorphisms. It is easy to show that two P-printable sets, or P-rankable sets, of similar densities are P-isomorphic. Since the usual proof relies on binary search, it does not immediately extend to L-rankable sets. However, we are able to exploit the sparseness of L-printable sets to show the following.

THEOREM 5.1. *If A and B are* L*-printable and have similar densities, then A and B are* L*-isomorphic (i.e., $A \cong_{log} B$).*

Proof. For each x, define y_x to be the image of x in the lexicographic isomorphism from A to B. Since A and B are L-printable, they are both sparse. Let $p(n)$ be a strictly increasing polynomial that bounds the densities of both sets. If $x \notin A$, then x is "close" to y_x in the sense that there are at most $p(|x|)$ strings between them in the lexicographic ordering. (Recall Definition 2.7.) In fact, for all x, $|y_x| \leq p(|x|+1)$.

Let $r_A(x)$ be the rank of x in A. If $x \notin A$, then the rank of x in $\overline{A}$ is $x - r_A(x)$. Furthermore, $x - r_A(x) = y_x - r_B(y_x)$, and y_x is the unique element of $\overline{B}$ for which this holds. Note that both $r_A(x)$ and $r_B(y_x)$ can be written in space $O(\log |x|)$. Thus, to compute y_x, we need to compute $x - r_A(x) + r_B(y_x)$. We do so by maintaining a variable d that is initialized to $r_A(x)$. Counter c is initialized to 0. The following loop is iterated until c reaches $p(|x|+1)$:

1. L-print (in lexicographic order) the elements of B of length c; for each string that is lexicographically smaller than $(x-d)$, decrement d;
2. increment c.

Output $x - d$.

Note that, if d is written on the work tape, each bit of $x - d$ can be computed in logspace as needed, and the output of the L-printing function can be compared to $x - d$ in a bit-by-bit manner.

If $x \in A$, since the L-printing function outputs strings in lexicographic order, computing y_x is easy: compute $r_A(x)$, then "L-print" B internally, actually outputting the $r_A(x)$th string.

Without loss of generality, we can assume that the simulated L-printer for B prints B in lexicographic order. Thus, as soon as the $(r_A(x) - 1)$st element of B is printed internally, the simulation switches to output mode.

The following is an overview of the logspace algorithm computing the desired isomorphism.

1. Compute $A(x)$.
2. Compute $r_A(x)$, and write it on a work tape.
3. If $x \in A$, find the $r_A(x)$th element of B, and output it.
4. If $x \notin A$, find the unique string $y_x \notin B$ such that $x - r_A(x) = y_x - r_B(y_x)$, and output y_x. □

Using this theorem, we can now characterize the L-printable sets in terms of isomorphisms to tally sets, and in terms of sets of low Kolmogorov space complexity.

THEOREM 5.2. *The following are equivalent:*

1. *S is* L*-printable.*
2. *S is* L*-isomorphic to some tally set in* L.
3. *There exists a constant k such that $S \subseteq KS[k \log n, k \log n]$ and $S \in$ L.*

Although it is not known whether or not every sparse L-rankable set is L-isomorphic to a tally set (see Theorem 6.1), we can prove the following lemma, that will be of use in the proof of Theorem 5.2.

LEMMA 5.3. *Let A be sparse and* L*-rankable. Then there exists a tally set $T \in$ L such that A and T have similar density.*

Proof. Let $A^{\leq n}$ denote the strings of length at most n in A. Let $p(n)$ be an everywhere positive monotonic increasing polynomial such that $|A^{\leq n}| \leq p(n)$ for all n, and such that $p(n) - p(n-1)$ is greater than the number of strings of length n in A. Let $r(x)$ be the ranking function of A. We define the following tally set:

$$T = \{1^{p(|x|-1)+r(x)-r(1^{|x|-1})} : x \in A\}.$$

To show that $T \in L$, notice that of the tally strings 1^i, $p(n-1) < i \leq p(n)$, $1^i \in T$ if and only if $p(n-1) < i \leq p(n-1)+r(1^i)-r(1^{i-1})$. So, to decide $T(1^m)$, we first find the largest n such that $p(n-1) < m \leq p(n)$. (Note that n can be written in binary in space $O(\log m)$.) Then compute $d_1 = m - p(n-1)$. This difference is bounded by $p(n)$, and thus can be written in logspace. Finally, compute $d_2 = r(1^n) - r(1^{n-1})$ and compare to d_1. Accept if and only if $d_1 \leq d_2$.

Finally, we show that T and A have similar density. Let $f : A \to T$ be the lexicographic isomorphism between T and A. Note that f maps strings of length n to strings of length at most $p(n)$, so f is polynomially bounded. Note that p is always positive, which implies that f is length-increasing. So, f^{-1} must also be polynomially bounded. Thus, T and A have similar density. □

The following proof of Theorem 5.2 is very similar to the proof of the analogous theorem in [AR88].

Proof. [1 ⇒ 2] Let S be L-printable. Then it is sparse and L-rankable. Let T be the tally set guaranteed by Lemma 5.3. By Proposition 4.1, T is L-printable. Thus,

T and S are L-printable, and T and S have similar density. So, by Theorem 5.1, $S \cong_{log} T$.

[$2 \Rightarrow 3$] Let S be L-isomorphic to a tally set T, and let f be the L-isomorphism from S to T. Let $x \in S$ be a string of length n. Let $f(x) = 0^r$. Since f is logspace computable, there exists a constant c such that $r \leq n^c$, i.e., $|r| \leq c \log n$. In order to recover x from r, we only have to compute $f^{-1}(0^r)$. Computing 0^r given r requires $\log n$ space for one counter. Further, there exists a constant l such that computing $f^{-1}(0^r)$ requires at most $lc \log n$ space, since $r \leq n^c$. So, the total space needed to compute x given r is less than or equal to $\log n + lc \log n \leq k \log n$ for some k. Hence, $S \subseteq KS[k \log n, k \log n]$. If $T \in$ L, then $S \in$ L, since $S \cong_{log} T$.

[$3 \Rightarrow 1$] Assume $S \subseteq KS[k \log n, k \log n]$ for some k, and $S \in$ L. On input 0^n, we simulate M_u for each string of length $k \log n$. For a given string x, $|x| = k \log n$, we first simulate $M_u(x)$ and check whether it completes in space $k \log n$. If it does, we recompute $M_u(x)$, this time checking whether the output is of length n and in S. If it is, we recompute $M_u(x)$ and print out the result. The entire computation only needs $O(\log n)$ space, so S is L-printable. □

It was shown in [AR88] that a set has small generalized Kolmogorov complexity if and only if it is P-isomorphic to a tally set. (Note: this was an improvement of the result in [BB86], which showed that a set has small generalized Kolmogorov complexity if and only if it is "semi-isomorphic" to a tally set.) Using a similar argument and Theorem 5.2 we can show an analogous result for sets with small generalized Kolmogorov space complexity. First, we prove the following result.

PROPOSITION 5.4. *For all M_v and k, $KS_v[k \log n, k \log n]$ is L-printable.*

Proof. To L-print for length n, simulate M_v on each string of length less than or equal to $k \log n$ and output every string of length n produced. □

COROLLARY 5.5. *There exists a k such that $A \subseteq KS[k \log n, k \log n]$ if and only if A is L-isomorphic to a tally set.*

Proof. Suppose A is L-isomorphic to a tally set. Then, by the argument given in the proof of [$2 \Rightarrow 3$] in Theorem 5.2, $A \subseteq KS[k \log n, k \log n]$.

Now, suppose $A \subseteq KS[k \log n, k \log n]$. By Proposition 5.4 and Theorem 5.2, $KS[k \log n, k \log n]$ is L-isomorphic to a tally set in L via some L-isomorphism f. It is clear that A is L-isomorphic to $f(A)$. Since $f(A)$ is a subset of a tally set, $f(A)$ must also be a tally set. □

6. Printability, rankability, and decision. In this section we examine the relationship among L-printable sets, L-rankable sets, and L-decidable sets. We show that any collapse of these classes, even for sparse sets, is equivalent to some unlikely complexity class collapse.

THEOREM 6.1. *The following are equivalent:*

1. *Every sparse* L*-rankable set is* L*-printable.*
2. *There are no tally sets in* P − L.
3. *E = LinearSPACE.*

Proof. [$2 \Leftrightarrow 3$] This equivalence follows from techniques similar to those of Book [Boo74].

[$2 \Rightarrow 1$] Suppose A is a sparse L-rankable set. Note that $A \in$ L.

Let

$$T = \{1^{\langle i,j \rangle} : \text{ The } i\text{th bit of the } j\text{th string in } A \text{ is } 1\},$$

where

$$\langle i, j\rangle = \frac{(i+j)(i+j+1)}{2} + i.$$

Note that $\langle i, j\rangle$ can be computed in space linear in $|i| + |j|$. Since A is sparse, i and j are bounded by a polynomial in the length of the jth string. Hence, $\langle i, j\rangle$ can be computed using logarithmic space with respect to the length of the jth string.

Given $\langle i, j\rangle$, we can determine i and j in polynomial time, and we can find the jth string of A by using binary search and the ranking function of A. Hence, $T \in$ P. So, by assumption, $T \in$ L.

Next we give a method for printing A in logspace. Given a length n, we compute (and store) the ranks of 0^n and 1^n in A. Let r_{start} and r_{end} be the ranks of 0^n and 1^n, respectively. If $0^n \notin A$, the string with rank r_{start} has length less than n. First, we check to see if $0^n \in A$, and if so, print it. Then, for each j, $r_{start} < j \leq r_{end}$, we output the jth string by computing and printing $T(1^{\langle i,j\rangle})$ for each bit i. This procedure prints the strings of A of length n.

Note that since A is sparse, we can store r_{start} and r_{end} in $O(\log n)$ space. Since $i \leq n$, we can also store and increment the current value of i in $\log n$ space.

[1 $\Rightarrow$ 2] Let $T \in$ P be a tally set. Since the monotone circuit value problem is P-complete (see [GHR95]), there exists a logspace computable function f and a nondecreasing polynomial p such that $f(n)$ produces a circuit C_n with the following properties.

1. C_n is monotone (i.e., C_n uses only AND and OR gates).
2. C_n has $p(n)$ gates.
3. The only inputs to C_n are 0 and 1.
4. C_n outputs 1 if and only if 1^n is in T.

We can assume that the reduction orders the gates of C_n so that the value of gate g_i depends only on the constants 0 and 1 and the values of gates g_j for $j < i$ [GHR95]. Let x_n be the string of length $p(n)$ such that the ith bit of x_n is the value of gate g_i.

Let $A = \{x_n : n \in \mathcal{N}\}$. Then A contains exactly one string of length $p(n)$ for all n and no strings of any other lengths.

Claim 6.1.1. *The set A is* L*-rankable.*

Proof. To prove this claim, let w be any string. In logspace, we can find the greatest n such that $p(n) \leq |w|$. If $p(n) \neq |w|$ then $w \notin A$, and the rank of w is n. Suppose $|w| = p(n)$. Since x_n is the only string of length $p(n)$ in A, the rank of w is $n-1$ if $w < x_n$, and n otherwise.

Consider the ith bit of w as a potential value for gate g_i in C_n. Let j be the smallest value such that w_j is not the value of g_j. In order to find the value of a gate g_i, we first use $f(n)$ (our original reduction) to determine the inputs to g_i. By the time we consider the ith bit of w, we know that w is a correct encoding of all of the gates g_k such that $k < i$, so we can use those bits of w as the values for the gates. Thus, we can determine the value of g_i and compare it to the ith bit of w. If they differ, we are done. If they are the same, we continue with the next gate. We can count up to $p(n)$ in logspace, so this whole process needs only $O(\log p(n))$ space to compute.

Once j is found, there are three cases to consider.

1. If j doesn't exist then $w = x_n$.
2. If the jth bit of w is 0 then $w < x_n$.
3. If the jth bit of w is 1 then $w > x_n$.

These follow since the ith bit of x_n matches the ith bit of w for all $i < j$. □

Thus A is L-rankable and, by assumption, L-printable.

So, to determine if 1^n is in T, L-print A for length $p(n)$ to get x_n. The bit of x_n that encodes the output gate of C_n is 1 if and only if $1^n \in T$. Since every step of this algorithm is computable in logspace, $T \in L$.

This completes the proof of Theorem 6.1. □

COROLLARY 6.2. *There exist two non-*L*-isomorphic* L*-rankable sets of the same density, unless there are no tally sets in* P − L.

Proof. Consider the sets T and A from the second part of the proof of Theorem 6.1. The set $B = \{1^{p(n)} : n \in \mathcal{N}\}$ has the same density as A. By Proposition 4.1, B is L-printable. If A and B were L-isomorphic then by Proposition 3.6, A would also be L-printable and T would be in L. □

One may wonder whether every sparse set in L is L-printable or at least L-rankable. We show that either case would lead to the unlikely collapse of FewP and L. Recall that FewP consists of the languages in NP accepted by nondeterministic polynomial-time Turing machines with at most a polynomial number of accepting paths.

Fix a nondeterministic Turing machine M and an input x. Let p specify an accepting path of $M(x)$ represented as a list of configurations of each computation step along that path. Note that in logarithmic space we can verify whether p is such an accepting computation since if one configuration follows another only a constant number of bits of the configuration change.

We can assume without loss of generality that all paths have the same length and that no accepting path consists of all zeros or all ones.

Define the set P_M by

$$P_M = \{x\#p : p \text{ is an accepting path of } M \text{ on } x\}.$$

From the above discussion we have the following proposition that we will use in the proofs of Theorems 6.6 and 6.7.

PROPOSITION 6.3. *For any nondeterministic machine M, P_M is in* L. □

Allender and Rubinstein [AR88] showed the following about P-printable sets.

THEOREM 6.4 (see [AR88]). *Every sparse set in* P *is* P*-printable if and only if there are no sparse sets in* FewP − P. □

Allender [All86] also relates this question to inverting functions.

DEFINITION 6.5. *A function f is* strongly L-invertible *on a set S if there exists a logspace computable function g such that for every $x \in S$, $g(x)$ prints out all of the strings y such that $f(y) = x$.*

We extend the techniques of Allender [All86] and Allender and Rubinstein [AR88] to show the following.

THEOREM 6.6. *The following are equivalent.*

1. *There are no sparse sets in* FewP − L.
2. *Every sparse set in* L *is* L*-printable.*
3. *Every sparse set in* L *is* L*-rankable.*
4. *Every* L*-computable, polynomial-to-one, length-preserving function is strongly* L*-invertible on* $\{1\}^*$.
5. FewE = LinearSPACE.

Proof. [1 ⇒ 2] Let A be a sparse set in L. Then A is in P. By (1) we have that there are no sparse sets in FewP − P. By Theorem 6.4, A is P-printable.

Consider the following set B:

$$B = \{1^{\langle n,i,j,b\rangle} : \text{ the } i\text{th bit of the } j\text{th element of } A \text{ of length } n \text{ is } b\}.$$

Since A is P-printable then B is in P. By statement 1 (as B is sparse and in P $\subseteq$ FewP), we have that B is in L. Then A is L-printable by reading the bits off from B.

[$2 \Rightarrow 3$] Follows immediately from Proposition 3.4.

[$3 \Rightarrow 1$] Let A be a sparse set in FewP accepted by a nondeterministic machine M with computation paths of length $q(n)$ for inputs of length n.

Consider the set P_M defined as above. Note that P_M is sparse since for any length n, M only accepts a polynomial number of strings with at most a polynomial number of accepting paths each. Also, by Proposition 6.3, we have P_M in L.

By statement 3 we have that P_M is L-rankable. We can then determine in logarithmic space whether $M(x)$ accepts (and thus x is in A) by checking whether

$$r_{P_M}(x\#0^{q(|x|)}) < r_{P_M}(x\#1^{q(|x|)}).$$

[$2 \Rightarrow 4$] Let f be an L-computable, polynomial-to-one, length-preserving function. Consider $S = \{y : f(y) \in \{1\}^*\}$. Since S is in L, S is L-printable.

[$4 \Rightarrow 2$] Let A be a sparse set in L. Define $f(x) = 1^{|x|}$ if x is in A and x otherwise. If g is a strong L-inverse of f on $\{1\}^*$ then $g(1^n)$ will print out the strings of length n of A and 1^n. We can then print out the strings of length n in logspace by printing the strings output by $g(1^n)$, except we print 1^n only if 1^n is in A.

[$1 \Leftrightarrow 5$] In [RRW94], Rao, Rothe, and Watanabe show that there are no sparse sets in FewP $-$ P if and only if FewE = E. A straightforward modification of their proofs is sufficient to show that there are no sparse sets in FewP $-$ L if and only if FewE = LinearSPACE. □

Unlike L-printability, L-rankability does not imply sparseness. One may ask whether every set computable in logarithmic space may be rankable. We show this equivalent to the extremely unlikely collapse of PP and L.

THEOREM 6.7. *The following are equivalent.*

1. *Every* #P *function is computable in logarithmic space.*
2. L = PP.
3. *Every set in* L *is* L*-rankable.*

Our proof uses ideas from Blum (see [GS91]), who shows that every set in P is P-rankable if and only if every #P function is computable in polynomial time. Note that Hemachandra and Rudich [HR90] proved results similar to Blum's.

Proof. [$1 \Rightarrow 2$] If A is in PP then there is a #P function f such that x is in A if and only if the high-order bit of $f(x)$ is 1.

[$2 \Rightarrow 1$] Note that L = PP implies that P = PP implies that P = P^{PP} implies that P = $\mathrm{P}^{\#\mathrm{P}}$. Thus we have L = $\mathrm{P}^{\#\mathrm{P}}$ and we can compute every bit of a #P function in logarithmic space.

[$1 \Rightarrow 3$] Let A be in L. Consider the nondeterministic polynomial-time machine M that on input x guesses a $y \leq_{lex} x$ and accepts if y is in A. The number of accepting paths of $M(x)$ is a #P function equal to $r_A(x)$.

[$3 \Rightarrow 1$] Let f be a #P function. Let M be a nondeterministic polynomial-time machine such that $f(x)$ is the number of accepting computations of $M(x)$. Let $q(n)$ be the polynomial-sized bound on the length of the computation paths of M. Consider P_M as defined above. By Proposition 6.3 we have that P_M is in L, so by (3) P_M is L-rankable. We then can compute $f(x)$ in logarithmic space by noticing

$$f(x) = r_{P_M}(x\#1^{q(|x|)}) - r_{P_M}(x\#0^{q(|x|)}). \qquad \square$$

7. Conclusions. The class of L-printable sets has many properties analogous to its polynomial-time counterpart. For example, even without the ability to do binary searching, one can show that two L-printable sets of the same density are isomorphic. However, some properties do not appear to carry over: it is very unlikely that every sparse L-rankable set is L-printable.

Despite the strict computational limits on L-printability, this class still has some bite: every tally set in L, every sparse regular and context-free language, and every L-computable set of low space-bounded Kolmogorov complexity strings is L-printable.

Acknowledgments. The authors want to thank David Mix Barrington for a counterexample to a conjecture about sparse regular sets, Alan Selman for suggesting the tally set characterization of L-printable sets and Corollary 5.5, Chris Lusena for proofreading, and Amy Levy, John Rogers, and Duke Whang for helpful discussions. The simple proof sketch of Theorem 4.3 was provided by an anonymous referee. The last equivalence of Theorem 6.6 was suggested by another anonymous referee. The authors would like to thank both referees for many helpful suggestions and comments.

REFERENCES

[All86] E. Allender, *The complexity of sparse sets in P*, in Proc. Conference on Structure in Complexity Theory, A. Selman, ed., Springer-Verlag, Berlin, New York, 1986, pp. 1–11.

[AR88] E. Allender and R. Rubinstein, *P-printable sets*, SIAM J. Comput., 17 (1988), pp. 1193–1202.

[BB86] J. Balcazar and R. Book, *Sets with small generalized Kolmogorov complexity*, Acta Inform., 23 (1986), pp. 679–688.

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity* I, Springer-Verlag, Berlin, New York, 1988.

[BEGO71] R. Book, S. Even, S. Greibach, and G. Ott, *Ambiguity in graphs and expressions*, IEEE Trans. Comput., C-20 (1971), pp. 149–153.

[Boo74] R. Book, *Tally languages and complexity classes*, Inform. and Control, 26 (1974), pp. 186–193.

[Gin66] S. Ginsburg, *The Mathematical Theory of Context-Free Languages*, McGraw-Hill, New York, 1966.

[GHR95] R. Greenlaw, H. J. Hoover, and W. Ruzzo, *Limits to Parallel Computation: P-Completeness Theory*, Oxford University Press, New York, 1995.

[GS91] A. Goldberg and M. Sipser, *Compression and ranking*, SIAM J. Comput., 20 (1991), pp. 524–536.

[Har83] J. Hartmanis, *Generalized Kolmogorov complexity and the structure of feasible computations*, in Proc. 24th IEEE Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1983, pp. 439–445.

[HH88] J. Hartmanis and L. Hemachandra, *On sparse oracles separating feasible complexity classes*, Inform. Process. Lett., 28 (1988), pp. 291–295.

[HR90] L. Hemachandra and S. Rudich *On the complexity of ranking*, J. Comput. System Sci., 41 (1990), pp. 251–271.

[HY84] J. Hartmanis and Y. Yesha, *Computation times of NP sets of different densities*, Theoret. Comput. Sci., 34 (1984), pp. 17–32.

[IR86] O. Ibarra and B. Ravikumar, *On sparseness, ambiguity and other decision problems for acceptors and transducers*, in Proc. 3rd Annual Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, Berlin, New York, 1986, pp. 171–179.

[JK89] B. Jenner and B. Kirsig, *Alternierung und Logarithmischer Platz*, Ph.D. Thesis, Universität Hamburg, Hamburg, Germany, 1989.

[Mar91] J. Martin, *Introduction to Languages and the Theory of Computation*, McGraw-Hill, New York, 1991.

[Pap94] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, New York, 1994.

[RRW94] R. P. N. Rao, J. Rothe, and O. Watanabe, *Upward separation for FewP and related classes*, Inform. Process. Lett., 52 (1994), pp. 175–180.

[Rub86] R. Rubinstein, *A Note on Sets with Small Generalized Kolmogorov Complexity*, Technical Report TR 86-4, Iowa State University, Ames, IA, March 1986.

[SH85] R.E. Stearns and H.B. Hunt III, *On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata*, SIAM J. Comput., 14 (1985), pp. 598-611.

[Sip83] M. Sipser, *A complexity theoretic approach to randomness,* in Proc. 15th ACM Symposium on Theory of Computing, ACM, New York, 1983, pp. 330–335.

[SSYZ92] J. Shallit, A. Szilard, S. Yu, and K. Zhang, *Characterizing regular languages with polynomial densities*, in Proc. 17th International Symposium on Mathematical Foundations of Computer Science, Springer-Verlag, Berlin, New York, 1992, pp. 494–503.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 152–163

THE INVERSE SATISFIABILITY PROBLEM*

DIMITRIS KAVVADIAS† AND MARTHA SIDERI‡

Abstract. We study the complexity of telling whether a set of bit-vectors represents the set of all satisfying truth assignments of a Boolean expression of a certain type. We show that the problem is coNP-complete when the expression is required to be in conjunctive normal form with three literals per clause (3CNF). We also prove a dichotomy theorem analogous to the classical one by Schaefer, stating that, unless P=NP, the problem can be solved in polynomial time if and only if the clauses allowed are all Horn, or all anti-Horn, or all 2CNF, or all equivalent to equations modulo two.

Key words. computational complexity, polynomial-time algorithms, coNP-completeness, Boolean satisfiability, model

AMS subject classifications. 11Y16, 68Q20, 68Q25, 68T30

PII. S0097539795285114

1. Introduction. Logic deals with logical formulae, and more particularly with the *syntax* and the *semantics* of such formulae, as well as with the interplay between these two aspects [CK90]. In the domain of *Boolean logic*, for example, a Boolean formula ϕ may come in a variety of syntactic classes—conjunctive normal form (CNF), its subclasses 3CNF, 2CNF, Horn, etc.— and its semantics is captured by its *models* or satisfying truth assignments, that is, the set $\mu(\phi)$ of all truth assignments that satisfy the formula (see Figure 1 for an example).

Going back and forth between these two representations of a formula is therefore of interest. One direction has been studied extensively from the standpoint of computational complexity: going from ϕ to $\mu(\phi)$. In particular, telling whether $\mu(\phi) = \emptyset$ is the famous *satisfiability problem* (SAT), which is known to be NP-complete in its generality and its special case 3SAT, among others, and polynomial-time solvable in its special cases Horn, 2SAT, and exclusive-or [Co71, Sc78, Pa94]. All in all, this direction is a much-studied computational problem. *In this paper we study, and in a certain sense completely settle, the complexity of the inverse problem, that is, going from $\mu(\phi)$ back to ϕ.* That is, for all the syntactic classes mentioned above, we identify the complexity of telling, given a set M of models, whether there is a formula ϕ in the class (3SAT, Horn, etc.) such that $M = \mu(\phi)$. We call this problem *inverse satisfiability.*

Besides its fundamental nature, there are many more factors that make inverse satisfiability a most interesting problem. A major motivation comes from AI (in fact, what we call here the inverse satisfiability problem is implicit in much of the recent AI literature [Ca93, DP92, KKS95, KKS93, KPS93]). A set of models such as those in Figure 1(b) can be seen as a *state of knowledge.* That is, it may mean that at present, for all we know, the state of our three-variable world can be in any one of the three states indicated. In this context, formula ϕ is some kind of *knowledge representation.* In AI there are many sophisticated competing methods for knowledge representation

*Received by the editors April 24, 1995; accepted for publication (in revised form) November 20, 1996; published electronically June 15, 1998. This work was partially supported by the Esprit Project ALCOM II and the Greek Ministry of Research (ΠΕΝΕΔ program 91EΔ648).
http://www.siam.org/journals/sicomp/28-1/28511.html
†Department of Mathematics, University of Patras, Patras, Greece (djk@math.upatras.gr).
‡Department of Computer Science, Athens University of Economics and Business, Athens, Greece (mss@dias.aueb.gr).

$$\phi = (x \vee y \vee z) \wedge (\overline{x} \vee \overline{y}) \wedge (y \vee \overline{z})$$

(a)

$$\mu(\phi) = \{011, 010, 100\}$$

(b)

FIG. 1. *A Boolean formula in* 3*CNF* (a), *and the corresponding set of models* (b).

(Boolean logic is perhaps the most primitive; see [GN87, Le86, Mc80, Mo84, Re80, SK90]), and it is important to understand the expressibility of each. This is a form of the inverse satisfiability problem.

The inverse satisfiability problem was also proposed in [DP92] as a form of *discovering structure in data*. For example, establishing that a complex binary relation is the set of models of a simple formula may indeed uncover the true structure and nature of the heretofore meaningless table. [DP92] only address this problem in certain fairly straightforward cases. The problem of *learning* a formula [AFP92] can be seen as a generalization of the inverse satisfiability problem.

A recent trend in AI is to *approximate* complex formulae by simple ones, such as Horn formulae [SK91, KPS93, GPS94]. Quantifying the quality and computational feasibility of such approximations also involves understanding the inverse satisfiability problem.

The basic computational problem we study is this: given a set of models M, is there a CNF formula ϕ with at most three literals per clause, such that $M = \mu(\phi)$? We call this problem INVERSE 3SAT. Our first result is that *INVERSE* 3*SAT is coNP-complete* (Theorem 1).

Note. INVERSE 3SAT, as well as all other problems we consider in this paper, can be solved in polynomial time if the given $m \times n$ table M has $m = 2^{\Theta(n)}$, that is, if there are exponentially many models in M. The interesting cases of the problem are therefore when $m = 2^{o(n)}$.

There are three well-known tractable cases of SAT: 2SAT (all clauses have two literals), HORNSAT (all clauses are Horn, with at most one positive literal each, and its symmetric case of *anti-Horn formulae*, in which all clauses have at most one *negative* literal), and XORSAT (the clauses are equations modulo two). Schaefer's elegant *dichotomy theorem* [Sc78] states that, unless P=NP, in a certain sense *these are precisely the only tractable cases of SAT.* Interestingly, the inverse problem for these three cases happens to also be tractable! That is, we can tell in polynomial time if a set of models is the set of models of a Horn (or anti-Horn) formula, of a 2CNF formula, or of an exclusive-or formula (interestingly, the latter two results were in fact pointed out by Schaefer himself [Sc78], while the first, left open in [Sc78], is from [DP92, KPS93]). The question comes to mind: are there other tractable cases of the inverse problem? Our Theorem 2 answers this in the negative; rather surprisingly, a strong dichotomy theorem similar to Schaefer's holds for the inverse satisfiability problem as well, in that the problem is coNP-complete for all syntactic classes of CNF formulae except for the cases of Horn (and anti-Horn), 2CNF, and exclusive-or. The proof of our dichotomy theorem draws from both that of Theorem 1 and Schaefer's proof, and in fact strengthens Schaefer's main expressibility result (Theorem 3.0 in [Sc78]).

2. Definitions. Most of the nonstandard terminology used in this paper comes from [Sc78].

Let $\{x_1, \ldots, x_n\}$ be a set of Boolean variables. A literal is a variable or its negation. A model is a vector in $\{0,1\}^n$, intuitively a truth assignment to the Boolean variables. We denote by $\vee$ and $\wedge$ the logical *or* and *and*, respectively. We also extend this notation to bitwise operations between models. If t is a model, we denote by t_i the constant (i.e., 0 or 1) in the ith position of t.

A k-place logical relation is a subset of $\{0,1\}^k$ (k integer). We use the notation $[\phi]$, where ϕ is a Boolean formula, to denote the relation defined by ϕ when the variables are taken in lexicographic order. Let R be a logical relation. Call R *Horn* if it is logically equivalent to a conjunction of clauses, each with at most one positive literal. We call it *anti-Horn* if it is equivalent to a conjunction of clauses with at most one negative literal. We call it *2CNF* if it is equivalent to a 2CNF expression. Finally, we call it *affine* if it is the solution of a system of equations in the two-element field.

Let $S = \{R_1, \ldots, R_m\}$ be a set of Boolean relations. An *S-clause* (of arity k) is an expression of the form $R(a_1, \ldots, a_k)$, where R is a k-ary relation in S and the a_i's are either Boolean literals or constants (0 or 1). Given a truth assignment, we consider an S-clause to be true if the combination of the constants, if any, and the values assigned to the variables form a tuple in R. Define an *S-formula* to be any conjuction of S-clauses defined by the relations in S.

The generalized satisfiability problem is the problem of deciding whether a given S-formula is satisfiable. Schaefer's dichotomy theorem [Sc78] states that the satisfiability of an S-formula can be decided in polynomial time in each of the following cases: (a) all relations in S are Horn, (b) all relations in S are anti-Horn, (c) all relations in S are 2CNF, (d) all relations in S are affine. In all other cases the problem is NP-complete. That is, Schaefer's result totally characterizes the complexity of the CNF satisfiability problem where in addition, the clauses are allowed to be arbitrary relations of bounded arity. It is interesting to note that several restricted forms of SAT such as ONE-IN-THREE 3SAT, NOT-ALL-EQUAL 3SAT etc., all follow as special cases of generalized satisfiability (see [GJ79, Pa94]). To make this point more clear, notice that the problem ONE-IN-THREE 3SAT can be considered as a set of four 3-ary relations $S = \{R_1, \ldots, R_4\}$. The first relation is $\{\{1,0,0\},\{0,1,0\},\{0,0,1\}\}$ and corresponds to the S-clause $R_1(x_1, x_2, x_3)$, the second relation is $\{\{0,0,0\},\{1,1,0\},\{1,0,1\}\}$ and corresponds to the S-clauses with one negated literal, e.g., $R_2(\overline{x_1}, x_2, x_3)$, and so on.

For any Boolean formula ϕ we denote by $\mu(\phi)$ its set of models. We say that a set of models M is *a 3CNF set* (kCNF in general) if there is a formula ϕ in 3CNF (respectively, kCNF) such that $M = \mu(\phi)$. Notice that for any model set M we can construct a kCNF formula that has M as its model set, but in general, this may require extra existentially quantified variables.

Based on the above we define the *INVERSE SAT problem for a set of relations* S as follows.

Given a set $M \subseteq \{0,1\}^n$, is there a conjunction of S-clauses over n variables that has M as its set of models?

Our main result states that if the relations fall in each of the four cases above, the INVERSE SAT problem is also polynomial. Otherwise it is coNP-complete.

Notice that we have excluded S from being part of the instance since we want to emphasize that INVERSE SAT is actually a collection of infinitely many subproblems. This means that all relations of S are of constant arity. Otherwise, relations of non-

constant arity could have exponentially many tuples and the problem becomes trivially intractable.

In the next section we prove that the INVERSE SAT problem is coNP-complete for 3CNF formulas. This proof includes the main construction that will be used in the proof of the main theorem in the last section. This last proof makes use of an expressibility result which is interesting on its own and partially relies on Schaefer's main theorem but with several interesting extensions.

3. coNP-completeness of inverse 3SAT. We begin this section with a technical definition that will be used throughout the paper.

DEFINITION. *Let n be a positive integer and let $M \subseteq \{0,1\}^n$ be a set of Boolean vectors. For $k > 1$, we say that a Boolean vector $m \in \{0,1\}^n$ is k-compatible with M if for any sequence of k positions $0 \leq i_1 < \cdots < i_k \leq n$, there exists a vector in M that agrees with m in these k positions.*

The above definition implies that a vector $m \in \{0,1\}^n$ is not k-compatible with a set of Boolean vectors M if there exists a sequence of k positions in m that does not agree with any vector of M. The following is a useful characterization of kCNF sets.

LEMMA 1. *Let $M \subseteq \{0,1\}^n$ be a set of models. Then the following are equivalent.*

(a) *M is a kCNF set.*

(b) *If $m \in \{0,1\}^n$ is k-compatible with M, then $m \in M$.*

Proof. Let ϕ_M be the conjuction of all possible kCNF clauses defined on n variables and satisfied by all models in M. Notice that ϕ_M is the most restricted kCNF formula (in terms of its model set) which is satisfied by all models in M. Hence if (a) holds, $M = \mu(\phi_M)$. Let $m \in \{0,1\}^n$ and $m \notin M$. Then m does not satisfy at least one clause of ϕ_M and concequently disagrees with all models in M in the same k positions corresponding to the variables in the clause, that is, m is not k-compatible with M.

Conversely, assume that any model not in M is not k-compatible with M. Then $m \notin M$ means m does not satisfy ϕ_M: m differs from all members of M in some k positions, so the k-clause indicating the complement of m in those k positions is in ϕ_M, and m does not satisfy ϕ_M. So $M = \mu(\phi_M)$ and M is a kCNF set. □

The INVERSE 3SAT problem is this: given a set of models M, is it a 3CNF set? We now state our first complexity result.

THEOREM 1. *INVERSE 3SAT is coNP-complete.*

Proof. Lemma 1 establishes that the problem is in coNP: given a set M of models, in order to prove that it is *not* a 3CNF set, it suffices to produce a model $m \notin M$ that is 3-compatible with M (obviously, 3-compatibility can be checked in polynomial time). Alternatively, given M, we immediately have a candidate 3CNF formula ϕ_M: the conjunction of all 3CNF clauses that are satisfied by all models in M. Thus M is *not* a 3CNF set iff there is a model not in M that satisfies ϕ_M.

To prove coNP-completeness, we shall reduce the following well-known coNP-complete problem to INVERSE 3SAT: given a 3CNF formula, is it unsatisfiable? Given a 3CNF formula ψ with $n \geq 4$ variables and c clauses, we shall construct a set of models M such that M is 3CNF iff ψ is unsatisfiable.

The set M will contain $k = 8\binom{n}{3} - c$ models, one for each set W of three variables, and each truth assignment T to these three variables *that does not contradict a clause of* ψ (since we may assume that ψ consists of clauses that have exactly three literals each). Let W be a set of three variables chosen among the variables $\{x_1, \ldots, x_n\}$ of formula ψ, and let $T : W \mapsto \{0,1\}$ be a truth assignment to the variables of W, such that ψ does not contain a clause not satisfied by T. Consider some total order among

the pairs (W,T), say the lexicographic one. The set M will contain a model $m_{W,T}$ for each W and T and no other model.

Every boolean vector $m_{W,T}$ is a concatenation $m_{W,T} = \tau^T_W(x_1)\cdots\tau^T_W(x_n)$ of the encodings $\tau^T_W(x_i)$ for each variable x_i occuring in the formula ψ. The encoding $\tau^T_W(x)$ of a variable x is a Boolean vector of length $k+2$ and is defined as follows:

$$\tau^T_W(x) = \begin{cases} 01\overbrace{0\cdots0\,0\cdots0}^{k \text{ positions}} & \text{if } x \in W \text{ and } T(x)=1,\\ 10\,0\cdots0\,0\cdots0 & \text{if } x \in W \text{ and } T(x)=0,\\ 00\underbrace{0\cdots0}_{i-1}1\cdots1 & \text{if } x \notin W, \end{cases}$$

where (W,T) is the ith pair in the total order mentioned above. Notice that if $x \in W$, the value of x in T is determined by the first two positions of τ^T_W: the code 01 stands for the value 1, and the code 10 stands for x being 0. In these two cases we call the string $\tau^T_W(x)$ a *value pattern*. When $x \notin W$, the code 00 in the first two positions denotes the absence of x from W, while the rest of the string uniquely determines the pair (W,T). In this case we call the string $\tau^T_W(x)$ a *padding pattern*. Notice that by our construction in a vector $m_{W,T} = \tau^T_W(x_1)\cdots\tau^T_W(x_n)$ there are exactly $n-3$ occurences of the unique padding pattern for (W,T), while the remaining three are value patterns. Hence, the length of each Boolean vector $m_{W,T}$ is $n(k+2)$; therefore, there is no exponential blow-up in the construction of the set M.

The proof of Theorem 1 now rests on the next claim.

Claim. *There is a model not in M that is 3-compatible with M if and only if ψ is satisfiable.*

Proof of the claim. For the moment, consider a Boolean vector $m = m_1\cdots m_n$, where the length of each substring m_i equals $k+2$. It is obvious that if the model m is 3-compatible with M, then it is 3-compatible in the positions restricted to one substring $m_i = m_{i1}\cdots m_{i(k+2)}$. That is, if we take three arbitrary positions of m_i, there is a vector $m_{W,T}$ in M that agrees with m_i in these three positions. The 3-compatibility of m_i with M also implies something stronger: that there is a vector $m_{W,T} \in M$ which contains a substring $\tau^T_W(x_i)$ identical to m_i. To see this, first assume that m_i does not have the value 1 in any position j for $3 \le j \le k+2$. Then 3-compatibility forces m_i to have the values 0 and 1 or 1 and 0 in the first and second positions; i.e., m_i is a value pattern. Now, if m_i has the values 0 and 1 in positions $j-1$ and j, $3 \le j \le k+2$, then the values in any triple of positions that includes positions $j-1$ and j can only agree with the values in the same positions of a specific model of M, namely, the one having the padding pattern with 0 in position $j-1$ and 1 in position j. Therefore, m_i is identical to this padding pattern. In this case, however, an analogous observation shows that the whole 3-compatible model m is identical to the model of M that has this pattern. So if m is 3-compatible with M, either it is in M or it consists of value patterns only.

Assume now that there exists a model $m \notin M$ that is 3-compatible with M. As already proved, this model $m = m_1\cdots m_n$ consists only of value patterns m_i. Model m encodes a satisfying truth assignment to the variables of ψ. For suppose it did conflict with a clause c of ψ over variables $\{x_i, x_j, x_\ell\}$. Consider the three value patterns m_i, m_j, m_ℓ of m in the positions of the variables of c. Since m is 3-compatible with M and each value pattern contains only one 1, we can conclude that there exists a model $m_{W,T} = \tau^T_W(x_1)\cdots\tau^T_W(x_n)$ in M, which encodes a truth assignment T to the set of variables $W = \{x_i, x_j, x_\ell\}$ such that $\tau^T_W(x_i) = m_i$, $\tau^T_W(x_j) = m_j$, and $\tau^T_W(x_\ell) = m_\ell$.

But since by construction T does not contradict a clause of ψ, we couldn't have conflicted with a clause of ψ. Therefore, the Boolean vector $m = m_1 \cdots m_n$ is an encoding of a satisfying assignment to the variables for formula ψ: string m_i is an encoding of the truth value assigned to the variable x_i for each $i = 1, \ldots, n$. Hence, formula ψ is satisfiable since every clause of ψ is satisfied by the truth assignment described by vector m.

Conversely, assume that ψ is satisfiable; i.e., there exists a satisfying truth assignment s for the variables $\{x_1, \ldots, x_n\}$. Construct the model $m = m_1 \cdots m_n$ as a concatenation of value patterns, where every string m_i is defined as follows:

$$m_i = \begin{cases} 01\ \overbrace{0\cdots0}^{k \text{ positions}} & \text{if } s(x_i) = 1, \\ 10\ 0\cdots0 & \text{if } s(x_i) = 0. \end{cases}$$

Obviously, model m is not included in the set M, since every model in M contains a padding pattern. Suppose that m is not 3-compatible with M. In this case m contains three positions that do not agree with any model in M. Since m is a concatenation of value patterns, it must contain three substrings m_i, m_j, m_ℓ that represent a truth assignment T for the set of variables $W = \{x_i, x_j, x_\ell\}$ such that the pair (W, T) is not encoded in any model of M. All $\binom{n}{3}$ sets of variables are, however, examined during the construction of M, and the only truth assignments that are not encoded are those conflicting with a clause of ψ. Since T does not conflict with any clause—because it is a restriction of s to three variables—we conclude that the pair (W, T) is encoded in some model of M. Hence, m is 3-compatible with M. So, if ψ is satisfiable, there exists a model 3-compatible with M, specifically the model encoding a satisfying truth assignment. □

4. The dichotomy theorem. Our main result is the following generalization of Theorem 1.

THEOREM 2. *The INVERSE SAT problem for S is in PTIME in each of the following cases.*

(a) *All relations in S are Horn.*
(b) *All relations in S are anti-Horn.*
(c) *All relations in S are 2CNF.*
(d) *All relations in S are affine.*

In all other cases, the INVERSE SAT problem for S is coNP-complete.

[Sc78] proves a surprisingly similar dichotomy theorem for SAT: SAT is in PTIME for all of these four classes, and NP-complete otherwise. Our proof is based on an interesting extension of Schaefer's main result, explained below.

DEFINITION. *Let S be a set of Boolean relations and let R be another Boolean relation, of arity r. We say that S faithfully represents R if there are binary Boolean functions $f_1, \ldots, f_s$ such that there is a conjunction of S-clauses over the variables $x_1, \ldots, x_{r+s}$ which is logically equivalent to the formula*

$$R(x_1, \ldots, x_r) \wedge \bigwedge_{\ell=1}^{s} \big(x_{r+\ell} \equiv f_\ell(x_{i_\ell}, x_{j_\ell})\big),$$

for some $i_\ell, j_\ell < r + \ell$, $\ell = 1, \ldots, s$. That is, S-clauses can express R with the help of uniquely defined auxiliary variables.

This is a substantial restriction of Schaefer's notion of "represents," which allows arbitrary existentially quantified conjunctions of S-clauses (our definition only allows

quantifiers which are logically equivalent to $\exists!x$). Hence our main technical result below extends the main result of [Sc78, Theorem 3.0]. Independently, Creignou and Hermann [CH96] have defined the concept "quasi-equivalent," which is the same as the concept of "faithful representation" defined in this paper.

THEOREM 3. *If S does not satisfy any of the four conditions of Theorem* 2, *then S faithfully represents all Boolean relations.*

Proof. Assuming that none of the four conditions are satisfied by S, the proof proceeds by finding more and more elaborate Boolean relations that are faithfully represented by S. Notice that, since the notion of faithful representation was defined as equivalence of two S-formulas, we shall restrict the proof to the construction of appropriate S-clauses—faithful representation of the corresponding relations will then follow immediately. In this process the allowed operations must preserve the uniqueness of the values of the auxiliary variables and produce a formula which is also in conjuctive form. Therefore, if C and C' are S-formulas, the allowed operations are: (a) $C \wedge C'$, i.e., conjuction of two S-formulas, (b) $C[a/x]$, i.e., substitution of a variable symbol by another symbol, (c) $C[0/x]$ and $C[1/x]$, i.e., substitution of a variable by a constant (this is actually a selection of the tuples that agree in the specified constant), and (d) $\exists!xC(x)$, i.e., existential quantification, where the bound variables are uniquely defined. Some of the steps are provided by Schaefer's proof, and some are new.

Step 1. *Expressing* $[x \equiv \overline{y}]$. This was shown in [Sc78, Lemma 3.2 and Corollary 3.2.1]. The following exposition is somewhat simpler and is based on the fact that a set $M \subseteq \{0,1\}^n$ is the model set of a Horn formula iff it is closed under bitwise $\wedge$; see the Appendix and [KPS93].

Let R be any non-Horn relation of S (say of arity k). The closure property mentioned above implies that there exist models t and t' in R such that $t \wedge t' \notin R$. Based on R we may define the clause $R' = R(a_1, a_2, \ldots, a_k)$: set $a_i = 0$ (resp., 1) to all positions i where both t_i and ${t_i}'$ are 0 (resp., 1). Set $a_i = x$ to all positions where $t_i = 1$ and ${t_i}' = 0$, and $a_i = y$ to all positions where $t_i = 0$ and ${t_i}' = 1$. It is easy to see that both x and y actually appear in R'. (If not, then one of t and t' coincides with their conjunction.) Now 01 and 10 are models of R', but 00 is not. Hence R' is either $(x \equiv \overline{y})$ or $(x \vee y)$. If, in addition, S contains a relation which is not anti-Horn, then a symmetric argument rules out tuple 11, resulting in a clause R'' which is either $(x \equiv \overline{y})$ or $(\overline{x} \vee \overline{y})$. Hence $R' \wedge R''$ is $(x \equiv \overline{y})$. Notice that since this is the case we shall henceforth feel free to use negative literals in our expressions.

Step 2. *Expressing* $[x \vee y]$. Schaefer shows in Lemma 3.3 that there is an S-clause involving variables x, y, z whose set of models contains $000, 101, 011$, but not 110. The proof is as follows: it is known (see the Appendix) that an S-clause is affine if and only if for any three models t_0, t_1, t_2, their *exclusive-or* $t_0 \oplus t_1 \oplus t_2$ is also a model. Consider, therefore, an S-clause that is not affine and assume that $[x \equiv \overline{y}]$ can be represented. By the observation in Step 1 we may negate the variables of the clause in the positions where t_0 is 1. Now the new S-clause, call it S', is satisfied by the all-zero truth assignment and moreover by the assignments ${t_1}' = t_1 \oplus t_0$ and ${t_2}' = t_2 \oplus t_0$, but not by $0 \oplus {t_1}' \oplus {t_2}'$. Construct a new clause $R(a_1, a_2, \ldots, a_k)$ from S' (k is the arity of $[S']$) as follows. Set $a_i = 0$ in all positions i where both ${t_1}'$ and ${t_2}'$ are 0, $a_i = z$ where both are 1, $a_i = x$ where ${t_1}'$ is 0 and ${t_2}'$ is 1, and finally $a_i = y$ where ${t_1}'$ is 1 and ${t_2}'$ is 0. The S-clause R defined on x, y, z, has models 000, 011, 101 (corresponding to the all-zero assignment, ${t_1}'$ and ${t_2}'$ of S', respectively), but not 110 (which corresponds to ${t_1}' \oplus {t_2}'$).

We will show that R faithfully represents one of the four versions of *or:* $(x \vee y)$, $(x \vee \overline{y})$, $(\overline{x} \vee y)$, and $(\overline{x} \vee \overline{y})$. Observe that at least two of x, y, z actually occur in R. If exactly two variables are present in R, then R represents a version of *or* as follows: if x and y are present, then $R(x, y) = (\overline{x} \vee \overline{y})$; if x and z are present, then $R(x, z) = (\overline{x} \vee z)$; if y and z are present, then $R(y, z) = (\overline{y} \vee z)$. If all three variables are present, depending on which of the remaining four possible models are also in the model set of the S-clause, we have sixteen possible relations. Of these, the strongest, with models identical to the set $M = \{000, 101, 011\}$, can be used to define $X(x, y, z)$ (which is true when exactly one of x, y, z is true) as follows: $X(x, y, z) = R(x, y, \overline{z})$, and in this case the current step is unnecessary. In each of the other fifteen cases, we show by exhaustive analysis that there is an R-clause, with one constant, which represents a version of *or*. If $[R] = M \cup \{001\}$, then $R(x, y, 1) = (\overline{x} \vee \overline{y})$. If $[R] = M \cup \{010\}$, then $R(0, y, z) = (y \vee \overline{z})$. If $[R] = M \cup \{100\}$, then $R(x, 0, z) = (x \vee \overline{z})$. If $[R] = M \cup \{111\}$, then $R(x, y, 1) = (x \vee y)$. If $[R] = M \cup \{001, 010\}$, then $R(x, y, 1) = (\overline{x} \vee \overline{y})$. If $[R] = M \cup \{001, 100\}$, then $R(x, y, 1) = (\overline{x} \vee \overline{y})$. If $[R] = M \cup \{001, 111\}$, then $R(0, y, z) = (\overline{y} \vee z)$. If $[R] = M \cup \{010, 100\}$, then $R(0, y, z) = (y \vee \overline{z})$. If $[R] = M \cup \{010, 111\}$, then $R(0, y, z) = (y \vee \overline{z})$. If $[R] = M \cup \{100, 111\}$, then $R(x, y, 1) = (x \vee y)$. If $[R] = M \cup \{001, 010, 100\}$, then $R(x, y, 1) = (\overline{x} \vee \overline{y})$. If $[R] = M \cup \{001, 010, 111\}$, then $R(x, 1, z) = (\overline{x} \vee z)$. If $[R] = M \cup \{001, 111, 100\}$, then $R(x, 0, z) = (x \vee z)$. If $[R] = M \cup \{111, 010, 100\}$, then $R(x, y, 1) = (x \vee y)$. If $[R] = M \cup \{001, 010, 100, 111\}$, then $R(x, 1, z) = (\overline{x} \vee z)$. Since we can also faithfully express $[w \equiv \overline{x}]$, by Step 1, we have all four versions of *or*.

Step 3. *Expressing* $X(x, y, z)$. X is a formula which is satisfied if exactly one of the three variables has the value 1. It is known (see the Appendix) that an S-clause is 2CNF iff for any set of three satisfying assignments t_0, t_1, t_2, the assignment $(t_0 \vee t_1) \wedge (t_1 \vee t_2) \wedge (t_2 \vee t_0)$ is also a satisfying assignment.

We use this characterization to prove that if a relation set S contains a relation which is not 2CNF and also contains relations which are not Horn, anti-Horn, and affine, then $X(x, y, z)$ can be faithfully represented.

Consider an S-clause which is not 2CNF. We may therefore find three satisfying assignments t_0, t_1, t_2 such that the expression $(t_0 \vee t_1) \wedge (t_1 \vee t_2) \wedge (t_2 \vee t_0)$ is not a satisfying assignment. As in the previous step we may negate the variables in the positions where t_0 has the value 1, resulting in a new clause S', which is satisfied by the all-zero assignment, by $t_1' = t_1 \oplus t_0$ and by $t_2' = t_2 \oplus t_0$, but not by $t_1' \wedge (t_1' \vee t_2') \wedge t_2'$, which is equal to $t_1' \wedge t_2'$. Set 0 to all positions where both t_1' and t_2' are 0, x to all positions where both t_1' and t_2' are 1, y where t_1' is 0 and t_2' is 1, and finally z where t_1' is 1 and t_2' is 0. Observe that all three variables actually occur in the constructed clause R: if x is not present then $t_1' \wedge t_2'$ is identical to the all-zero assignment, a contradiction; if either y or z is not present then $t_1' \wedge t_2'$ is identical to t_1' or t_2', again a contradiction. The clause $R = (x, y, z)$ so constructed includes models 000, 110, and 101, but not 100. Now the S-clause $R(\overline{x}, y, z) \wedge (\overline{x} \vee \overline{y}) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{z} \vee \overline{x})$ has exactly the models 100, 010, and 001; i.e., it is $X(x, y, z)$.

Step 4. *Expressing* $[x \equiv (y \vee z)]$. Notice that the expression $X(\overline{x}, s, y) \wedge X(\overline{x}, t, z) \wedge X(s, t, u)$ is equivalent to

$$\big(x \equiv (y \vee z)\big) \wedge \big(s \equiv (\overline{y} \wedge z)\big) \wedge \big(t \equiv (y \wedge \overline{z})\big) \wedge \big(u \equiv (y \equiv z)\big).$$

Thus we prove that we can faithfully represent a relation in which a variable is logically equivalent to the *or* of two other variables. Notice that the auxiliary variables s, t, u are uniquely defined by the values of y and z.

Step 5. Using repeatedly $[x \equiv (y \vee z)]$ and $[x \equiv \overline{y}]$ we can faithfully represent any clause, and by taking conjunctions of arbitrary clauses we can faithfully represent any Boolean relation, completing the proof of Theorem 3. □

Proof of Theorem 2. Let S be a set of relations satisfying one of conditions (a)–(d), and let r be the maximum arity of any relation in S; we can solve the inverse satisfiability problem for S as follows.

Given a set of models M, we first identify in time $O(n^r|M|)$ all S-clauses that are satisfied by all models in M; call the conjunction of these S-clauses ϕ. Clearly, if there is a conjunction of S-clauses that has M as its set of models, then by the arguments used in Lemma 1, it is precisely ϕ. To tell whether the set of models of ϕ is indeed M, we show how to *generate* the set of models of ϕ with *polynomial delay* between consecutive outputs [JPY88]. Provided that such generation is possible, we can decide whether $M = \mu(\phi)$ by checking if the generated models belong in M. If a model not in M is generated, then we reply "no"; otherwise, if the set of models generated is exactly M, we reply "yes." Observe that the answer will be obtained after at most $|M| + 1$ generations, i.e., in overall polynomial time.

Our generation algorithm is based on a more general observation that also explains the analogy of our dichotomy theorem to the one of Schaefer's. Call a syntactic form of a Boolean formula *hereditary* if the substitution of a variable by a constant results in a new formula of the same syntactic form. Observe that the four cases for which we claim that the inverse satisfiability problem is polynomial are indeed hereditary and coincide with the polynomial cases of satisfiability [Sc78].

THEOREM 4. *If the following two conditions hold for a class of Boolean formulas:*

(a) *the syntactic form of the class is hereditary, and*

(b) *the satisfiability problem for the class is in PTIME,*

then the models of any formula in the class can be generated with polynomial delay between consecutive outputs.

Proof. Here is an informal description of the generation algorithm: at each step we substitute a variable by a constant, first by the value 1 and then by 0. Since (a) holds, the substitution results in a new formula of the same syntactic form. We then ask a polynomial-time oracle whether the produced formula is satisfiable. Since (b) holds, such an oracle exists. If the produced formula is satisfiable, we proceed recursively and substitute the next variable until all variables have been assigned a value, in which case we return the model. When at a certain step we are through with the value 1 for a variable (either by discovering a model or by rejecting the value because the produced formula is unsatisfiable), we try the value 0, and when finished, we backtrack to the previous step. It is easy to see that after at most $2n$ queries to the oracle (where n is the number of variables) we either generate a new model or we know that all models of the formula have been generated. □

Now, to show coNP-completeness of all other cases, let S be a set of Boolean relations not satisfying conditions (a)–(d). It is clear that the INVERSE SAT problem for S is in coNP: let $r \geq 3$ be the largest arity of any relation in S. Given a set of models M, we construct all S-clauses satisfied by all models in M—this takes $O(|M|n^r)$ time. M is the set of models of a conjunction of S-clauses if and only if all models not in M fail to satisfy at least one of these S-clauses.

To show completeness, we shall reduce UNSATISFIABILITY, the problem of telling whether a 3CNF expression ψ is unsatisfiable, to the INVERSE SAT(S). We suppose that ψ is a 3CNF expression on $n > 3r$ variables. Set M contains a model for each $3r$-tuple of variables and values for these variables that don't contradict any

clause of ψ. Let k be the cardinality of M, a quantity bounded by a function of r and of the number of variables and clauses of ψ. Notice that since r is constant, the number of models is not exponential. Our construction is a generalization of that of Theorem 1: we consider some total order among the pairs (W, T), where W is a set of $3r$ variables and T a truth assignment to those variables that does not contradict any clause of ψ. Every Boolean vector $m_{W,T}$ in M is a concatenation of two strings: $m_{W,T} = \beta_W^T \epsilon_W^T$.

String β_W^T is a concatenation of the encodings $\tau_W^T(x_i)$ for each variable x_i occuring in the formula ψ: $\beta_W^T = \tau_W^T(x_1) \cdots \tau_W^T(x_n)$. The encoding of $\tau_W^T(x)$ of a variable x is a Boolean vector of length $k+2$ and is defined in the proof of Theorem 1. Notice that in this construction the unique padding pattern for (W, T) occurs $n - 3r$ times in the string β_W^T. Call $N = n(k+2)$ the length of a string β_W^T.

The string ϵ_W^T is constructed as follows: we consider all 3CNF clauses on N variables satisfied by the set of strings β_W^T for all sets of $3r$ variables W and assignments T to those variables. Call ϕ the conjuction of all these clauses. We express ϕ faithfully by S-clauses. This will involve auxiliary variables $x_{N+1}, \ldots, x_{N+s}$. From the definition of faithful representation we see that $x_{N+\ell} \equiv f_\ell(x_{i_\ell}, x_{j_\ell})$, where $\ell = 1, \ldots, s$ and $i_\ell, j_\ell < N+\ell$. Notice, however, that each of the auxiliary variables depends on at most three of the N variables appearing in the 3CNF clauses. This follows from the fact that we are representing 3CNF clauses, and consequently, we can express each 3CNF clause separately by S-clauses and then take the conjunction of the representations. Thus, the overall dependency of an auxiliary variable $x_{N+\ell}, \ell = 1, \ldots, s$, is through a Boolean function $f_\ell(x_{i_\ell}, x_{j_\ell}, x_{k_\ell}), 1 \leq i_\ell, j_\ell, k_\ell \leq N$. Let $b_1 \cdots b_N$ be a string β_W^T. The values in the s positions of the corresponding string $\epsilon_W^T = b_{N+1} \cdots b_{N+s}$ are the values of the auxiliary variables: $b_{N+\ell} = f_\ell(b_{i_\ell}, b_{j_\ell}, b_{k_\ell})$, $\ell = 1, \ldots, s$. (Note that these values are stated explicitly, i.e., not encoded as value patterns.) This is where the concept of faithful representation is necessary: for each string β_W^T there is a unique string ϵ_W^T. With ordinary representation the multiple ways to extend a string β_W^T via the auxiliary variables would result in an exponential increase of our model set.

Let $M \subseteq \{0,1\}^{N+s}$ be the constructed set of models. We claim that M is the set of models of a conjunction of S-clauses iff the original 3CNF expression ψ is unsatisfiable.

If ψ is satisfiable, then M is not the set of models of any rCNF expression. Consider the model corresponding to the satisfying truth assignment. This model is a concatenation of two parts: the first has N positions and consists of the value patterns encoding the values of all variables in the satisfying truth assignment, exactly as in the proof of Theorem 1, and the second consists of the corresponding values of the s auxiliary variables. This model is r-compatible with M: any r-tuple restricted to the first N positions certainly matches a corresponding tuple in some model, by the construction of M. In fact, when the tuple is restricted to the first part, any $3r$-tuple can be matched. This is precisely why an r-tuple that is not restricted to the first N positions is also r-compatible: by the dependency of each auxiliary value to at most 3 of the first N, a compatibility of an i-tuple ($i \leq r$) in the second part holds if a $3i$-compatibility in the first part holds. Alternatively, instead of looking at a position in the second part, we can look at the three corresponding positions of the first part. Therefore, the whole model corresponding to the satisfying truth assignment is r-compatible with M. It follows by Lemma 1 that M is not rCNF, and as a result, M is not the set of models of any conjunction of S-clauses (recall that the maximum arity in S is r).

Suppose then that ψ is unsatisfiable. Let M' be M restricted to the first N positions. Then M' is exactly the set of models of ϕ (the conjuction of all 3CNF clauses on N variables which don't disagree with M') by the reasoning in Theorem 1: no model is 3-compatible with M' except those in M'. Since M' is the set of models of ϕ, it follows that M is the set of models of the corresponding conjunction of S-expressions that faithfully represents ϕ. □

Appendix. This appendix contains the proof of the closure properties of Horn, anti-Horn 2CNF, and affine sets of models, which are used in the proof of Theorem 3. In what follows, $M \subseteq \{0,1\}^n$ denotes a set of models.

HORN SETS. *M is Horn iff for any two models $t, t' \in M$ the model $(t \wedge t')$ is also in M.*

The proof is based on the following proposition from [KPS93]. If t and t' are bit-vectors we use the notation $t \leq t'$ to denote that $t_i = 1$ implies $t_i{}' = 1$.

PROPOSITION. *The following are equivalent.*

(a) *There is a Horn formula whose model set is M.*
(b) *For each $t \notin M$ either there is no $t' \in M$ with $t \leq t'$, or there is a unique minimal $t' \in M$ such that $t \leq t'$.*
(c) *If $t, t' \in M$, then also $t \wedge t' \in M$.*

Proof. That (a) implies (c) is easy. To establish (b) from (c), take t' to be the $\wedge$ of all $t'' \in M$ such that $t \leq t''$. Finally, if we have property (b), we can construct the following set of Horn clauses: for each $t \notin M$ let t' be the model guaranteed by (b); create a Horn clause $((\bigwedge_{t_i=1} x_i) \rightarrow x_j)$ for each j such that $t_j = 0$ and $t'_j = 1$. It is easy to see that the set of all these Horn clauses comprises the desired ϕ.

ANTI-HORN SETS. This case is symmetric to the above. Just replace 1 with 0 and $\wedge$ with $\vee$.

2CNF SETS. *M is 2CNF iff for any set of three models $t_0, t_1, t_2 \in M$ the model $(t_0 \vee t_1) \wedge (t_1 \vee t_2) \wedge (t_2 \vee t_0)$ is also in M.*

Proof. This was shown in [Sc78, Lemma 3.1B]. We give a different proof, which is simpler and is based on Lemma 1 for $k = 2$. First notice that the model $t = (t_0 \vee t_1) \wedge (t_1 \vee t_2) \wedge (t_2 \vee t_0)$ has the following property. The value of t in each position $i = 1, \ldots, n$ is equal to a value, which is the *majority* among the three values of the models t_0, t_1, t_2 in this position (e.g., if the values of models t_0, t_1, t_2 in position i are $(1, 1, 0)$, respectively, the value of t in position i is 1). Call the outcome of the operation $(t_0 \vee t_1) \wedge (t_1 \vee t_2) \wedge (t_2 \vee t_0)$ the *majority* model of t_0, t_1, t_2.

Only if: Suppose M is 2CNF. By Lemma 1 any 2-compatible model with M is in M. It is easy to see that the majority model of any three models is 2-compatible with these three models.

If: Suppose that the majority model of any set of three models $t_0, t_1, t_2 \in M$ is also in M. We shall prove that any 2-compatible model with M is in M. We prove this inductively, by showing that any 2-compatible model is in fact n-compatible. Consider a model m k-compatible with M and a $(k+1)$-tuple of positions in this model. The k distinct k-tuples of this $(k+1)$-tuple agree with some model in M. Take three of those not necessarily distinct k models. (If the models are less than three, then $m \in M$.) Notice that any one of those differs in at most one position of the $(k+1)$-tuple with m. Therefore, the $(k+1)$-tuple of m agrees with the majority model of those three models. Hence, m is $(k+1)$-compatible with M. Therefore, any 2-compatible model with M is in M and, by Lemma 1, M is a 2CNF set.

AFFINE SETS. *M is affine iff for any three models t_0, t_1, t_2 the model $t_0 \oplus t_1 \oplus t_2$ is also in M.*

Proof. This fact follows from linear algebra and especially the theory of diophantine linear equations. It states the intuitive observation (and its converse) that every convex polytope is the convex hull of its vertices. For more on that see the book of Schrijver [Sc86].

Acknowledgments. We are grateful to Christos Papadimitriou for helpful discussions and suggestions. We are also indebted to the anonymous referees for their detailed comments and suggestions that decisively helped us improve the presentation by making it more complete and precise.

REFERENCES

[AFP92] D. Angluin, M. Frazier, and L. Pitt, *Learning conjunctions of Horn clauses*, Machine Learning, 9 (1992), pp. 147–164.

[Ca93] M. Cadoli, *Semantical and computational considerations in Horn approximations*, in Proc. 13th International Joint Conference of Artificial Intelligence (IJCAI), Chambery France, Springer-Verlag, Berlin, 1993, pp. 39–44.

[CH96] N Creignou and M. Hermann, *Complexity of generalized satisfiability counting problems*, Inform. and Comput., 125 (1996), pp. 1–12.

[CK90] C. C. Chang and H. J. Keisler, *Model Theory*, Studies in Logic and the Foundation of Mathematics 73, 3rd ed., North-Holland, Amsterdam, 1990.

[Co71] S. A. Cook, *The complexity of theorem-proving procedures*, in Proc. 3rd Annual ACM Symposium on Theory of Computing, Shaker Heights, OH, 1971, pp. 151–158.

[DP92] R. Dechter and J. Pearl, *Structure identification in relational data*, Artificial Intelligence, 58 (1992), pp. 237–270.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, CA, 1979.

[GN87] M. R. Genesereth and N. J. Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, San Francisco, CA, 1987.

[GPS94] G. Gogic, C. H. Papadimitriou, and M. Sideri, *Incremental recompilation of knowledge*, in Proc. 12th National Conference on Artificial Intelligence, Seattle, WA, AAAI Press, Menlo Park, CA, 1994, pp. 922–927.

[JPY88] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, *On generating all maximal independent sets*, Inform. Process. Lett., 27 (1988), pp. 119–123.

[KKS95] H. A. Kautz, M. J. Kearns, and B. Selman, *Horn approximations of empirical data*, Artificial Intelligence, 74 (1995), pp. 129–145.

[KKS93] H. A. Kautz, M. J. Kearns, and B. Selman, *Reasoning with characteristic models*, in Proc. 11th National Conference on Artificial Intelligence, Washington, DC, AAAI Press, Menlo Park, CA, 1993, pp. 34–39.

[KPS93] D. Kavvadias, C. H. Papadimitriou, and M. Sideri, *On Horn envelopes and hypergraph transversals*, in Proc. 4th Annual International Symposium on Algorithms and Complexity, Hong Kong, Springer-Verlag, Berlin, 1993, pp. 399–405.

[Le86] H. Levesque, *Making believers out of computers*, Artificial Intelligence, 30 (1986), pp. 81–108.

[Mc80] J. McCarthy, *Circumscription–a form of nonmonotonic reasoning*, Artificial Intelligence, 13 (1980), pp. 27–39.

[Mo84] R. Moore, *Possible-world semantics for autoepistemic logic*, in Proc. 1st Nonmonotonic Reasoning Workshop, New Paltz, NY, 1984, pp. 344–354.

[Pa94] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.

[Re80] R. Reiter, *A Logic for default reasoning*, Artificial Intelligence, 13 (1980), pp. 81–132.

[Sc78] T. J. Schaefer, *The complexity of satisfiability problems*, in Proc. 10th Annual ACM Symposium on Theory of Computing, San Diego, CA, 1978, pp. 216–226.

[Sc86] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley, New York, 1986.

[SK90] B. Selman and H. A. Kautz, *Model preference default theories*, Artificial Intelligence, 45 (1990), pp. 287–322.

[SK91] B. Selman and H. A. Kautz, *Knowledge compilation using Horn approximation*, in Proc. 9th National Conference on Artificial Intelligence, Anaheim, CA, MIT Press, Cambridge, MA, 1991, pp. 904–909.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 164–191

ON SYNTACTIC VERSUS COMPUTATIONAL VIEWS OF APPROXIMABILITY*

SANJEEV KHANNA†, RAJEEV MOTWANI‡, MADHU SUDAN§, AND UMESH VAZIRANI¶

Abstract. We attempt to reconcile the two distinct views of approximation classes: *syntactic* and *computational*. Syntactic classes such as MAX SNP permit structural results and have natural complete problems, while computational classes such as APX allow us to work with classes of problems whose approximability is well understood. Our results provide a syntactic characterization of computational classes and give a computational framework for syntactic classes.

We compare the syntactically defined class MAX SNP with the computationally defined class APX and show that every problem in APX can be "placed" (i.e., has approximation-preserving reduction to a problem) in MAX SNP. Our methods introduce a simple, yet general, technique for creating approximation-preserving reductions which shows that any "well"-approximable problem can be reduced in an approximation-preserving manner to a problem which is hard to approximate to corresponding factors. The reduction then follows easily from the recent nonapproximability results for MAX SNP-hard problems. We demonstrate the generality of this technique by applying it to other classes such as MAX SNP-RMAX(2) and MIN $F^+\Pi_2(1)$ which have the clique problem and the set cover problem, respectively, as complete problems.

The syntactic nature of MAX SNP was used by Papadimitriou and Yannakakis [*J. Comput. System Sci.*, 43 (1991), pp. 425–440] to provide approximation algorithms for every problem in the class. We provide an alternate approach to demonstrating this result using the syntactic nature of MAX SNP. We develop a general paradigm, *nonoblivious local search*, useful for developing simple yet efficient approximation algorithms. We show that such algorithms can find good approximations for all MAX SNP problems, yielding approximation ratios comparable to the best known for a variety of specific MAX SNP-hard problems. Nonoblivious local search provably outperforms standard local search in both the degree of approximation achieved and the efficiency of the resulting algorithms.

Key words. approximation algorithms, complete problems, computational complexity, computational classes, polynomial reductions, local search

AMS subject classification. 68Q15

PII. S0097539795286612

1. Introduction.

The approximability of NP optimization (NPO) problems has been investigated in the past via the definition of two different types of problem classes: syntactically defined classes such as MAX SNP (the class of NPO problems expressible as bounded-arity constraint satisfaction problems) and computationally defined classes such as APX (the class of NPO problems to which a constant-factor approximation can be found in polynomial time); see section 2 for formal definitions. The former

*Received by the editors May 26, 1996; accepted for publication (in revised form) November 22, 1996; published electronically June 15, 1998. A preliminary version of this paper appeared in *The Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, 1994.
http://www.siam.org/journals/sicomp/28-1/28661.html.

†Department of Computer Science, Stanford University, Stanford, CA 94305. Current address: Fundamental Math Research Department, Bell Labs, 700 Mountain Ave., Murray Hill, NJ 07974 (sanjeev@research.bell-labs.com). This research was supported by a Schlumberger Foundation Fellowship, an OTL grant, and NSF grant CCR-9357849.

‡Department of Computer Science, Stanford University, Stanford, CA 94305 (rajeev@theory.stanford.edu). This research was supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Development Award, an OTL grant, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, the Schlumberger Foundation, the Shell Foundation, and Xerox Corporation.

§IBM T.J. Watson Research Center, Yorktown Heights, NY 10598 (madhu@watson.ibm.com).

¶Computer Science Division, University of California at Berkeley, CA 94720 (vazirani@cs.berkeley.edu). This research was supported by NSF grant CCR-9310214.

is useful for obtaining structural results and has natural complete problems, while the latter allows us to work with classes of problems whose approximability is completely determined. We attempt to develop linkages between these two views of approximation problems and thereby obtain new insights into both types of classes. We show that a natural generalization of MAX SNP renders it identical to the class APX. This further validates Papadimitriou and Yannakakis's definition [23] of MAX SNP as providing a structural basis for the study of approximability. As a side-effect, we resolve the open problem of identifying complete problems for MAX NP. Our techniques extend to a generic theorem that can be used to create an approximation hierarchy. We also develop a generic algorithmic paradigm which is guaranteed to provide good approximations for MAX SNP problems and may also have other applications.

1.1. Background and motivation. A wide variety of classes is defined based directly on the polynomial-time approximability of the problems contained within, e.g., APX (constant-factor approximable problems), PTAS (problems with polynomial-time approximation schemes), and FPTAS (problems with fully-polynomial-time approximation schemes). The advantage of working with classes defined using approximability as the criterion is that it allows us to work with problems whose approximability is well understood. Crescenzi and Panconesi [8] have recently also been able to exhibit complete problems for such classes, particularly APX. Unfortunately such complete problems seem to be rare and artificial, and do not seem to provide insight into the more natural problems in the class. Research in this direction has to find approximation-preserving reductions from the known complete but artificial problems in such classes to the natural problems therein, with a view to understanding the approximability of the latter.

The second family of classes of NPO problems that have been studied are those defined via syntactic considerations, based on a syntactic characterization of NP due to Fagin [10]. Research in this direction, initiated by Papadimitriou and Yannakakis [23] and followed by Panconesi and Ranjan [22] and Kolaitis and Thakur [20], has led to the identification of approximation classes such as MAX SNP, RMAX(2), and MIN $F^+\Pi_2(1)$. The syntactic prescription in the definition of these classes has proved very useful in the establishment of complete problems. Moreover, the recent results of Arora et al. [2] have established the hardness of approximating complete problems for MAX SNP to within (specific) constant factors unless P = NP. It is natural to wonder why the hardest problems in this syntactic subclass of APX should bear any relation to all of NP.

Though the computational view allows us to precisely classify the problems based on their approximability, it does not yield structural insights into natural questions such as: Why are certain problems easier to approximate than some others? What is the canonical structure of the hardest representative problems of a given approximation class? and so on. Furthermore, intuitively speaking, this view is too abstract to facilitate identification of, and reductions to establish, natural complete problems for a class. The syntactic view, on the other hand, is essentially a structural view. The syntactic prescription gives a natural way of identifying canonical hard problems in the class and performing approximation-preserving reductions to establish complete problems.

Attempts at trying to find a class with both the above-mentioned properties, i.e., natural complete problems and capturing all problems of a specified approximability, have not been very successful. Typically the focus has been to relax the syntactic criteria to allow a wider class of problems to be included in the class. However, in

all such cases it seems inevitable that these classes cannot be expressive enough to encompass all problems with a given approximability. This is because each of these syntactically defined approximation classes is strictly contained in the class NPO; the strict containment can be shown by syntactic considerations alone. As a result, if we could show that any of these classes contains all of P, then we would have separated P from NP. We would expect that every class of this nature would be missing some problems from P, and this has indeed been the case with all current definitions.

We explore a different direction by studying the structure of the syntactically defined classes when we look at their closure under approximation-preserving reductions. The idea of looking at the closure of a class is implicit in the work of Papadimitriou and Yannakakis [23] who state that *minimization problems will be "placed" in the classes through L-reductions to maximization problems.* The advantage of looking at the closure of a set is that it maintains the complete problems of the set, while managing to include all of P in the closure (for problems in P, the reduction is to simply use a polynomial-time algorithm to compute an exact solution). It now becomes interesting, for example, to compare the closure of MAX SNP (denoted $\overline{\text{MAX SNP}}$) with APX. A positive resolution, i.e., $\overline{\text{MAX SNP}} = \text{APX}$, would immediately imply the nonexistence of a PTAS for MAX SNP-hard problems, since it is known that PTAS is a strict subset of APX, if $\text{P} \neq \text{NP}$. On the other hand, an unconditional negative result would be difficult to obtain, since it would imply $\text{P} \neq \text{NP}$.

Here we resolve this question in the affirmative. The exact nature of the result obtained depends upon the precise notion of an approximation-preserving reduction used to define the closure of the class MAX SNP. The strictest notion of such reductions available in the literature are the *L*-reductions due to Papadimitriou and Yannakakis [23]. We work with a slight extension of the reduction, which we call *E*-reductions. Using such reductions to define the class $\overline{\text{MAX SNP}}$ we show that this equals APX-PB, the class of all polynomially bounded NPO problems which are approximable to within constant factors. By using slightly looser definitions of approximation-preserving reductions (and in particular the PTAS-reductions of Crescenzi and Trevisan [9]) this can be extended to include all of APX in $\overline{\text{MAX SNP}}$. We then build upon this result to identify an interesting hierarchy of such approximability classes. An interesting side-effect of our results is the positive answer to the question of Papadimitriou and Yannakakis [23] about whether MAX NP has any complete problems.

The syntactic view seems useful not only in obtaining structural complexity results but also in developing paradigms for designing efficient approximation algorithms. This was demonstrated first by Papadimitriou and Yannakakis [23] who show approximation algorithms for every problem in MAX SNP. We further exploit the syntactic nature of MAX SNP to develop another paradigm for designing good approximation algorithms for problems in that class and thereby provide an alternate computational view of it. We refer to this paradigm as *nonoblivious local search*, and it is a modification of the standard local search technique [25]. We show that every MAX SNP problem can be approximated to within constant factors by such algorithms. It turns out that the performance of nonoblivious local search is comparable to that of the best-known approximation algorithms for several interesting and representative problems in MAX SNP. An intriguing possibility is that this is not a coincidence, but rather a hint at the universality of the paradigm or some variant thereof.

Our results are related to some extent to those of Ausiello and Protasi [4]. They

define a class GLO (for guaranteed local optima) of NPO problems which have the property that for all locally optimum solutions, the ratio between the values of the global and the local optima is bounded by a constant. It follows that GLO is a subset of APX, and it was shown that it is in fact a strict subset. We show that a MAX SNP problem is not contained in GLO, thereby establishing that MAX SNP is not contained in GLO. This contrasts with our notion of nonoblivious local search which is guaranteed to provide constant-factor approximations for all problems in MAX SNP. In fact, our results indicate that nonoblivious local search is significantly more powerful than standard local search in that it delivers strictly better constant ratios, and will also provide constant-factor approximations to problems not in GLO. Independently of our work, Alimonti [1] has used a similar local search technique for the approximation of a specific problem not contained in GLO or MAX SNP.

1.2. Summary of results. In section 2, we present the definitions required to state our results, and in particular the definitions of an E-reduction, APX, APX-PB, MAX SNP, and $\overline{\text{MAX SNP}}$. In section 3, we show that $\overline{\text{MAX SNP}} = \text{APX-PB}$. A generic theorem which allows us to equate the closure of syntactic classes to appropriate computational classes is outlined in section 4; we also develop an approximation hierarchy based on this result.

The notion of nonoblivious local search and NONOBLIVIOUS GLO is developed in section 5. In section 6, we illustrate the power of nonobliviousness by first showing that oblivious local search can achieve at most the performance ratio 3/2 for MAX 2-SAT, even if it is allowed to search *exponentially* large neighborhoods; in contrast, a very simple nonoblivious local search algorithm achieves a performance ratio of 4/3. We then establish that this paradigm yields a $2^k/(2^k - 1)$ approximation to MAX k-SAT. In section 7, we provide an alternate characterization of MAX SNP via a class of problems called MAX k-CSP. It is shown that a simple nonoblivious algorithm achieves the best-known approximation for this problem, thereby providing a *uniform* approximation for all of MAX SNP. In section 8, we further illustrate the power of this class of algorithm by showing that it can achieve the best-known ratio for a specific MAX SNP problem and for VERTEX COVER (which is not contained in GLO). This implies that MAX SNP is not contained in GLO, and that GLO is a strict subset of NONOBLIVIOUS GLO. In section 9, we apply it to approximating the traveling salesman problem. Finally, in section 10, we apply this technique to improving a long-standing approximation bound for maximum independent sets in bounded-degree graphs.

2. Preliminaries and definitions. Given an NPO problem Π and an instance $\mathcal{I}$ of Π, we use $|\mathcal{I}|$ to denote the length of $\mathcal{I}$ and $OPT(\mathcal{I})$ to denote the optimum value for this instance. For any solution S to $\mathcal{I}$, the value of the solution, denoted by $V(\mathcal{I}, S)$, is assumed to be a polynomial-time computable function which takes positive integer values (see [7] for a precise definition of NPO).

DEFINITION 1 (error). *Given a solution S to an instance $\mathcal{I}$ of an* NPO *problem* Π*, we define its* error $\mathcal{E}(\mathcal{I}, S)$ *as*

$$\mathcal{E}(\mathcal{I}, S) = \max\left\{\frac{V(\mathcal{I}, S)}{OPT(\mathcal{I})}, \frac{OPT(\mathcal{I})}{V(\mathcal{I}, S)}\right\} - 1.$$

Notice that the above definition of error applies uniformly to the minimization and maximization problems at all levels of approximability.

DEFINITION 2 (performance ratio). *An approximation algorithm A for an optimization problem Π has* performance ratio $\mathcal{R}(n)$ *if, given an instance $\mathcal{I}$ of Π with $|\mathcal{I}| = n$, the solution $A(\mathcal{I})$ satisfies*

$$\max\left\{\frac{V(\mathcal{I}, A(\mathcal{I}))}{OPT(\mathcal{I})}, \frac{OPT(\mathcal{I})}{V(\mathcal{I}, A(\mathcal{I}))}\right\} \leq \mathcal{R}(n).$$

A solution of value within a multiplicative factor r of the optimal value is referred to as an r-approximation.

The performance ratio for A is $\mathcal{R}$ if it always computes a solution with error at most $\mathcal{R} - 1$.

2.1. *E*-reductions. We now describe the precise approximation-preserving reduction we will use in this paper. This reduction, which we call the E-reduction, is essentially the same as the L-reduction of Papadimitriou and Yannakakis [23] and differs from it in only one relatively minor aspect.

DEFINITION 3 (E-reduction). *A problem Π* E-reduces *to a problem Π' (denoted $\Pi \propto_E \Pi'$) if there exist polynomial-time computable functions f, g and a constant β such that*

- *f maps an instance $\mathcal{I}$ of Π to an instance $\mathcal{I}'$ of Π' such that $OPT(\mathcal{I})$ and $OPT(\mathcal{I}')$ are related by a polynomial factor; i.e., there exists a polynomial $p(n)$ such that $OPT(\mathcal{I}') \leq p(|\mathcal{I}|)OPT(\mathcal{I})$.*
- *g maps solutions S' of $\mathcal{I}'$ to solutions S of $\mathcal{I}$ such that*

$$\mathcal{E}(\mathcal{I}, S) \leq \beta\mathcal{E}(\mathcal{I}', S').$$

Remark 1. Among the many approximation-preserving reductions in the literature, the L-reduction appears to be the strictest. The E-reduction appears to be slightly weaker (in that it allows polynomial scaling of the problems), but is stricter than any of the other known reductions. Since all the reductions given in this paper are E-reductions, they would also qualify as approximation-preserving reductions under most other definitions, and in particular, they fit the definitions of F-reductions and P-reductions of Crescenzi and Panconesi [8].

Remark 2. Having $\Pi \propto_E \Pi'$ implies that Π is as well approximable as Π'; in fact, an E-reduction is an FPTAS-preserving reduction. An important benefit is that this reduction can be applied uniformly at all levels of approximability. This is not the case with the other existing definitions of FPTAS-preserving reduction in the literature. For example, the FPTAS-preserving reduction (F-reduction) of Crescenzi and Panconesi [8] is much more unrestricted in scope and does not share this important property of the E-reduction. Note that Crescenzi and Panconesi [8] showed that there exists a problem $\Pi' \in$ PTAS such that for any problem $\Pi \in$ APX, $\Pi \propto_F \Pi'$. Thus, there is the undesirable situation that a problem Π with no PTAS has an FPTAS-preserving reduction to a problem Π' with a PTAS.

Remark 3. The L-reduction of Papadimitriou and Yannakakis [23] enforces the condition that the optima of an instance $\mathcal{I}$ of Π be linearly related to the optima of the instance $\mathcal{I}'$ of Π' to which it is mapped. This appears to be an unnatural restriction considering that the reduction itself is allowed to be an arbitrary polynomial-time computation. This is the only real difference between their L-reduction and our E-reduction, and an E-reduction in which the linearity relation of the optimas is satisfied is an L-reduction. Intuitively, however, in the study of approximability the desirable attribute is simply that the errors in the corresponding solutions are closely

(linearly) related. The somewhat artificial requirement of a linear relation between the optimum values precludes reductions between problems which are related to each other by some *scaling factor*. For instance, it seems desirable that two problems whose objective functions are simply related by any fixed polynomial factor should be interreducible under any reasonable definition of an approximation-preserving reduction. Our relaxation of the L-reduction constraint is motivated precisely by this consideration.

Let $\mathcal{C}$ be any class of NPO problems. Using the notion of an E-reduction, we define hardness and completeness of problems with respect to $\mathcal{C}$, as well as its closure and polynomially bounded subclass.

DEFINITION 4 (hard and complete problems). *A problem* Π' *is said to be* $\mathcal{C}$-hard *if for all problems* $\Pi \in \mathcal{C}$*, we have* $\Pi \propto_E \Pi'$*. A* $\mathcal{C}$*-hard problem* Π *is said to be* $\mathcal{C}$-complete *if, in addition,* $\Pi \in \mathcal{C}$*.*

DEFINITION 5 (closure). *The* closure *of* $\mathcal{C}$*, denoted by* $\overline{\mathcal{C}}$*, is the set of all* NPO *problems* Π *such that* $\Pi \propto_E \Pi'$ *for some* $\Pi' \in \mathcal{C}$*.*

Remark 4. The closure operation maintains the set of complete problems for a class.

DEFINITION 6 (polynomially bounded subset). *The* polynomially bounded subset *of* $\mathcal{C}$*, denoted* $\mathcal{C}$*-PB, is the set of all problems* $\Pi \in \mathcal{C}$ *for which there exists a polynomial* $p(n)$ *such that for all instances* $\mathcal{I} \in \Pi$*,* $OPT(\mathcal{I}) \leq p(|\mathcal{I}|)$*.*

2.2. Computational and syntactic classes. We first define the basic computational class APX.

DEFINITION 7 (APX). *An* NPO *problem* Π *is in the class* APX *if there exists a polynomial-time algorithm* A *for* Π *with performance ratio bounded by some constant* c*.*

The class APX-PB consists of all polynomially bounded NPO problems which can be approximated to within constant factors in polynomial time.

If we let F-APX denote the class of NPO problems that are approximable to within a factor F, then we obtain a *hierarchy* of approximation classes. For instance, poly-APX and log-APX are the classes of NPO problems which have polynomial-time algorithms with performance ratio bounded polynomially and logarithmically, respectively, in the input length. More precise versions of these definitions are provided in section 4.

Let us briefly review the definitions of some syntactic classes.

DEFINITION 8 (MAX SNP and MAX NP [23]). MAX SNP *is the class of* NPO *problems expressible as finding the structure* S *which maximizes the objective function*

$$V(\mathcal{I}, S) = |\{\vec{x} \mid \Phi(\mathcal{I}, S, \vec{x})\}|,$$

where $\mathcal{I} = (U; \mathcal{P})$ *denotes the input (consisting of a finite universe* U *and a finite set of bounded arity predicates* $\mathcal{P}$*),* S *is a finite structure, and* Φ *is a quantifier-free first-order formula. The class* MAX NP *is defined analogously, but the objective function is*

$$V(\mathcal{I}, S) = |\{\vec{x} \mid \exists \vec{y}, \Phi(\mathcal{I}, S, \vec{x}, \vec{y})\}|.$$

A natural extension is to associate a weight with every tuple $\vec{x}$; the modified objective is to find an S which maximizes $V(\mathcal{I}, S) = \sum_{\vec{x}} w(\vec{x})\Phi(\mathcal{I}, S, \vec{x})$, where $w(\vec{x})$ denotes the weight associated with the tuple $\vec{x}$.

Example 1 (MAX k-SAT). The MAX k-SAT problem is: given a collection of m clauses on n boolean variables where each (possibly weighted) clause is a disjunction of precisely k literals, find a truth assignment satisfying a maximum weight collection of clauses. For any fixed integer k, MAX k-SAT belongs to the class MAX SNP. The results of Papadimitriou and Yannakakis [23] can be adapted to show that for $k \geq 2$, MAX k-SAT is complete under E-reductions for the class MAX SNP.

DEFINITION 9 (RMAX(k) [22]). RMAX(k) *is the class of* NPO *problems expressible as finding a structure* S *which maximizes the objective function*

$$V(\mathcal{I}, S) = \begin{cases} |\{\vec{x} \mid S(\vec{x})\}| & \text{if } \forall \vec{y}, \Phi(\mathcal{I}, S, \vec{y}), \\ 0 & \text{otherwise}, \end{cases}$$

where S *is a single predicate and* $\Phi(\mathcal{I}, S, \vec{y})$ *is a quantifier-free CNF formula in which* S *occurs at most* k *times in each clause and all its occurrences are negative.*

The results of Panconesi and Ranjan [22] can be adapted to show that MAX CLIQUE is complete under E-reductions for the class RMAX(2).

DEFINITION 10 (MIN $\mathrm{F}^+\mathbf{\Pi_2(}k\mathbf{)}$ [20]). MIN $\mathrm{F}^+\Pi_2(k)$ *is the class of* NPO *problems expressible as finding a structure* S *which minimizes the objective function*

$$V(\mathcal{I}, S) = \begin{cases} |\{\vec{x} : S(\vec{x})\}| & \text{if } \forall \vec{x}, \exists \vec{y}, \Phi(\mathcal{I}, S, \vec{x}, \vec{y}), \\ 0 & \text{otherwise}, \end{cases}$$

where S *is a single predicate,* $\Phi(\mathcal{I}, S, \vec{y})$ *is a quantifier-free CNF formula in which* S *occurs at most* k *times in each clause, and all its occurrences are positive.*

The results of Kolaitis and Thakur [20] can be adapted to show that SET COVER is complete under E-reductions for the class MIN $\mathrm{F}^+\Pi_2(1)$.

3. MAX SNP closure and APX-PB. In this section, we will establish the following theorem and examine its implications. The proof is based on the results of Arora et al. [2] on efficient proof verifications.

THEOREM 1. $\overline{\text{MAX SNP}} = \text{APX-PB}$.

Remark 5. The seeming weakness that $\overline{\text{MAX SNP}}$ only captures polynomially bounded APX problems can be removed by using looser forms of approximation-preserving reduction in defining the closure. In particular, Crescenzi and Trevisan [9] define the notion of a PTAS-preserving reduction under which $\text{APX} = \overline{\text{APX-PB}}$. Using their result in conjunction with the above theorem, it is easily seen that $\overline{\text{MAX SNP}} = \text{APX}$. This weaker reduction is necessary to allow for reductions from fine-grained optimization problems to coarser (polynomially bounded) optimization problems (cf. [9]).

The following is a surprising consequence of Theorem 1.

THEOREM 2. $\overline{\text{MAX NP}} = \overline{\text{MAX SNP}}$.

Papadimitriou and Yannakakis [23] (implicitly) introduced both these closure classes but did not conjecture them to be the same. It would be interesting to see if this equality can be shown independent of the result of Arora et al. [2]. We also obtain the following resolution to the problem posed by Papadimitriou and Yannakakis [23] of finding complete problems for MAX NP.

THEOREM 3. MAX SAT *is complete for* MAX NP.

The following subsections establish that $\overline{\text{MAX SNP}} \supseteq \text{APX-PB}$. The idea is to first E-reduce any minimization problem in APX-PB to a maximization problem in APX-PB, and to then E-reduce any maximization problem in APX-PB to a specific complete problem for MAX SNP, viz., MAX 3-SAT. Since an E-reduction forces the

optima of the two problems involved to be related by polynomial factors, it is easy to see that $\overline{\text{MAX SNP}} \subseteq$ APX-PB. Combining these two facts, we obtain Theorem 1.

3.1. Reducing minimization to maximization. Observe that the fact that Π belongs to APX implies the existence of an approximation algorithm A and a constant c such that

$$\frac{OPT(\mathcal{I})}{c} \leq V(\mathcal{I}, A(\mathcal{I})) \leq c \times OPT(\mathcal{I}).$$

Henceforth, we will use $a(\mathcal{I})$ to denote $V(\mathcal{I}, A(\mathcal{I}))$. We first reduce any minimization problem $\Pi \in$ APX-PB to a maximization problem $\Pi' \in$ APX-PB, where the latter is obtained by merely modifying the objective function for Π, as follows. Let Π' have the objective function

$$V'(\mathcal{I}, S) = \max\{1, (c+1)a(\mathcal{I}) - cV(\mathcal{I}, S)\}$$

for all instances $\mathcal{I}$ and solutions S for Π. Clearly, $V'(\mathcal{I}, S)$ takes only positive values. To ensure that $V'(\mathcal{I}, S)$ is integer-valued, we can assume, without loss of generality, that c is an integer (a real-valued performance ratio can always be rounded up to the next integer). It can be verified that the optimum value for any instance $\mathcal{I}$ of Π' always lies between $a(\mathcal{I})$ and $(c+1)a(\mathcal{I})$. Thus A is a $(c+1)$-approximation algorithm for Π'.

Now, given a solution S', for instance, $\mathcal{I}$, of Π' such that it has error δ, we want to construct a solution S, for instance, $\mathcal{I}$, of Π such that the error is at most $\beta\delta$ for some β. We will show this for $\beta = (c+1)$.

First consider the case when $V'(\mathcal{I}, S') = 1$; i.e., $\delta = a(\mathcal{I}) - 1$. In this case, we simply output the solution $S = A(\mathcal{I})$. If $a(\mathcal{I}) = 1$ then we are trivially done; else we observe that

$$\mathcal{E}(I, S) \leq (c-1) \leq (c+1)(a(\mathcal{I}) - 1) \leq \beta\mathcal{E}'(I, S').$$

On the other hand, if $V'(\mathcal{I}, S') > 1$, we may proceed as follows. If S' is a δ-error solution to the optimum of Π', i.e.,

$$V'(\mathcal{I}, S) \geq \frac{OPT'(\mathcal{I})}{1+\delta} \geq (1-\delta)OPT'(\mathcal{I}),$$

where $OPT'(\mathcal{I})$ is the optimal value of V' for $\mathcal{I}$, we can conclude that

$$\begin{aligned} V(\mathcal{I}, S) &= \frac{(c+1)a(\mathcal{I}) - V'(\mathcal{I}, S)}{c} \\ &\leq \frac{(c+1)a(\mathcal{I}) - OPT'(\mathcal{I}) + \delta \times OPT'(\mathcal{I})}{c} \\ &\leq \frac{c \times OPT(\mathcal{I}) + \delta \times OPT'(\mathcal{I})}{c} \\ &\leq OPT(\mathcal{I}) + (c+1)\delta\, OPT(\mathcal{I}). \end{aligned}$$

Thus a solution s to Π' with error δ is a solution to Π with error at most $(c+1)\delta$, implying an E-reduction with $\beta = c+1$.

3.2. NP languages and MAX 3-SAT. The following theorem, adapted from a result of Arora et al. [2], is critical to our E-reduction of maximization problems to MAX 3-SAT.

THEOREM 4 (see [2]). *Given a language $L \in$ NP and an instance $x \in \Sigma^n$, one can compute in polynomial time an instance $\mathcal{F}_x$ of* MAX 3-SAT*, with the following properties.*

1. *The formula $\mathcal{F}_x$ has m clauses, where m depends only on n.*
2. *There exists a constant $\epsilon > 0$, independent of the input x, such that $(1-\epsilon)m$ clauses of $\mathcal{F}_x$ are satisfied by some truth assignment.*
3. *If $x \in L$, then $\mathcal{F}_x$ is (completely) satisfiable.*
4. *If $x \notin L$, then no truth assignment satisfies more than $(1-\epsilon)m$ clauses of $\mathcal{F}_x$.*
5. *Given a truth assignment which satisfies more than $(1-\epsilon)m$ clauses of $\mathcal{F}_x$, a truth assignment which satisfies $\mathcal{F}_x$ completely (or, alternatively, a witness showing $x \in L$) can be constructed in polynomial time.*

Some of the properties above may not be immediately obvious from the construction given by Arora et al. [2]. It is easy to verify that they provide a reduction with properties 1, 3, and 4. Property 5 is obtained from the fact that all assignments which satisfy most clauses are actually close (in terms of Hamming distance) to valid codewords from a linear code, and the uniquely error-corrected codeword obtained from this "corrupted codeword" will satisfy all the clauses of $\mathcal{F}_x$.

Property 2 requires a bit more care, and we provide a brief sketch of how it may be ensured. The idea is to revert back to the PCP model and redefine the proof verification game. Suppose that the original game had the properties that for $x \in L$ there exists a proof such that the verifier accepts with probability 1; and otherwise, for $x \notin L$, the verifier accepts with probability at most $1/2$. We now augment this game by adding to the proof a zeroth bit which the prover uses as follows: if the bit is set to 1, then the prover "chooses" to play the old game; else he is effectively "giving up" on the game. The verifier in turn first looks at the zeroth bit of the proof. If this is set, then she performs the usual verification; else she tosses an unbiased coin and accepts if and only if it turns up heads. It is clear that for $x \in L$ there exists a proof on which the verifier always accepts. Also, for $x \notin L$, no proof can cause the verifier to accept with probability greater than $1/2$. Finally, by setting the zeroth bit to 0, the prover can create a proof which the verifier accepts with probability exactly $1/2$. This proof system can now be transformed into a 3-CNF formula of the desired form.

3.3. Reducing maximization to MAX 3-SAT. We have already established that, without loss of generality, we only need to worry about maximization problems $\Pi \in$ APX-PB. Consider such a problem Π, and let A be a polynomial-time algorithm which delivers a c-approximation for Π, where c is some constant. Given any instance $\mathcal{I}$ of Π, let $p = ca(\mathcal{I})$ be the bound on the optimum value for $\mathcal{I}$ obtained by running A on input $\mathcal{I}$. Note that this may be a stronger bound than the a priori polynomial bound on the optimum value for any instance of length $|\mathcal{I}|$. An important consequence is that $p \leq cOPT(\mathcal{I})$.

We generate a sequence of NP decision problems $L_i = \{\mathcal{I} | \, OPT(\mathcal{I}) \geq i\}$ for $1 \leq i \leq p$. Given an instance $\mathcal{I}$, we create p formulas $\mathcal{F}_i$, for $1 \leq i \leq p$, using the reduction from Theorem 4, where the ith formula is obtained from the NP language L_i.

Consider now the formula $\mathcal{F} = \bigwedge_{i=1}^{p} \mathcal{F}_i$ that has the following features.

- The number of satisfiable clauses of $\mathcal{F}$ is exactly

$$\text{MAX} = (1-\epsilon)mp + \epsilon m\, OPT(\mathcal{I}),$$

  where ϵ and m are as guaranteed by Theorem 4.
- Given an assignment which satisfies $(1-\epsilon)mp + \epsilon mj$ clauses of $\mathcal{F}$, we can construct in polynomial time a solution to $\mathcal{I}$ of value at least j. To see this, observe the following: any assignment with so many clauses must satisfy more than $(1-\epsilon)m$ clauses in at least j of the formulas $\mathcal{F}_i$. Let i be the largest index for which this happens; clearly, $i \geq j$. Furthermore, by property (5) of Theorem 4, we can now construct a truth assignment which satisfies $\mathcal{F}_i$ completely. This truth assignment can be used to obtain a solution S such that $V(\mathcal{I}, S) \geq i \geq j$.

In order to complete the proof it remains to be shown that given any truth assignment with error δ, i.e., which satisfies $\text{MAX}/(1+\delta)$ clauses of $\mathcal{F}$, we can find a solution S for $\mathcal{I}$ with error $\mathcal{E}(\mathcal{I}, S) \leq \beta\delta$ for some constant β. We show that this is possible for $\beta = (c^2 + c\epsilon)/\epsilon$. The main idea behind finding such a solution is to use the second property above to find a "good" solution to $\mathcal{I}$ using a "good" truth assignment for $\mathcal{F}$.

Suppose we are given a solution which satisfies $\text{MAX}/(1+\delta)$ clauses. Since $\text{MAX}/(1+\delta) \geq (1-\delta)\,\text{MAX}$ and $\text{MAX} = (1-\epsilon)mp + \epsilon m\, OPT(\mathcal{I})$, we can use the second feature from above to construct a solution S_1 such that

$$\begin{aligned} V(\mathcal{I}, S_1) &\geq \frac{(1-\delta)\,\text{MAX} - (1-\epsilon)mp}{\epsilon m} \\ &\geq (1-\delta)\, OPT(\mathcal{I}) - \frac{\delta}{\epsilon} p \\ &\geq \left(1 - \delta\left(1 + \frac{c}{\epsilon}\right)\right) OPT(\mathcal{I}). \end{aligned}$$

Suppose $\delta \leq (c-1)\epsilon/(c(c+\epsilon))$. Let $\delta^* = \delta(1 + c/\epsilon)$ and $\gamma = \delta^*/(1-\delta^*)$. Then it is readily seen that

$$V(\mathcal{I}, S_1) \geq \frac{OPT(\mathcal{I})}{1+\gamma}$$

and that

$$0 \leq \gamma \leq \left(\frac{c^2 + c\epsilon}{\epsilon}\right)\delta.$$

On the other hand, if $\delta > (c-1)\epsilon/(c(c+\epsilon))$, then the error in a solution S_2 obtained by running the c-approximation algorithm for Π is given by

$$c - 1 \leq \left(\frac{c^2 + c\epsilon}{\epsilon}\right)\delta.$$

Therefore, choosing $\beta = (c^2 + c\epsilon)/\epsilon$, we immediately obtain that the solution with larger value, among S_1 and S_2, has error at most $\beta\delta$. Thus, this reduction is indeed an E-reduction.

4. Generic reductions and an approximation hierarchy. In this section we describe a generic technique for turning a hardness result into an approximation-preserving reduction.

We start by listing the kind of constraints imposed on the hardness reduction, the approximation class, and the optimization problem. We will observe at the end that these restrictions are obeyed by all known hardness results and the corresponding approximation classes.

DEFINITION 11 (additive problems). *An* NPO *problem* Π *is said to be* additive *if there exists an operator* $+$ *and a polynomial-time computable function* f *such that* $+$ *maps a pair of instances* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *to an instance* $\mathcal{I}_1 + \mathcal{I}_2$ *such that* $OPT(\mathcal{I}_1 + \mathcal{I}_2) = OPT(\mathcal{I}_1) + OPT(\mathcal{I}_2)$*, and* f *maps a solution* s *to* $\mathcal{I}_1 + \mathcal{I}_2$ *to a pair of solutions* s_1 *and* s_2 *to* $\mathcal{I}_1$ *and* $\mathcal{I}_2$*, respectively, such that* $V(\mathcal{I}_1 + \mathcal{I}_2, s) = V(\mathcal{I}_1, s_1) + V(\mathcal{I}_2, s_2)$.

DEFINITION 12 (downward closed family). *A family of functions* $F = \{f : \mathcal{Z}^+ \to \mathcal{Z}^+\}$ *is said to be* downward closed *if for all* $g \in F$ *and for all constants* c *(and in particular for all integers* $c > 1$*),* $g'(n) \in O(g(n^c))$ *implies that* $g' \in F$*. A function* g *is said to be* hard *for the family* F *if for all* $g' \in F$*, there exists a constant* c *such that* $g'(n) \in O(g(n^c))$*; the function* g *is said to be* complete *for* F *if* g *is hard for* F *and* $g \in F$.

DEFINITION 13 (F-APX). *For a downward closed family* F*, the class* F-APX *consists of all polynomially bounded optimization problems approximable to within a ratio of* $g(|\mathcal{I}|)$ *for some function* $g \in F$.

DEFINITION 14 (canonical hardness). *An* NP *maximization problem* Π *is said to be* canonically hard *for the class* F-APX *if there exists a transformation* T *mapping instances of* 3-SAT *to instances of* Π*, constants* n_0 *and* c*, and a gap function* G *which is hard for the family* F*, such that given an instance* x *of* 3-SAT *on* $n \geq n_0$ *variables and* $N \geq n^c$*,* $\mathcal{I} = T(x, N)$ *is an instance of* Π *with the following properties.*

- *If* $x \in$ 3-SAT*, then* $OPT(\mathcal{I}) = N$.
- *If* $x \notin$ 3-SAT*, then* $OPT(\mathcal{I}) = N/G(N)$.
- *Given a solution* S *to* $\mathcal{I}$ *with* $V(\mathcal{I}, S) > N/G(N)$*, a truth assignment satisfying* x *can be found in polynomial time.*

In the above definition, the transformation T from 3-SAT to Π is somewhat special in that one can specify the size/optimum of the reduced problem and T can produce a mapped instance of the desired size. This additional property is easily obtained for additive problems by using a sufficient number of additions until the optimum of the reduced problem is close to the target optimum, and then adding a problem of known optimum value to the reduced problem.

Canonical hardness for NP *minimization problems* is analogously defined: $OPT(\mathcal{I}) = N$ when the formula is satisfiable, and $OPT(\mathcal{I}) = NG(N)$ otherwise. Given any solution with value less than $NG(N)$, one can construct a satisfying assignment in polynomial time.

4.1. The reduction.

THEOREM 5. *If* F *is a downward closed family of functions, and an additive* NPO *problem* Ω *is canonically hard for the class* F-APX-PB*, then all problems in* F-APX-PB *E-reduce to* Ω.

Proof. Let Π be a polynomially bounded optimization problem in F-APX, approximable to within $c(.)$ by an algorithm A, and let Ω be a problem shown to be hard to within a factor $G(.)$ where G is hard for F. Let V and V' denote the objective functions of Π and Ω, respectively. We start with the special case where both Π and Ω are maximization problems. We describe the functions f, g and the constant β as

required for an E-reduction.

Let $\mathcal{I} \in \Pi$ be an instance of size n; pick N so that $c(n)$ is $O(G(N))$. To describe our reduction, we need to specify the functions f and g. The function f is defined as follows. Let $m = V(\mathcal{I}, A(\mathcal{I}))$. For each $i \in \{1, \ldots, mc(n)\}$, let L_i denote the NP-language $\{\mathcal{I} | \, OPT(\mathcal{I}) \geq i\}$. Now, for each i, we create an instance $\phi_i \in \Omega$ of size N such that if $\mathcal{I} \in L_i$ then $OPT(\phi_i)$ is N, and it is $N/G(N)$ otherwise. We define $f(\mathcal{I}) = \phi = \sum_i \phi_i$.

We now construct the function g. Given an instance $\mathcal{I} \in \Pi$ and a solution s' to $f(\mathcal{I})$, we compute a solution s to $\mathcal{I}$ in the following manner. We first use A to find a solution s_1. We also compute a second solution s_2 to $\mathcal{I}$ as follows. Let j be the largest index such that the solution s' projects down to a solution s'_j to the instance ϕ_j such that $V'(\phi_j, s'_j) > N/G(N)$. This in turn implies that we can find a solution s_2 to witness $V(\mathcal{I}, s_2) \geq j$. Our solution s is the one among s_1 and s_2 that yields the larger objective function value.

We now show that the reduction is an E-reduction with $\beta = 1 + c(n)/(G(N) - 1)$.

Let $\alpha = OPT(I)/m$. Observe that

$$OPT(I') = Nm \left(\alpha + \frac{c(n)}{G(N)} - \frac{\alpha}{G(N)} \right).$$

Consider the following two cases.

Case 1 [$j \leq m$]: In this case, $V(\mathcal{I}, s) = m$. Since s is a solution to $\mathcal{I}$ of error at most $(\alpha - 1)$, it suffices to argue that the error of s' as a solution to ϕ is at least $(\alpha - 1)/\beta$. We start with the following upper bound on $V(\phi, s')$.

$$V(\phi, s') \leq Nm \left(1 + \frac{c(n)}{G(N)} - \frac{1}{G(N)} \right).$$

Thus the approximation factor achieved by s' is given by

$$\begin{aligned} \mathcal{E}(\phi, s') &\geq \left(\frac{Nm \left(\alpha + \frac{c(n)}{G(N)} - \frac{\alpha}{G(N)} \right)}{Nm \left(1 + \frac{c(n)}{G(N)} - \frac{1}{G(N)} \right)} \right) - 1 \\ &= (\alpha - 1) \left(\frac{G(N) - 1}{G(N) + c(n) - 1} \right) \\ &= \frac{\alpha - 1}{\beta}. \end{aligned}$$

So in this case s_1 (and hence s) is a solution to $\mathcal{I}$ with an error of at most $\beta\epsilon$, if s' is a solution to ϕ with an error of ϵ.

Case 2 [$j \geq m$]: Let $j = \gamma m$. Note that $\gamma > 1$ and that the error of s as a solution to $\mathcal{I}$ is $(\alpha - \gamma)/\gamma$. We bound the value of the solution s' to ϕ as

$$V(\phi, s') \leq Nm \left(\gamma + \frac{c(n)}{G(N)} - \frac{\gamma}{G(N)} \right),$$

and its error as

$$\mathcal{E}(\phi, s') = \left(\frac{\alpha + \frac{c(n)}{G(N)} - \frac{\alpha}{G(N)}}{\gamma + \frac{c(n)}{G(N)} - \frac{\gamma}{G(N)}} \right) - 1$$

$$= \left(\frac{\alpha-\gamma}{\gamma}\right)\left(\frac{1}{1+\frac{c(n)}{\gamma(G(N)-1)}}\right)$$
$$\geq \left(\frac{\alpha-\gamma}{\gamma}\right)\frac{1}{\beta}.$$

The final inequality follows from the fact that

$$1+\frac{c(n)}{\gamma(G(N)-1)} \leq 1+\frac{c(n)}{(G(N)-1)} = \beta.$$

Thus, in this case also, we find that s (by virtue of s_2) is a solution to $\mathcal{I}$ of error at most $\beta\epsilon$ if s' is a solution to ϕ of error ϵ.

We now consider the more general cases where Π and Ω are not both maximization problems. For the case where both are minimization problems, the above transformation works with one minor change. When creating ϕ_i, the NP language consists of instances $(\mathcal{I}, i)$ such that there exists s with $V(\mathcal{I}, s) \leq i$.

For the case where Π is a minimization problem and Ω is a maximization problem, we first E-reduce Π to a maximization problem Π' and then proceed as before. The reduction proceeds as follows. Since Π is a polynomially bounded optimization problem, we can compute an upper bound on the value of any solution s to an instance $\mathcal{I}$. Let m be such a bound for an instance $\mathcal{I}$. The objective function of Π' on the instance $\mathcal{I}$ is defined as $V'(\mathcal{I}, s) = \lfloor 2m^2/V(\mathcal{I}, s)\rfloor$. To begin with, it is easy to verify that $\Pi \in F$-APX implies $\Pi' \in F$-APX.

Let s be a solution to instance $\mathcal{I}$ of Π of error β. We will show that s as a solution to instance $\mathcal{I}$ of Π' has an error of at least $\beta/2$. Assume, without loss of generality, that $\beta \neq 0$. Then

$$V(\mathcal{I}, s) - OPT(\mathcal{I}) = \beta\, OPT(\mathcal{I}) \geq 1.$$

Multiplying by $2m^2/(OPT(\mathcal{I})V(\mathcal{I}, s))$, we get

$$\frac{2m^2}{OPT(\mathcal{I})} - \frac{2m^2}{V(\mathcal{I}, s)} = \beta\frac{2m^2}{V(\mathcal{I}, s)} \geq 2.$$

This implies that

$$\frac{2m^2}{OPT(\mathcal{I})} - \frac{2m^2}{V(\mathcal{I}, s)} \geq 1 + \frac{1}{2}\times\frac{2m^2}{OPT(\mathcal{I})} - \frac{2m^2}{V(\mathcal{I}, s)}$$
$$= 1 + \frac{\beta}{2}\times\frac{2m^2}{V(\mathcal{I}, s)}.$$

Upon rearranging,

$$V'(\mathcal{I}, s) \leq \frac{1}{(1+\beta/2)}\left(\frac{2m^2}{OPT(\mathcal{I})} - 1\right) \leq \frac{1}{(1+\beta/2)}\left\lfloor\frac{2m^2}{OPT(\mathcal{I})}\right\rfloor.$$

Thus the reduction from Π to Π' is an E-reduction.

Finally, the last remaining case, i.e., Π being a maximization problem and Ω being a minimization problem, is dealt with similarly: we transform Π into a minimization problem Π'. □

Remark 6. This theorem appears to merge two different notions of the relative ease of approximation of optimization problems. One such notion would consider a problem Π_1 easier than Π_2 if there exists an approximation-preserving reduction from Π_1 to Π_2. A different notion would regard Π_1 to be easier than Π_2 if one seems to have a better factor of approximation than the other. The above statement essentially states that these two comparisons are indeed the same. For instance, the MAX CLIQUE problem and the CHROMATIC NUMBER problem, which are both in poly-APX, are interreducible to each other. The above observation motivates the search for other interesting function classes f, for which the class f-APX may contain interesting optimization problems.

4.2. Applications. The following is a consequence of Theorem 5.

THEOREM 6.

1. $\overline{\text{RMAX}(2)} = \text{poly-APX}$.
2. *If* SET COVER *is canonically hard to approximate to within a factor of* $\Omega(\log n)$*, then* $\text{log-APX} = \overline{\text{MIN F}^+\Pi_2(1)}$.

We briefly sketch the proof of this theorem. The hardness reduction for MAX SAT and CLIQUE are canonical [2, 11]. The classes APX-PB, poly-APX, log-APX are expressible as classes F-APX for downward closed function families. The problems MAX SAT, MAX CLIQUE, and SET COVER are additive. Thus, we can now apply Theorem 5.

Remark 7. We would like to point out that almost all known instances of hardness results seem to be shown for problems which are additive. In particular, this is true for all MAX SNP problems, MAX CLIQUE, CHROMATIC NUMBER, and SET COVER. Two cases where a hardness result does not seem to directly apply to an additive problem is that of LONGEST PATH [17] and BIN PACKING. In the former case, the closely related LONGEST S-T PATH problem is easily seen to be additive, and the hardness result essentially stems from this problem. As for the case of BIN PACKING, which does not admit a PTAS, the hardness result is not of a multiplicative nature, and in fact this problem can be approximated to within arbitrarily small factors, provided a small additive error term is allowed. This yields a reason why this problem will not be additive. Lastly, the most interesting optimization problems which do not seem to be additive are problems related to GRAPH BISECTION or PARTITION, and these also happen to be notable instances where no hardness of approximation results have been achieved!

5. Local search and MAX SNP. In this section we present a formal definition of the paradigm of nonoblivious local search. The idea of nonoblivious local search has been implicitly present in some well-known techniques such as the interior-point methods. We will formalize this idea in the context of MAX SNP and illustrate its application to MAX SNP problems. Given a MAX SNP problem Π, recall that the goal is to find a structure S which maximizes the objective function: $V(\mathcal{I}, S) = \sum_{\vec{x}} \Phi(\mathcal{I}, S, \vec{x})$. In the subsequent discussion, we view S as a k-dimensional boolean vector.

5.1. Classical local search. We start by reviewing the standard mechanism for constructing a local search algorithm. A δ-local algorithm $\mathcal{A}$ for Π is based on a *distance function* $\mathcal{D}(S_1, S_2)$ which is the Hamming distance between two k-dimensional vectors. The δ-*neighborhood* of a structure S is given by $N(S, \delta) = \{S' \subseteq U^n \mid D(S, S') \leq \delta\}$, where U is the universe. A structure S is called δ-*optimal* if $\forall S' \in N(S, \delta)$, we have $V(\mathcal{I}, S) \geq V(\mathcal{I}, S')$. The algorithm computes a δ-optimum

by performing a series of greedy improvements to an initial structure S_0, where each iteration moves from the current structure S_i to some $S_{i+1} \in N(S_i, \delta)$ of better value (if any). For constant δ, a δ-local search algorithm for a polynomially bounded NPO problem runs in polynomial time because:

- each local change is polynomially computable, and
- the number of iterations is polynomially bounded since the value of the objective function improves monotonically by an integral amount with each iteration, and the optimum is polynomially bounded.

5.2. Nonoblivious local search. A nonoblivious local search algorithm is based on a 3-tuple $\langle S_0, \mathcal{F}, \mathcal{D} \rangle$, where S_0 is the initial solution structure which must be independent of the input, $\mathcal{F}(\mathcal{I}, S)$ is a real-valued function referred to as the *weight function*, and $\mathcal{D}$ is a real-valued *distance function* which returns the distance between two structures in some appropriately chosen metric. The weight function $\mathcal{F}$ should be such that the number of distinct values taken by $\mathcal{F}(\mathcal{I}, S)$ is polynomially bounded in the input size. Moreover, the distance function $\mathcal{D}$ should be such that given a structure S and a fixed δ, $N(S, \delta)$ can be computed in time polynomial in $|S|$. Then, as in classical local search, for constant δ, a nonoblivious δ-local algorithm terminates in time polynomial in the input size.

The classical local search paradigm, which we call *oblivious local search*, makes the natural choice for the function $\mathcal{F}(\mathcal{I}, S)$ and the distance function $\mathcal{D}$, i.e., it chooses them to be $V(\mathcal{I}, S)$ and the Hamming distance. However, as we show later, this choice does not always yield a good approximation ratio. We now formalize our notion of this more general type of local search.

DEFINITION 15 (nonoblivious local search algorithm). *A* nonoblivious local search *algorithm is a δ-local search algorithm whose weight function is defined to be*

$$\mathcal{F}(\mathcal{I}, S) = \sum_{\vec{x}} \sum_{i=1}^{r} p_i \Phi_i(\mathcal{I}, S, \vec{x}) \ ,$$

where r is a constant, Φ_i's are quantifier-free first-order formulas, and the profits p_i are real constants. The distance function $\mathcal{D}$ is an arbitrary polynomial-time computable function.

A nonoblivious local search can be implemented in polynomial time in much the same way as the oblivious local search. Note that the we are only considering polynomially bounded weight functions and the profits p_i are fixed independent of the input size. In general, the nonoblivious weight functions do not direct the search in the direction of the actual objective function. In fact, as we will see, this is exactly the reason why they are more powerful and allow for better approximations.

We now define two classes of NPO problems.

DEFINITION 16 (oblivious GLO). *The class of problems* OBLIVIOUS GLO *consists of all* NPO *problems which can be approximated within constant factors by an oblivious δ-local search algorithm for some constant δ.*

DEFINITION 17 (nonoblivious GLO). *The class of problems* NONOBLIVIOUS GLO *consists of all* NPO *problems which can be approximated within constant factors by a nonoblivious δ-local search algorithm for some constant δ.*

Remark 8. It would be perfectly reasonable to allow weight functions that are nonlinear, but we stay with the above definition for the purposes of this paper. Allowing only a constant number of predicates in the weight functions enables us to prevent

the encoding of arbitrarily complicated approximation algorithms. The structure S is a k-dimensional vector, and so a natural metric for the distance function $\mathcal{D}$ is the Hamming distance. In fact, classical local search is indeed based on the Hamming metric and this is useful in proving negative results for the paradigm. In contrast, the definition of nonoblivious local search allows for other distance functions, but we will use only the Hamming metric in proving positive results in the remainder of this paper. However, we have found that it is sometimes useful to modify this, for example, by modifying the Hamming distance so that the complement of a vector is considered to be at distance 1 from it. Finally, it is sometimes convenient to assume that the local search makes the best possible move in the bounded neighborhood, rather than an arbitrary move which improves the weight function. We believe that this does not increase the power of nonoblivious local search.

6. The power of nonoblivious local search. We will show that there exists a choice of a nonoblivious weight function for MAX k-SAT such that any assignment which is 1-optimal with respect to this weight function yields a performance ratio of $2^k/(2^k-1)$ with respect to the optimal. But first, we obtain tight bounds on the performance of oblivious local search for MAX 2-SAT, establishing that its performance is significantly weaker than the best-known result even when allowed to search exponentially large neighborhoods. We use the following notation: for any fixed truth assignment $\vec{Z}$, S_i is the set of clauses in which exactly i literals are true; and, for a set of clauses S, $W(S)$ denotes the total weight of the clauses in S.

6.1. Oblivious local search for MAX 2-SAT**.** We show a strong separation in the performance of oblivious and nonoblivious local search for MAX 2-SAT. Suppose we use a δ-local strategy with the weight function $\mathcal{F}$ being the total weight of the clauses satisfied by the assignment, i.e., $\mathcal{F} = W(S_1) + W(S_2)$. The following theorem shows that for any $\delta = o(n)$, an oblivious δ-local strategy cannot deliver a performance ratio better than $3/2$. This is rather surprising given that we are willing to allow near-exponential time for the oblivious algorithm.

THEOREM 7. *The asymptotic performance ratio for an oblivious δ-local search algorithm for* MAX 2-SAT *is* $3/2$ *for any positive $\delta = o(n)$. This ratio is still bounded by* $5/4$ *when δ may take any value less than $n/2$.*

Proof. We first show the existence of an input instance for MAX 2-SAT which may elicit a relatively poor performance ratio for any δ-local algorithm provided $\delta = o(n)$. In our construction of such an input instance, we assume that $n \geq 2\delta + 1$. The input instance comprises a disjoint union of four sets of clauses, say $\Gamma_1, \Gamma_2, \Gamma_3$, and Γ_4, defined as below:

$$\Gamma_1 = \bigcup_{1 \leq i < j \leq n} (z_i + \overline{z}_j),$$

$$\Gamma_2 = \bigcup_{1 \leq i < j \leq n} (\overline{z}_i + z_j),$$

$$\Gamma_3 = \bigcup_{0 \leq i \leq \delta} \zeta_{2i+1},$$

$$\Gamma_4 = \bigcup_{2\delta+2 \leq i \leq n} \zeta_i,$$

$$\zeta_i = \bigcup_{i < j \leq n} (\overline{z}_i + \overline{z}_j).$$

Clearly, $|\Gamma_1| = |\Gamma_2| = \binom{n}{2}$ and $|\Gamma_3| + |\Gamma_4| = \binom{n}{2} - n\delta + \delta(\delta+1)$. Without loss of generality, assume that the current input assignment is $\vec{Z} = (1, 1, \ldots, 1)$. This satisfies all clauses in Γ_1 and Γ_2. But none of the clauses in Γ_3 and Γ_4, are satisfied. If we flip the assignment of values to any $k \leq \delta$ variables, it would unsatisfy precisely $k(n-k)$ clauses in $\Gamma_1 + \Gamma_2$. This is the number of clauses in $\Gamma_1 + \Gamma_2$ where a flipped variable occurs with an unflipped variable.

On the other hand, flipping the assigned values of any $k \leq \delta$ variables can satisfy at most $k(n-k)$ clauses in $\Gamma_3 + \Gamma_4$, as we next show.

Let $\Pi(n, \delta)$ denote the set of clauses on n variables given by $\bigcup_{0 \leq i \leq \delta} \zeta_{2i+1} + \bigcup_{2\delta+2 \leq i \leq n} \zeta_i$, where $2\delta + 1 \leq n$. We claim the following.

LEMMA 1. *Any assignment of values to the n variables such that at most $k \leq \delta$ variables have been assigned value false, can satisfy at most $k(n-k)$ clauses in $\Pi(n, \delta)$.*

Proof. We prove by simultaneous induction on n and δ that the statement is true for any instance $\Pi(n, \delta)$ where n and δ are nonnegative integers such that $2\delta + 1 \leq n$. The base case includes $n = 1$ and $n = 2$ and is trivially verified to be true for the only allowable value of δ, namely $\delta = 0$. We now assume that the statement is true for any instance $\Pi(n', \delta')$ such that $n' < n$ and $2\delta' + 1 \leq n'$. Consider now the instance $\Pi(n, \delta)$. The statement is trivially true for $\delta = 0$. Now consider any $\delta > 0$ such that $2\delta + 1 \leq n$. Let $\{z_{j_1}, z_{j_2}, \ldots z_{j_k}\}$ be any choice of $k \leq \delta$ variables such that $j_p < j_q$ for $p < q$. Again the assertion is trivially true if $k = 0$ or $k = 1$. We assume that $k \geq 2$ from now on. If we delete all clauses containing the variables z_1 and z_2 from $\Pi(n, \delta)$, we get the instance $\Pi(n-2, \delta-1)$. We now consider three cases.

Case 1 [$j_1 \geq 3$]: In this case, we are reduced to the problem of finding an upper bound on the maximum number of clauses satisfied by setting any k variables to false in $\Pi(n-2, \delta-1)$. If $k \leq \delta - 1$, we may use the inductive hypothesis to conclude that no more than $(n-2-k)(k)$ clauses will be satisfied. Thus the assertion holds in this case. However, we may not directly use the inductive hypothesis if $k = \delta$. But in this case we observe that since by the inductive hypothesis, setting any $k-1$ variables in $\Pi(n-2, \delta-1)$ to false satisfies at most $(n-2-(k-1))(k-1)$ clauses, assigning the value false to any set of k variables can satisfy at most

$$(n-2-(k-1))(k-1) + \frac{1}{k-1}(n-2-(k-1))(k-1) = (n-k)k - k^2$$

clauses. Hence the assertion holds in this case also.

Case 2 [$j_1 = 2$]: In this case, z_{j_1} satisfies one clause and the remaining $k-1$ variables satisfy at most $(n-2-(k-1))(k-1)$ clauses by the inductive hypothesis on $\Pi(n-2, \delta-1)$. Adding up the two terms, we see that the assertion holds.

Case 3 [$j_1 = 1$]: We analyze this case based on whether $j_2 = 2$ or $j_2 \geq 3$. If $j_2 = 2$, then z_1 and z_2 together satisfy precisely $n-1$ clauses, and the remaining $k-2$ variables satisfy at most $(n-2-(k-2))(k-2)$ clauses using the inductive hypothesis. Thus the assertion still holds. Otherwise, z_1 satisfies precisely $n-1$ clauses and the remaining $k-1$ variables satisfy no more than $(n-1-(k-1))(k-1)$ clauses using the inductive hypothesis. Summing up the two terms, we get $(n-k)k$ as the upper bound on the total number of clauses satisfied. Thus the assertion holds in this case also.

To see that this bound is tight, simply consider the situation when the k variables set to false are $z_1, z_3, \ldots, z_{2k-1}$, for any $k \leq \delta$. The total number of clauses satisfied is given by $\sum_{i=1}^{k} |\zeta_{2i-1}| = (n-k)k$. ∎

Assuming that each clause has the same weight, Lemma 1 allows us to conclude that a δ-local algorithm cannot increase the total weight of satisfied clauses with this starting assignment. An optimal assignment, on the other hand, can satisfy all the clauses by choosing the vector $\vec{Z} = (0, 0, \ldots, 0)$. Thus the performance ratio of a δ-local algorithm, say R_δ, is bounded as

$$R_\delta = \frac{|\Gamma_1| + |\Gamma_2| + |\Gamma_3| + |\Gamma_4|}{|\Gamma_1| + |\Gamma_2|} \leq \frac{3\binom{n}{2} + \delta(\delta+1) - \delta n}{2\binom{n}{2}}.$$

For any $\delta = o(n)$, this ratio asymptotically converges to 3/2. We next show that this bound is tight since a 1-local algorithm achieves it. However, before we do so, we make another intriguing observation, namely, that for any $\delta < n/2$, the ratio R_δ is bounded by 5/4.

Now, to see that a 1-local algorithm ensures a performance ratio of 3/2, consider any 1-optimal assignment $\vec{Z}$ and let α_i denote the set of clauses containing the variable z_i such that no literal in any clause of α_i is satisfied by $\vec{Z}$. Similarly, let β_i denote the set of clauses containing the variable z_i such that precisely one literal is satisfied in any clause in β_i, and furthermore, it is precisely the literal containing the variable z_i. If we complement the value assigned to the variable z_i, it is exactly the set of clauses in α_i which becomes satisfied and the set of clauses in β_i which is no longer satisfied. Since $\vec{Z}$ is 1-optimal, it must be the case that $W(\alpha_i) \leq W(\beta_i)$. If we sum up this inequality over all the variables, then we get the inequality $\sum_{i=1}^{n} W(\alpha_i) \leq \sum_{i=1}^{n} W(\beta_i)$. We observe that $\sum_{i=1}^{n} W(\alpha_i) = 2W(S_0)$ and $\sum_{i=1}^{n} W(\beta_i) = W(S_1)$ because each clause in S_0 gets counted twice while each clause in S_1 gets counted exactly once. Thus the fractional weight of the number of clauses not satisfied by a 1-local assignment is bounded as

$$\frac{W(S_0)}{W(S_0) + W(S_1) + W(S_2)} \leq \frac{W(S_0)}{3W(S_0) + W(S_2)} \leq \frac{W(S_0)}{3W(S_0)} = \frac{1}{3}.$$

Hence the performance ratio achieved by a 1-local algorithm is bounded from above by 3/2. Combining this with the upper bound derived earlier, we conclude that $R_1 = 3/2$. This concludes the proof of the theorem. □

6.2. Nonoblivious local search for MAX 2-SAT**.** We now illustrate the power of nonoblivious local search by showing that it achieves a performance ratio of 4/3 for MAX 2-SAT, using 1-local search with a simple nonoblivious weight function.

THEOREM 8. *Nonoblivious* 1*-local search achieves a performance ratio of* 4/3 *for* MAX 2-SAT.

Proof. We use the nonoblivious weight function

$$\mathcal{F}(\mathcal{I}, \vec{Z}) = \frac{3}{2}W(S_1) + 2W(S_2).$$

Consider any assignment $\vec{Z}$ which is 1-optimal with respect to this weight function. Without loss of generality, we assume that the variables have been renamed such that each unnegated literal gets assigned the value true. Let $P_{i,j}$ and $N_{i,j}$, respectively, denote the total weight of clauses in S_i containing the literals z_j and $\overline{z}_j$, respectively. Since $\vec{Z}$ is a 1-optimal assignment, each variable z_j must satisfy the following equation.

$$-\frac{1}{2}P_{2,j} - \frac{3}{2}P_{1,j} + \frac{1}{2}N_{1,j} + \frac{3}{2}N_{0,j} \leq 0.$$

Summing this inequality over all the variables and using

$$\sum_{j=1}^{n} P_{1,j} = \sum_{j=1}^{n} N_{1,j} = W(S_1),$$
$$\sum_{j=1}^{n} P_{2,j} = 2W(S_2),$$
$$\sum_{j=1}^{n} N_{0,j} = 2W(S_0),$$

we obtain the following inequality:

$$W(S_2) + W(S_1) \geq 3W(S_0).$$

This immediately implies that the total weight of the unsatisfied clauses at this local optimum is no more than 1/4 times the total weight of all the clauses. Thus, this algorithm ensures a performance ratio of 4/3. □

Remark 9. The same result can be achieved by using the oblivious weight function and instead modifying the distance function so that it corresponds to distances in a hypercube augmented by edges between nodes whose addresses complement each other.

6.3. Generalization to MAX *k*-SAT. We can also design a nonoblivious weight function for MAX k-SAT such that a 1-local strategy ensures a performance ratio of $2^k/(2^k - 1)$. The weight function $\mathcal{F}$ will be of the form $\mathcal{F} = \sum_{i=0}^{k} c_i W(S_i)$ where the coefficients c_i's will be specified later.

THEOREM 9. *Nonoblivious* 1*-local search achieves a performance ratio of* $2^k/(2^k - 1)$ *for* MAX k-SAT.

Proof. Again, without loss of generality, we will assume that the variables have been renamed so that each unnegated literal is assigned true under the current truth assignment. Thus the set S_i is the set of clauses with i unnegated literals.

Let $\Delta_i = c_i - c_{i-1}$ and let $\frac{\partial \mathcal{F}}{\partial z_j}$ denote the change in the current weight when we flip the value of z_j, that is, set it to 0. It is easy to verify the following equation:

$$\frac{\partial \mathcal{F}}{\partial z_j} = -\Delta_k P_{k,j} + \sum_{i=k}^{2} (\Delta_i N_{i-1,j} - \Delta_{i-1} P_{i-1,j}) + \Delta_1 N_{0,j}. \tag{6.1}$$

Thus when the algorithm terminates, we know that $\frac{\partial \mathcal{F}}{\partial z_j} \leq 0$, for $1 \leq j \leq n$. Summing over all values of j, and using the fact $\sum_{j=1}^{n} P_{i,j} = iW(S_i)$ and $\sum_{j=1}^{n} N_{i,j} = (k - i)W(S_i)$ we get the following inequality.

$$k\Delta_k W(S_k) + \sum_{i=k-1}^{2} (i\Delta_i - (k - i)\Delta_{i+1}) W(S_i) \geq k\Delta_1 W(S_0). \tag{6.2}$$

We now determine the values of Δ_i's such that the coefficient of each term on the left-hand side is unity. It can be verified that

$$\Delta_i = \frac{1}{(k - i + 1)\binom{k}{i-1}} \sum_{j=0}^{k-i} \binom{k}{j}$$

achieves this goal. Thus the coefficient of $W(S_0)$ on the right-hand side of equation (6.2) is $2^k - 1$. Clearly, the weight of the clauses not satisfied is bounded by $1/2^k$ times the total weight of all the clauses. It is worthwhile to note that this is regardless of the value chosen for the coefficient c_0. □

7. Local search for CSP and MAX SNP. We now introduce a class of constraint satisfaction problems such that the problems in MAX SNP are exactly equivalent to the problems in this class. Furthermore, every problem in this class can be approximated to within a constant factor by a nonoblivious local search algorithm.

7.1. Constraint satisfaction problems. The connection between the syntactic description of optimization problems and their approximability through nonoblivious local search is made via a problem called MAX k-CSP which captures all the problems in MAX SNP as a special case.

DEFINITION 18 (k-ary constraint). *Let $Z = \{z_1, \ldots, z_n\}$ be a set of boolean variables. A k-ary constraint on Z is $C = (V; P)$, where V is a size k subset of Z, and $P : \{T, F\}^k \to \{T, F\}$ is a k-ary boolean predicate.*

DEFINITION 19 (MAX k-CSP). *Given a collection C_1, ..., C_m of weighted k-ary constraints over the variables $Z = \{z_1, \ldots, z_n\}$, the* MAX k-CSP *problem is to find a truth assignment satisfying a maximum weight subcollection of the constraints.*

The following theorem shows that the MAX k-CSP problem is a "universal" MAX SNP problem, in that it contains as special cases all problems in MAX SNP.

THEOREM 10.

1. *For fixed k,* MAX k-CSP $\in$ MAX SNP.
2. *Let* $\Pi \in$ MAX SNP. *Then, for some constant k,* Π *is a* MAX k-CSP *problem. Moreover, the* k-CSP *instance corresponding to any instance of this problem can be computed in polynomial time.*

Proof. The proof of part 2 is implicit in Theorem 1 in [23], and so we concentrate on proving part 1. Our goal is to obtain a representation of the k-CSP problem in the MAX SNP syntax:

$$\max_S |\{x \mid \Phi(\mathcal{I}, S, x)\}|.$$

The input structure is $\mathcal{I} = (Z \cup \{T, F\} \cup \text{MAX}; \{\text{ARG}, \text{EVAL}\})$, where $Z = \{z_1, \ldots, z_n\}$, MAX contains the integers $[1, \max\{k, n, m\}]$, the predicate ARG encodes the sets V_i, and the predicate EVAL encodes the predicates P_i, as described below.

- ARG(r, s, z_t) is a 3-ary predicate which is true if and only if the rth argument of C_s is the variable z_t, for $1 \le r \le k$, $1 \le s \le m$, and $1 \le t \le n$.
- EVAL$(s, v_1, \ldots, v_k)$ is a $(k+1)$-ary predicate which is true if and only if $P_s(v_1, \ldots, v_k)$ evaluates to true, for $1 \le s \le m$ and all $v_i \in \{T, F\}$.

The structure S is defined as $(Z; \{\text{TRUE}\})$, where TRUE is a unary predicate which denotes an assignment of truth values to the variables in Z. The vector x has $k+1$ components which will be called x_1, ..., x_k and s, for convenience. The intention is that the x_i's refer to the arguments of the sth constraint.

All that remains is to specify the quantifier-free formula Φ. The basic idea is that $\Phi(\mathcal{I}, S, x)$ should evaluate to true if and only if the following two conditions are satisfied:

- the arguments of the constraint C_s are given by the variables x_1, ..., x_k, in that order, and
- the values given to these variables under the truth assignment specified by S are such that the constraint is satisfied.

The formula Φ is given by the following expression, with the two subformulas ensuring these two conditions.

$$\left(\bigwedge_{r=1}^{k} \text{ARG}(r, s, x_r)\right) \wedge \left(\bigvee_{v_1,\ldots,v_k \in \{T,F\}} \left(\text{EVAL}(s, v_1, \ldots, v_k) \wedge \left(\bigwedge_{r=1}^{k} v_r \Leftrightarrow \text{TRUE}(x_r)\right)\right)\right).$$

It is easy to see that the first subformula has the desired effect of checking that the x_r's correspond to the arguments of C_s. The second subformula considers all possible truth assignments to these k variables and checks that the particular set of values assigned by the structure S will make P_s evaluate to true.

For a fixed structure S, there is exactly one choice of x per constraint that could make Φ evaluate to true, and this happens if and only if that constraint is satisfied. Thus, the value of the solution given by any particular truth assignment structure S is exactly the number of constraints that are satisfied. This shows that the MAX SNP problem always has the same value as intended in the k-CSP problem.

Finally, there are still a few things which need to be checked to ensure that this is a valid MAX SNP formulation. Notice that all the predicates are of bounded arity and the structures consist of a bounded number of such predicates, i.e., independent of the input size which is given by MAX. Further, although the length of the formula is exponential in k, it is independent of the input. □

7.2. Nonoblivious local search for MAX k-CSP. A suitable generalization of the nonoblivious local search algorithm for MAX k-SAT yields the following result.

THEOREM 11. *A nonoblivious* 1*-local search algorithm has performance ratio* 2^k *for* MAX k-CSP.

Proof. We use an approach similar to the one used in the previous section to design a nonoblivious weight function $\mathcal{F}$ for the weighted version of the MAX k-CSP problem such that a 1-local algorithm yields a 2^k performance ratio to this problem.

We consider only the constraints with at least one satisfying assignment. Each such constraint can be replaced by a monomial which is the conjunction of some k literals such that when the monomial evaluates to true the corresponding literal assignment represents a satisfying assignment for the constraint. Furthermore, each such monomial has precisely one satisfying assignment. We assign to each monomial the weight of the constraint it represents. Thus any assignment of variables which satisfies monomials of total weight W_0 also satisfies constraints in the original problem of total weight W_0.

Let S_i denote the monomials with i true literals and assume that the weight function $\mathcal{F}$ is of the form $\sum_{i=1}^{k} c_i W(S_i)$. Thus, assuming that the variables have been renamed so that the current assignment gives value true to each variable, we know that for any variable z_j, $\frac{\partial \mathcal{F}}{\partial z_j}$ is given by equation (6.1). As before, using the fact that for any 1-optimal assignment, $\frac{\partial \mathcal{F}}{\partial z_j} \leq 0$ for $1 \leq j \leq n$, and summing over all values of j, we can write the following inequality.

$$k\Delta_1 W(S_0) + \sum_{i=2}^{k-1}((k-i)\Delta_{i+1} - i\Delta_i)W(S_i) \leq k\Delta_k W(S_k)\ . \tag{7.1}$$

We now determine the values of Δ_i's such that the coefficient of each term on the left-hand side is unity. It can be verified that

$$\Delta_i = \frac{1}{i\binom{k}{i}} \sum_{j=0}^{i-1} \binom{k}{j}$$

achieves this goal. Thus the coefficient of $W(S_k)$ on the right-hand side of equation (6.1) is $2^k - 1$. Clearly, the total weight of clauses satisfied is at least $1/2^k$ times the total weight of all the clauses with at least one satisfiable assignment. □

We conclude as in the following theorem.

THEOREM 12. *Every optimization problem* $\Pi \in$ MAX SNP *can be approximated to within some constant factor by a (uniform) nonoblivious* 1*-local search algorithm, i.e.,*

$$\text{MAX SNP} \subseteq \text{NONOBLIVIOUS GLO}.$$

For a problem expressible as k-CSP*, the performance ratio is at most* 2^k*.*

8. Nonoblivious versus oblivious GLO. In this section, we show that there exist problems for which no constant-factor approximation can be obtained by any δ-local search algorithm with oblivious weight function, even when we allow δ to grow with the input size. However, a simple 1-local search algorithm using an appropriate nonoblivious weight function can ensure a constant performance ratio.

8.1. MAX 2-CSP. The first problem is an instance of MAX 2-CSP where we are given a collection of monomials such that each monomial is an "and" of precisely two literals. The objective is to find an assignment to maximize the number of monomials satisfied.

We show an instance of this problem such that for every $\delta = o(n)$, there exists an instance one of whose local optima has value that is a vanishingly small fraction of the global optimum.

The input instance consists of a disjoint union of two sets of monomials, say Γ_1 and Γ_2, defined as below:

$$\Gamma_1 = \bigcup_{1 \le i < j \le n} (\overline{z}_i \wedge \overline{z}_j),$$

$$\Gamma_2 = \bigcup_{1 \le i \le \delta} \bigcup_{i < j \le n} (z_i \wedge z_j).$$

Clearly, $|\Gamma_1| = \binom{n}{2}$ and $|\Gamma_2| = n\delta - \binom{\delta+1}{2}$. Consider the truth assignment $\vec{Z} = (1, 1, \ldots, 1)$. It satisfies all monomials in Γ_2 but none of the monomials in Γ_1. We claim that this assignment is δ-optimal with respect to the oblivious weight function. To see this, observe that complementing the value of any $p \le \delta$ variables will unsatisfy at least $\delta p/2$ monomials in Γ_2 for any $\delta = o(n)$. On the other hand, this will satisfy precisely $\binom{p}{2}$ monomials in Γ_1. For any $p \le \delta$, we have $(\delta p)/2 \ge \binom{p}{2}$, and so Z is a δ-local optimum.

The optimal assignment, on the other hand, namely $\vec{Z}_{OPT} = (0, 0, \ldots, 0)$, satisfies all monomials in Γ_1. Thus, for $\delta < n/2$, the performance ratio achieved by any δ-local algorithm is no more than $\binom{n}{2}/(n\delta - \binom{\delta+1}{2})$ which asymptotically diverges to infinity for any $\delta = o(n)$. We have already seen in section 7 that a 1-local nonoblivious algorithm ensures a performance ratio of 4 for this problem. Since this problem is in MAX SNP, we obtain the following theorem.

THEOREM 13. *There exist problems in* MAX SNP *such that for* $\delta = o(n)$*, no* δ*-local oblivious algorithm can approximate them to within a constant performance ratio, i.e.,*

$$\text{MAX SNP} \not\subseteq \text{OBLIVIOUS GLO}.$$

8.2. Vertex cover. Ausiello and Protasi [4] have shown that VERTEX COVER does not belong to the class GLO and, hence, there does not exist any constant δ such that an oblivious δ-local search algorithm can compute a constant factor approximation. In fact, their example can be used to show that for any $\delta = o(n)$, the performance ratio ensured by δ-local search asymptotically diverges to infinity. However, we show that there exists a rather simple nonoblivious weight function which ensures a factor 2 approximation via a 1-local search. In fact, the algorithm simply enforces the behavior of the standard approximation algorithm which iteratively builds a vertex cover by simply including both endpoints of any currently uncovered edge.

We assume that the input graph G is given as a structure $(V, \{E\})$ where V is the set of vertices and $E \subseteq V \times V$ encodes the edges of the graph. Our solution is represented by a 2-ary predicate M which is iteratively constructed so as to represent a maximal matching. Clearly, the endpoints of any maximal matching constitute a valid vertex cover and such a vertex cover can be at most twice as large as any other vertex cover in the graph. Thus M is an encoding of the vertex cover computed by the algorithm.

The algorithm starts with M initialized to the empty relation, and at each iteration, at most one new pair is included in it. The nonoblivious weight function used is as below:

$$\mathcal{F}(\mathcal{I}, M) = \sum_{\langle x,y,z\rangle \in V^3} [\Phi_1(x,y,z) - 2\Phi_2(x,y,z) - \Phi_3(x,y,z)],$$

where

$$\begin{aligned}
\Phi_1(x,y,z) &= (M(x,y) \wedge E(x,y) \wedge (x = z)),\\
\Phi_2(x,y,z) &= (M(x,y) \wedge M(x,z)),\\
\Phi_3(x,y,z) &= (M(x,y) \wedge \overline{E(x,y)}).
\end{aligned}$$

Let M encode a valid matching in the graph G. We make the following observations.

- Any relation M' obtained from M by deleting an edge from it, or including an edge which is incident on an edge of M, or including a nonexistent edge, has the property that $\mathcal{F}(\mathcal{I}, M') \leq \mathcal{F}(\mathcal{I}, M)$. Thus in a 1-local search from M, we will never move to a relation M' which does not encode a valid matching of G.
- On the other hand, if a relation M' corresponds to the encoding of a matching in G which is larger than the matching encoded by M, then $\mathcal{F}(\mathcal{I}, M') > \mathcal{F}(\mathcal{I}, M)$. Thus if M does not encode a maximal matching in G, there always exist a relation in its 1-neighborhood of larger weight than itself.

These two observations, combined with the fact that we start with a valid initial matching (the empty matching), immediately allow us to conclude that any 1-optimal relation M always encodes a maximal matching in G. We have established the following.

THEOREM 14. *A* 1*-local search algorithm using the above nonoblivious weight function achieves a performance ratio of* 2 *for the* VERTEX COVER *problem.*

THEOREM 15. GLO *is a* strict *subset of* NONOBLIVIOUS GLO.

As an aside, it can be seen that this algorithm has the same performance starting with an arbitrary initial solution. This is because for any relation M not encoding a matching of the input graph, deleting one of the violating members strictly increases $\mathcal{F}(\mathcal{I}, M)$.

9. The traveling salesman problem. The TSP(1,2) problem is the traveling salesman problem restricted to complete graphs where all edge weights are either 1 or 2; clearly, this satisfies the triangle inequality. Papadimitriou and Yannakakis [24] showed that this problem is hard for MAX SNP. The natural weight function for TSP(1,2), that is, the weight of the tour, can be used to show that a 4-local algorithm yields a 3/2 performance ratio. The algorithm starts with an arbitrary tour, and in each iteration, it checks if there exist two disjoint edges (a, b) and (c, d) on the tour such that deleting them and replacing them with the edges (a, c) and (b, d) yields a tour of lesser cost.

THEOREM 16. *A 4-local search algorithm using the oblivious weight function achieves a* 3/2 *performance ratio for* TSP(1,2).

Proof. Let C be a 4-optimal solution and let π be a permutation such that the vertices in C occur in the order $v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_n}$. Consider any optimal solution O. With each unit cost edge e in O, we associate a unit cost edge e' in C as follows. Let $e = (v_{\pi_i}, v_{\pi_j})$ where $i < j$. If $j = i + 1$ then $e' = e$. Otherwise, consider the edges $e_1 = (v_{\pi_i}, v_{\pi_{i+1}})$ and $e_2 = (v_{\pi_j}, v_{\pi_{j+1}})$ on C. We claim that either e_1 or e_2 must be of unit cost. Suppose not; then the tour C' which is obtained by simply deleting both e_1 and e_2 and inserting the edges e and $f = (v_{\pi_{i+1}}, v_{\pi_{j+1}})$ has cost at least one less than C. But C is 4-optimal and thus this is a contradiction.

Let U_O denote the set of unit cost edges in O and let U_C be the set of unit cost edges in C which form the image of U_O under the above mapping. Since an edge $e' = (v_{\pi_i}, v_{\pi_{i+1}})$ in U_C can only be the image of unit cost edges incident on v_{π_i} in O and since O is a tour, there are at most two edges in U_O which map to e'. Thus $|U_C| \geq |U_O|/2$ and hence

$$\frac{cost(O)}{cost(C)} \geq \frac{|U_O| + 2(n - |U_O|)}{|U_O|/2 + 2(n - |U_O|/2)} \geq \frac{2}{3}. \qquad \square$$

In fact, the above bound can be shown to be tight.

10. Maximum independent sets in bounded-degree graphs. The input instance to the maximum independent set problem in bounded-degree graphs, denoted MIS-B, is a graph G such that the degree of any vertex in G is bounded by a constant Δ. We present an algorithm with performance ratio $(\sqrt{8\Delta^2 + 4\Delta + 1} - 2\Delta + 1)/2$ for this problem when $\Delta \geq 10$.

Our algorithm uses two local search algorithms such that the larger of the two independent sets computed by these algorithms gives us the above-claimed performance ratio. We refer to these two algorithms as $\mathcal{A}_1$ and $\mathcal{A}_2$.

In our framework, the algorithm $\mathcal{A}_1$ can be characterized as a 3-local algorithm with the weight function simply being $|I| - 3|(I \times I) \cap E|$. Thus if we start with I initialized to empty set, it is easy to see that at each iteration, I will correspond to an independent set in G. A convenient way of looking at this algorithm is as follows. We define an $i \leftrightarrow j$ swap to be the process of deleting i vertices from S and including j vertices from the set $V - S$ to the set S. In each iteration, the algorithm $\mathcal{A}_1$ performs either a $0 \leftrightarrow j$ swap where $1 \leq j \leq 3$, or a $1 \leftrightarrow 2$ swap. A $0 \leftrightarrow j$ swap, however, can be interpreted as j applications of $0 \leftrightarrow 1$ swaps. Thus the algorithm may be viewed as executing a $0 \leftrightarrow 1$ swap or a $1 \leftrightarrow 2$ swap at each iteration. The algorithm terminates when neither of these two operations is applicable.

Let I denote the 3-optimal independent set produced by the algorithm $\mathcal{A}_1$. Furthermore, let O be any optimal independent set and let $X = I \cap O$. We make the following useful observations.

- Since for no vertex in I, a $0 \leftrightarrow 1$ swap can be performed, it implies that each vertex in $V - I$ must have at least one incoming edge to I.
- Similarly, since no $1 \leftrightarrow 2$ swaps can be performed, it implies that at most $|I - X|$ vertices in $O - I$ can have precisely one edge coming into I. Thus $|O - X| - |I - X| = |O| - |I|$ vertices in $O - X$ must have at least two edges entering the set I.

A rather straightforward consequence of these two observations is the following lemma.

LEMMA 2. *The algorithm $\mathcal{A}_1$ has performance ratio $(\Delta + 1)/2$ for* MIS-B.

Proof. The above two observations imply that the minimum number of edges entering I from the vertices in $O - X$ is $|I - X| + 2(|O| - |I|)$. On the other hand, the maximum number of edges coming out of the vertices in I to the vertices in $O - X$ is bounded by $|I - X|\Delta$. Thus we must have

$$|I - X|\Delta \geq |I - X| + 2(|O| - |I|) \ .$$

Rearranging, we get

$$\frac{|I|}{|O|} \geq \frac{2}{\Delta + 1} + \frac{|X|(\Delta - 1)}{|O|(\Delta + 1)},$$

which yields the desired result. □

This nearly matches the approximation ratio of $\Delta/2$ due to Hochbaum [15]. It should be noted that the above result holds for a broader class of graphs, viz., k-claw free graphs. A graph is called *k-claw free* if there does not exist an independent set of size k or larger such that all the vertices in the independent set are adjacent to the same vertex. Lemma 2 applies to $(\Delta + 1)$-claw free graphs.

Our next objective is to further improve this ratio by using the algorithm $\mathcal{A}_1$ in combination with the algorithm $\mathcal{A}_2$. The following lemma uses a slightly different counting argument to give an alternative bound on the approximation ratio of the algorithm $\mathcal{A}_1$ when there is a constraint on the size of the optimal solution.

LEMMA 3. *For any real number $c < \Delta$, the algorithm $\mathcal{A}_1$ has performance ratio $(\Delta - c)/2$ for* MIS-B *when the optimal value itself is no more than*

$$\frac{(\Delta - c)|V|}{\Delta + c + 4}.$$

Proof. As noted earlier, each vertex in $V - I$ must have at least one edge coming into the set I, and at least $|O| - |I|$ vertices in O must have at least two edges coming into I. Therefore, the following inequality must be satisfied:

$$|I|\Delta \geq |V| - |I| + |O| - |I| \ .$$

Thus $|I| \geq (|V| + |O|)/(\Delta + 2)$. Finally, observe that

$$\frac{|V| + |O|}{\Delta + 2} \geq \frac{2}{\Delta - c}|O|$$

whenever $|O| \leq (\Delta - c)|V|/(\Delta + c + 4)$. □

The above lemma shows that the algorithm $\mathcal{A}_1$ yields a better approximation ratio when the size of the optimal independent set is relatively small.

The algorithm $\mathcal{A}_2$ is simply the classical greedy algorithm. This algorithm can be conveniently included in our framework if we use directed local search. If we let $N(I)$ denote the set of neighbors of the vertices in I, then the weight function is simply $|I|(\Delta+1)+|V-(I+N(I))|-|(I\times I)\cap E|(\Delta+1)$. It is not difficult to see that starting with an empty independent set, a 1-local algorithm with directed search on the above weight function simply simulates a greedy algorithm. The greedy algorithm exploits the situation when the optimal independent set is relatively large in size. It does so by using the fact that the existence of a large independent set in G ensures a large subset of vertices in G with relatively small average degree. The following two lemmas characterize the performance of the greedy algorithm.

LEMMA 4. *Suppose there exists an independent set $X\subseteq V$ such that the average degree of vertices in X is bounded by α. Then, for any $\alpha\geq 1$, the greedy algorithm produces an independent set of size at least $|X|/(1+\alpha)$.*

Proof. The greedy algorithm iteratively chooses a vertex of smallest degree in the remaining graph and then deletes this vertex and all its neighbors from the graph. We examine the behavior of the greedy algorithm by considering two types of iterations. First consider the iterations in which it picks a vertex outside X. Suppose that in the ith such iteration, it picks a vertex in $V-X$ with exactly k_i neighbors in the set X in the remaining graph. Since each one of these k_i vertices must also have at least k_i edges incident on them, we lose at least k_i^2 edges incident on X. Suppose only p such iterations occur and let $\sum_{i=1}^{p}k_i=x$. We observe that $\sum_{i=1}^{p}k_i^2\leq\alpha|X|$. Second, we consider the iterations when the greedy algorithm selects a vertex in X. Then we do not lose any other vertices in X because X is an independent set. Thus the total size of the independent set constructed by the greedy algorithm is at least $p+q$ where $q=|X|-x$.

By the Cauchy–Schwartz inequality, $\sum_{i=1}^{p}k_i^2\geq x^2/p$. Therefore, we have $(1+\alpha)|X|\geq x^2/p+x$. Rearranging, we obtain that

$$p\geq\frac{x^2}{(1+\alpha)|X|-x}\geq\frac{x^2}{(1+\alpha)|X|}\geq\frac{|X|}{1+\alpha}+\frac{q^2}{(1+\alpha)|X|}-\frac{2q}{1+\alpha}.$$

Thus

$$p+q\geq\frac{|X|}{1+\alpha}+\frac{q^2}{(1+\alpha)|X|}-\frac{2q}{1+\alpha}+q.$$

But $2q/(1+\alpha)\leq q$ for $\alpha\geq 1$, and the result follows. □

LEMMA 5. *For $\Delta\geq 10$ and any nonnegative real number $c\leq 3\Delta-\sqrt{8\Delta^2+4\Delta+1}-1$, the algorithm $\mathcal{A}_2$ has performance ratio $(\Delta-c)/2$ for* MIS-B *when the optimal value itself is at least $((\Delta-c)|V|)/(\Delta+c+4)$.*

Proof. Observe that the average degree of vertices in O is bounded by $(|V-O|\Delta/|O|)$ and thus, using the fact that $|O|\geq(\Delta-c)|V|/(\Delta+c+4)$, we know that the algorithm $\mathcal{A}_2$ computes an independent set of size at least $|O|/(1+\alpha)$ where $\alpha=(4\Delta+2\Delta c)/(\Delta-c)$, and $\alpha\geq 1$ for $c\geq 0$. Hence it is sufficient to determine the range of values c can take such that the following inequality is satisfied:

$$\frac{|O|}{1+\alpha}\geq\left(\frac{2}{\Delta-c}\right)|O|.$$

Substituting the bound on the value of α and rearranging the terms of the equation yields the following quadratic equation:

$$c^2-(6\Delta-2)c+\Delta^2-10\Delta\geq 0\ .$$

Since c must be strictly bounded by Δ, the above quadratic equation is satisfied for any choice of $c \leq 3\Delta - \sqrt{8\Delta^2 + 4\Delta + 1} - 1$ if $\Delta \geq 10$. □

Combining the results of Lemmas 3 and 5 and choosing the largest allowable value for c, we get the following result.

THEOREM 17. *An approximation algorithm which simply outputs the larger of the two independent sets computed by the algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ has performance ratio $(\sqrt{8\Delta^2 + 4\Delta + 1} - 2\Delta + 1)/2$ for* MIS-B.

The performance ratio claimed above is essentially $\Delta/2.414$. This improves upon the long-standing approximation ratio of $\Delta/2$ due to Hochbaum [15], when $\Delta \geq 10$. However, very recently, there has been a flurry of new results for this problem. Berman and Furer [6] have given an algorithm with performance ratio $(\Delta+3)/5+\epsilon$ when Δ is even, and $(\Delta+3.25)/5+\epsilon$ for odd Δ, where $\epsilon > 0$ is a fixed constant. Halldorsson and Radhakrishnan [14] have shown that algorithm $\mathcal{A}_1$, when run on k-clique free graphs, yields an independent set of size at least $2n/(\Delta + k)$. They combine this algorithm with a clique-removal-based scheme to achieve a performance ratio of $\Delta/6(1 + o(1))$.

In conclusion, note that Khanna, Motwani and Vishwanathan [19] have recently shown that a semidefinite programming technique can be used to obtain a $(\Delta \log\log \Delta)/(\log \Delta)$-approximation algorithm for this problem.

Acknowledgments. Many thanks to Phokion Kolaitis for his helpful comments and suggestions and to Giorgio Ausiello and Pierluigi Crescenzi for guiding us through the intricacies of approximation-preserving reductions and the available literature on them. Thanks also to the anonymous referees for their detailed comments and corrections on a previous draft.

REFERENCES

[1] P. ALIMONTI, *New local search approximation techniques for maximum generalized satisfiability problems*, in Proc. 2nd Italian Conference on Algorithms and Complexity, Springer-Verlag, Berlin, 1994, pp. 40–53.

[2] S. ARORA, C. LUND, R. MOTWANI, M. SUDAN, AND M. SZEGEDY, *Proof verification and hardness of approximation problems*, in Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science, 1992, pp. 14–23.

[3] G. AUSIELLO, P. CRESCENZI, AND M. PROTASI, *Approximate solution of NP optimization problems*, Theoret. Comput. Sci., 150 (1995), pp. 1–55.

[4] G. AUSIELLO AND M. PROTASI, *Local search, reducibility, and approximability of NP optimization problems*, Inform. Process. Lett., 54 (1995), pp. 73–79.

[5] M. BELLARE, S. GOLDWASSER, C. LUND, AND A. RUSSELL, *Efficient probabilistically checkable proofs and applications to approximation*, in Proc. 25th Annual ACM Symposium on Theory of Computing, 1993, pp. 294–304.

[6] P. BERMAN AND M. FURER, *Approximating maximum independent set in bounded degree graphs*, in Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms, 1993, pp. 365–371.

[7] D. P. BOVET AND P. CRESCENZI, *Introduction to the Theory of Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[8] P. CRESCENZI AND A. PANCONESI, *Completeness in approximation classes*, Inform. and Comput., 93 (1991), pp. 241–262.

[9] P. CRESCENZI AND L. TREVISAN, *On approximation scheme preserving reducibility and its applications*, in Proc. 14th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science 880, Springer-Verlag, New York, 1994, pp. 330–341.

[10] R. FAGIN, *Generalized first-order spectra and polynomial-time recognizable sets*, in Complexity of Computer Computations, Richard Karp, ed., AMS, Providence, RI, 1974.

[11] U. FEIGE, S. GOLDWASSER, L. LOVÁSZ, S. SAFRA, AND M. SZEGEDY, *Approximating clique is almost NP-complete*, in Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science, 1991, pp. 2–12.

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

[13] M. X. Goemans and D. P. Williamson, .878-*approximation algorithms for MAX CUT and MAX* 2*SAT*, in Proc. 26th ACM Symposium on Theory of Computing, 1994, pp. 422–431.

[14] M. M. Halldorsson and J. Radhakrishnan, *Improved approximations of independent sets in bounded-degree graphs*, Nordic J. Comput., 1 (1994), pp. 475–492.

[15] D. S. Hochbaum, *Efficient bounds for the stable set, vertex cover, and set packing problems*, Disc. Appl. Math., 6 (1982), pp. 243–254.

[16] V. Kann, *On the Approximability of NP-complete Optimization Problems*, Ph.D. Thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden, 1992.

[17] D. Karger, R. Motwani, and G. D. S. Ramkumar, *On approximating the longest path in a graph*, in Proc. 3rd Workshop on Algorithms and Data Structures, Springer-Verlag, Berlin, 1993, pp. 421–432.

[18] S. Khanna, R. Motwani, M. Sudan, and U. V. Vazirani, *On syntactic versus computational views of approximability*, in Proc. 35th Annual IEEE Symposium on Foundations of Computer Science, 1994, pp. 819–830.

[19] S. Khanna, R. Motwani, and S. Vishwanathan, *Approximating MAX SNP problems via semi-definite programming*, in preparation, 1996.

[20] P. G. Kolaitis and M. N. Thakur, *Approximation properties of NP minimization classes*, J. Comput. System Sci., 50 (1995), pp. 391–411.

[21] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. Assoc. Comput. Mach., 41 (1994), pp. 960–981.

[22] A. Panconesi and D. Ranjan, *Quantifiers and approximation*, Theoret. Comput. Sci., 107 (1993), pp. 145–163.

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

[24] C. H. Papadimitriou and M. Yannakakis, *The traveling salesman problem with distances one and two*, Math. Oper. Res., 18 (1993), pp. 1–11.

[25] M. Yannakakis, *The analysis of local search problems and their heuristics*, in Proc. 7th Annual Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, Berlin, 1990, pp. 298–311.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 192–209

MAXIMUM k-CHAINS IN PLANAR POINT SETS: COMBINATORIAL STRUCTURE AND ALGORITHMS*

STEFAN FELSNER† AND LORENZ WERNISCH†

Abstract. A chain of a set P of n points in the plane is a chain of the dominance order on P. A k-chain is a subset C of P that can be covered by k chains. A k-chain C is a *maximum k-chain* if no other k-chain contains more elements than C. This paper deals with the problem of finding a maximum k-chain of P in the cardinality and in the weighted case.

Using the skeleton $S(P)$ of a point set P introduced by Viennot we describe a fairly simple algorithm that computes maximum k-chains in time $O(kn \log n)$ and linear space. The basic idea is that the canonical chain partition of a maximum $(k-1)$-chain in the skeleton $S(P)$ provides k regions in the plane such that a maximum k-chain for P can be obtained as the union of a maximal chain from each of these regions.

By the symmetry between chains and antichains in the dominance order we may use the algorithm for maximum k-chains to compute maximum k-antichains for planar points in time $O(kn \log n)$. However, for large k one can do better. We describe an algorithm computing maximum k-antichains (and, by symmetry, k-chains) in time $O((n^2/k) \log n)$ and linear space. Consequently, a maximum k-chain can be computed in time $O(n^{3/2} \log n)$ for arbitrary k.

The background for the algorithms is a geometric approach to the Greene–Kleitman theory for permutations. We include a skeleton-based exposition of this theory and give some hints on connections with the theory of Young tableaux.

The concept of the skeleton of a planar point set is extended to the case of a weighted point set. This extension allows to compute maximum weighted k-chains with an algorithm that is similar to the algorithm for the cardinality case. The time and space requirements of the algorithm for weighted k-chains are $O(2^k n \log(2^k n))$ and $O(2^k n)$, respectively.

Key words. algorithms, antichains, chains, orders, point sets, skeletons, Young tableaux

AMS subject classifications. 68Q25, 65Y25, 06A07, 05C85

PII. S0097539794266171

1. Introduction.

The *dominance order* on points in the plane is given by the relations $p \leq q$ if $p_x \leq q_x$ and $p_y \leq q_y$. Here and throughout the paper p_x and p_y denote the x- and y-coordinates of a point p. The symbol P will always be an n-element set of points in the plane together with the dominance relation. A subset C of P is a *chain* if any two members p, q of C are comparable, i.e., either $p \leq q$ or $q \leq p$. On the other hand, a set $A \subseteq P$ with no two different points comparable is an *antichain*. If a subset C of P can be covered by k chains it is called a *k-chain*. If C is a k-chain but not a $(k-1)$-chain we call C a *strict k-chain*. A k-chain C is *maximum* if no other k-chain contains more elements than C. This paper deals with the problem of finding such k-chains in P. Note that the "greedy method" that repeatedly removes maximum chains may fail in computing a maximum k-chain even for $k = 2$ (see, e.g., the point set of Figure 1).

A permutation $\sigma: \{1, \ldots, n\} \to \{1, \ldots, n\}$ may be represented by points $(i, \sigma(i))$ in the plane. Chains and antichains of a point set correspond to increasing and

*Received by the editors April 15, 1994; accepted for publication (in revised form) December 5, 1996; published electronically JUne 15, 1998. A preliminary version of this article appeared in *Proc. 25th Ann. ACM Symp. on the Theory of Computing*, 1993, pp. 146–153.

http://www.siam.org/journals/sicomp/28-1/26617.html

†Freie Universität Berlin, Fachbereich Mathematik, Institut für Informatik, Takustraße 9, 14195 Berlin, Germany (felsner@inf.fu-berlin.de, wernisch@inf.fu-berlin.de). The work of the first author was supported by the Deutsche Forschungsgemeinschaft under grant FE 340/2-1. The work of the second author was supported by a grant from the German-Israeli Binational Science Foundation and by the ESPRIT Basic Research Action Program of the EC under project ALCOM II.

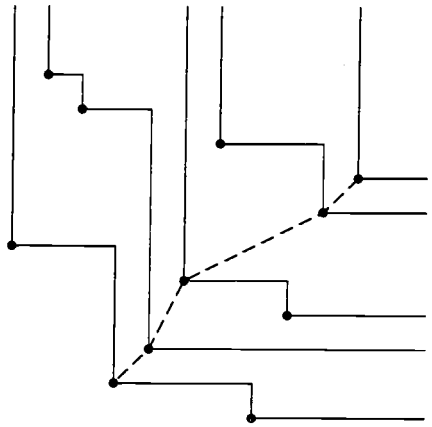


FIG. 1. *The canonical antichain partition and a maximum chain.*

decreasing subsequences of a permutation. Hence, finding maximum k-chains amounts to computing maximum k increasing subsequences. Fredman [3] shows that finding a maximum increasing subsequence requires $\Omega(n \log n)$ comparisons. Of course, this also gives a lower bound for k-chains, $k \geq 2$. On the other hand, an algorithm to compute a longest increasing subsequence in a permutation in time $O(n \log n)$ pertains to mathematicians folklore. A careful treatment of the algorithm can be found in [3]; older sources are, e.g., [1] or [8].

Interest in k-chains of orders goes back to Greene [6] and Greene and Kleitman [7] who discovered a rich duality between maximum k-chains and maximum ℓ-antichains. From this theory we quote a theorem relating maximum k-chains to maximum ℓ-antichains.

THEOREM 1.1. *For an order P with n elements there exists a partition α of n such that the Ferrers diagram F_α of α has the following properties.*

(1) *The number of squares in the k longest rows of F_α equals the size of a maximum k-chain for $1 \leq k \leq n$.*

(2) *The number of squares in the ℓ longest columns of F_α equals the size of a maximum ℓ-antichain for $1 \leq \ell \leq n$.*

The history of algorithms for maximum k-chains seems to start in some of the many alternative proofs for the Greene–Kleitman theorems; we mention two of these approaches. In [13] Viennot deals with the case of permutations or point sets, respectively, and indicates how to find k-chains in time $O((n^2/k) \log n)$. For general orders Frank [2] uses network flows which result in algorithms to compute maximum k-chains in an arbitrary order in time $O(n^3)$. Gavril [4] uses a network designed specifically for k-chain computations and improves the time bound to $O(kn^2)$. Gavril's approach was adapted to handle the weighted case within the same complexity by Sarrafzadeh and Lou [12]. For the case of planar point sets Lou and Sarrafzadeh [9] and Lou, Sarrafzadeh, and Lee [10] propose algorithms to compute 2- and 3-chains in optimal time $O(n \log n)$. They are motivated to consider k-chains in planar point sets by problems in VLSI design, e.g., multilayered via minimization for two-sided channels. Maximum k-chains also turn out to be useful in computational geometry, e.g., for counting points in triangles (see [11]).

We describe a fairly simple method to find maximum k-chains for arbitrary k in time $O(kn \log n)$ and linear space. Our approach is based on the useful concept of the *skeleton* of P introduced by Viennot [14] (see also [13]). We use a maximum $(k-1)$-chain in the skeleton to partition the plane into k regions. Taking a maximum chain from each of the regions already yields a maximum k-chain. Our method leads to a kind of complementary algorithm to the $O((n^2/k) \log n)$ algorithm of Viennot.

In section 2 the notion of skeletons is introduced. We give an algorithm to compute them and show that a point set P is determined by the skeleton $S(P)$ and two additional chains of *marginal points*. The notion of skeletons leads to a geometric interpretation of the well-known bijective correspondence between permutations and pairs of Young tableaux (the Robinson–Schensted correspondence, see, e.g., [8]). The section ends with a brief exposition of this connection.

Section 3 starts with the development of the combinatorial background for the algorithm. The algorithm is described in fairly detailed pseudocode and its correctness is proved. As a by-product we provide a direct geometric proof for part (1) of Theorem 1.1 for permutations. Section 4 is devoted to a complete presentation of Viennot's $O((n^2/k) \log n)$ algorithm for k-antichains. A by-product of the analysis is the direct geometric proof for part (2) of Theorem 1.1 for permutations. The constructions from sections 3 and 4 both imply a result of Greene [5] stating that the shape of the Ferrers diagram F_α in Theorem 1.1 is just the shape of the Young tableaux corresponding to the permutation.

In section 5 we extend the concept of skeletons to the case where a real weight $w(p)$ is associated with each point p in P. The algorithm of section 3 is extended to work with weighted planar point sets and weighted skeletons. This yields a maximum weighted k-chain in time $O(2^k n \log(2^k n))$ and space $O(2^k n)$. Of course, this makes sense only for small values of k. But note that even for constant k no better algorithm than that of Sarrafzadeh and Lou [12] with running time $O(kn^2)$ was known. Unfortunately, it is not obvious how to extend the algorithm of section 4 computing a maximum k-antichain in a similar way to the weighted case.

2. Skeleton and Young tableaux. Let P be our planar point set with n elements. We will always assume that the points of P are in *general position*, i.e., no two points have the same x- or y-coordinate. Arguments are simpler if we assume this generality. In the case of duplicate coordinates we can perturb the points such that they are in general position without changing the comparability relations. Simply change the values of the x-coordinate of points with the same x-coordinate—by definition they are comparable—by a small amount such that they get increasing x-coordinates with increasing y-coordinates. Points with the same y-coordinate are perturbed analogously. Such perturbations can be made in a single sweep, i.e., in time $O(n \log n)$. As is easily seen, the complexity of all algorithms in this paper remains as claimed even if we make such a sweep whenever we start working with a new set of points.

The *height* of a point $p \in P$ is the size of a longest chain with p as maximal element. Of course, two points of the same height cannot be comparable. Hence, collecting points with the same height in the same set yield a partition $\mathcal{A}$ of P into antichains, the *canonical antichain partition*. Observe that this partition is also obtained by a repeated removal of the set of minimal elements. By definition, the number of antichains in a canonical partition is the *height of* P, i.e., the size of a largest chain in P (see Figure 1). Since, obviously, a chain and an antichain can have at most one point in common, there can be no partition into fewer antichains than there are in $\mathcal{A}$; i.e., it is a *minimal antichain partition*.

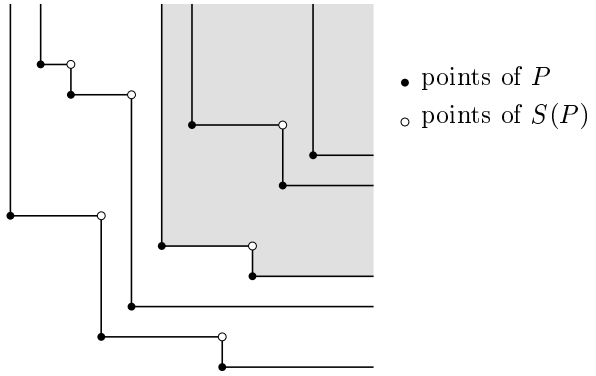


FIG. 2. *Point set P, its skeleton, and the shadow of the third layer.*

Following Viennot, we define the *left shadow* of point p as the set of all points (u, v) *dominating* p, i.e., with $u \geq p_x$ and $v \geq p_y$. For a set E of points, the *shadow of E* is the union of the left shadows of the points of E, i.e., the set of all points q dominating at least one point of E. The *right shadow* of p is the set of all points (u, v) with $u \leq p_x$ and $v \geq p_y$. The term shadow suggests some light coming from the left below or from the right below in the case of a right shadow (of course, this should not be taken literally, since the "shadows" have a form that is scarcely realizable physically). The *right down shadow* of p is the set of all points (u, v) with $u \leq p_x$ and $v \leq p_y$. The right and right down shadows of a set E are again defined as the union of the corresponding shadows of the points of E.

The *left jump line* or simply *jump line*, $L_{\nearrow}(E)$ or $L(E)$, of a point set E is the topological boundary of the left shadow of E. The *right jump line* $L_{\nwarrow}(E)$ and the *right down jump line* $L_{\swarrow}(E)$ are the topological boundaries of the right and the right down shadow of E. Let the unbounded half line of the jump line extending upward be the *top outgoing line*, and let the unbounded half line extending to the right be the *right outgoing line*. Additionally, we use the term *left outgoing line* when dealing with right or right down jump lines. It is easily seen that the jump line $L(A)$ of an antichain A is a downward staircase with the points of A in its lower corners. Collect the points in the upper corners of $L(A)$ in the set $S_{\nearrow}(A) = S(A)$; this is the set of *skeleton points* or briefly the *skeleton* of the antichain A. Formally, if $(x_1, y_1), \ldots, (x_k, y_k)$ are the points of A ordered by increasing x-coordinate then $S(A)$ consists of the points $(x_2, y_1), \ldots, (x_k, y_{k-1})$. Hence, $L(A)$ has exactly $|A|-1$ skeleton points (see Figure 2).

The minimal elements of a point set P form an antichain A such that the rest of $P - A$ lies completely in the shadow of A. Hence, by removing A and treating $P - A$ in the same way, we recursively obtain the canonical antichain partition $A = A_0, \ldots, A_{\lambda-1}$ with nonintersecting jump lines $L(A_i)$, $0 \leq i < \lambda$, which will be called the *layers $L_i(P)$ of P*. The *skeleton* or *left skeleton of P*, denoted by $S(P)$ or $S_{\nearrow}(P)$, is then defined as the union of the skeletons $S(A_i)$, $0 \leq i < \lambda$. Since, as noted above, the ith layer $L_i(P)$ has $|A_i| - 1$ skeleton points, the size of $S(P)$ is $|P| - \lambda$. A picture of a point set P, its skeleton $S(P)$, its antichain layer partition, and the shadow of antichain 2 can be found in Figure 2. Let us state an easy but quite useful observation.

LEMMA 2.1. *Suppose a point set P is partitioned into k antichains A_i in such a way that the jump lines $L(A_i)$ are pairwise disjoint. Then $A_1, \ldots, A_k$ is the canonical antichain partition of P.*

```
SKELETON(P)
insert dummy d_u in L at height +∞; link(d_u) ← nil;
k ← 0; S ← ∅;
for each p ∈ P from left to right do
    insert a new marker m' in L with m'_y ← p_y;
    point(m') ← p;
    m ← next marker below p on L;
    link(p) ← point(m);
    m ← next marker above p on L;
    if m = d_u then
        A_k ← {p}; antichain(m) ← A_k; k ← k + 1;
    else
        add p to antichain(m);
        add skeleton point (L_x, m_y) to S;
        remove m from L;
v ← point(m), m uppermost marker on L; C ← {v};
while link(v) ≠ nil do
    v ← link(v); C ← C ∪ {v};
return S, C, A_0, ..., A_{k−1};
```

FIG. 3. *Algorithm* SKELETON.

Proof. Suppose the A_i are ordered by increasing x-coordinates of the top outgoing lines of the $L(A_i)$. Then $P - A_0$ is in the shadow of A_0. Hence, by the definition of the shadow, each point of $P - A_0$ dominates at least one of A_0 and no point of A_0 dominates any other point in P; i.e., A_0 are just the minimal elements of P. Repeat the procedure on $P - A_0$. □

Let us describe a simple algorithm to compute the skeleton, a maximum chain, and the canonical antichain partition of a point set P (see Figure 3). Essentially, this is the well-known algorithm for longest increasing sequences of permutations (for a geometrically inspired version see [10]). A sweep line L going from left to right halts at every point of P. It contains an ordered set of markers m. A marker m on L has a y-coordinate m_y, and m is said to be above a point p if $m_y > p_y$.

Suppose L halts at some point p and the layers have been constructed for all points to the left of L. Find the next layer with right outgoing line above p. If there is no such layer (i.e., the marker found equals the dummy point), open a new one with p as its (yet) sole point. If there is one, add p to this layer and generate a new skeleton point. It is easily seen that the jump lines thus constructed cannot intersect and hence are the layers of a canonical antichain partition, by Lemma 2.1. Finally, a maximum chain is obtained by extracting a point from each of the antichains along a chain of properly established links. With L implemented as a dynamic binary tree we have Theorem 2.2.

THEOREM 2.2. *Algorithm* SKELETON *computes the skeleton, a maximum chain, and the canonical antichain partition of a point set P of size n in time $O(n \log(n))$ and linear space.* □

For the definition of the *right skeleton* $S_{\nwarrow}(P)$ use the right shadow and the right jump lines. And for the *right down skeleton* $S_{\swarrow}(P)$ use right down shadow and right down jump lines. Of course, with $S_{\nwarrow}(P)$ we obtain the canonical chain

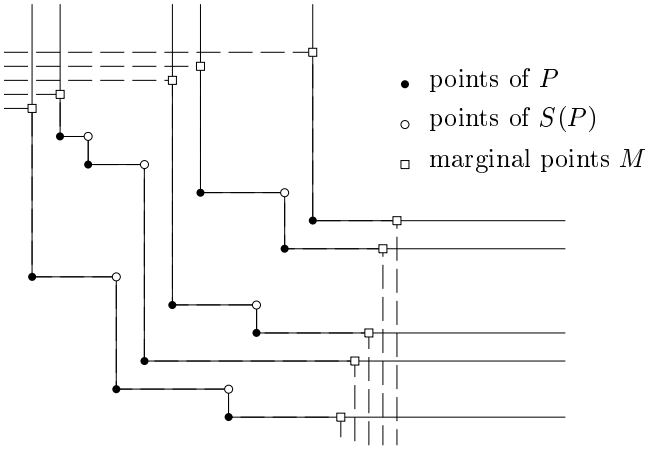


FIG. 4. *P is the right down skeleton of $S(P) \cup M(P)$.*

partition instead of the antichain partition. By symmetry, a lemma corresponding to Lemma 2.1 but dealing with chains instead of antichains is again true. With $S_{\swarrow}(P)$ we again obtain an antichain partition. A layer of this partition contains all the points with the same dual height (the dual height of p is the length of a maximum chain that has p as its minimal element).

It is convenient to conceive the construction of the skeleton as an operator on finite point sets consisting of points in general position, since the points of the skeleton $S(P)$ again have pairwise different x- and y-coordinates. Thus, we may apply operators $S_{\nearrow}$ and $S_{\nwarrow}$ to $S(P)$. As usual, the k-fold iteration of an operator O will be denoted by O^k; O^0 means identity. $S^k(P)$ will be called the *kth skeleton* of P. An interesting algebraic property of $S_{\nearrow}$ and $S_{\nwarrow}$, that they are commutative, is shown in [15].

One of the properties that seems to lie behind the usefulness of skeletons is the fact that it is possible to reconstruct P from $S(P)$ with a small amount of additional information. Let $x_{\max}$ be the maximal x-coordinate of points in P, and let $y_{\max}$ be defined analogously. Then the *right marginal points* $M_R(P)$ of P are the points $(x_{\max}+1, y_1), \ldots, (x_{\max}+\lambda, y_\lambda)$, where λ is the number of layers of P and $y_1, \ldots, y_\lambda$ are the y-coordinates of the right outgoing lines of the layers ordered increasingly (see Figure 4). Assuming $x_1, \ldots, x_\lambda$ are the x-coordinates of the top outgoing lines of the layers in increasing order, the *top marginal points* $M_T(P)$ of P are $(x_1, y_{\max}+1)$, $\ldots, (x_\lambda, y_{\max}+\lambda)$ (see Figure 4). Note that each of $M_R(P)$ and $M_T(P)$ is a chain of length height of P. With $M(P)$ we denote the collection of marginal points of P, i.e., $M(P) = M_R(P) \cup M_T(P)$.

THEOREM 2.3. *A point set P is the right down skeleton of the skeleton $S(P)$ together with the marginal points of P, i.e., $P = S_{\swarrow}(S(P) \cup M(P))$.*

Proof. The jump line $L(A)$ of an antichain A nearly coincides with the right down jump line $L_1 = L_{\swarrow}(S(A \cup \{s,t\}))$ of the skeleton of A with an arbitrary point s somewhere on the top outgoing line of $L(A)$ and some point t on the right outgoing line. More precisely, between s and t the two jump lines are equal. As the marginal points on the top and right outgoing lines are chosen so that they form chains, we may bend the original jump lines of P at the marginal points, and the bent lines remain nonintersecting. Each bent line L is the right down jump line of the points

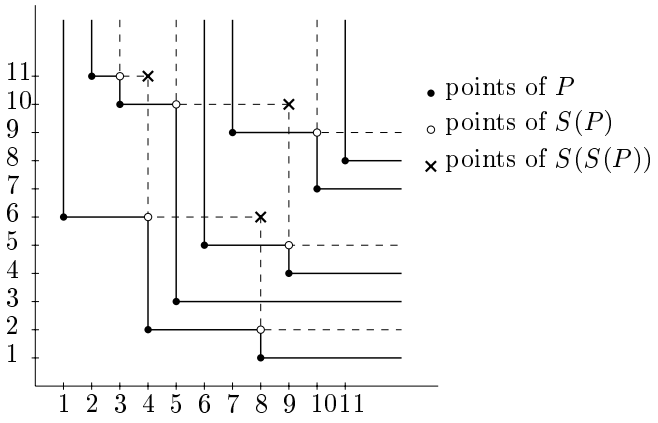


FIG. 5. *A point set P with $\lambda_0(P) = 5$ and $\lambda_1(P) = 3$.*

of $S(P) \cup M(P)$ contained in it. Moreover, the points of $S(P) \cup M(P)$ contained in each of these lines form an antichain and each point is on one of these lines. Hence, the right down version of Lemma 2.1 applies and we are done. □

A *partition* of an integer n is a sequence of integers $\lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{\mu-1} > 0$ such that $n = \lambda_0 + \cdots \lambda_{\mu-1}$. Such a partition may be represented graphically by Ferrers diagram also called the *Young shape*. This is a shape as that of the two figures in Figure 6, which consist of μ rows of rectangles or *cells* with λ_i cells in row i, when rows are taken from bottom to top (also called "French notation"). If numbers[1] are put in these cells in increasing order from left to right and from bottom to top we obtain a *Young tableau*.

Since we will often refer to the number of layers of P, let us adopt the following notation. Let $\mu(P)$ be minimal with $S^{\mu(P)}(P) = \emptyset$. Then $\lambda_i(P)$, $0 \leq i < \mu(P)$, denotes the number of layers of $S^i(P)$ (see Figure 5). It is convenient to assume $\lambda_i(P) = 0$ for $i \geq \mu(P)$.

LEMMA 2.4. *Let P be a point set; then $\lambda_0(P) \geq \lambda_1(P) \geq \cdots \geq \lambda_{\mu(P)-1} > 0$, and $|S^k(P)| = \sum_{k \leq i < \mu} \lambda_i(P)$, where $\mu = \mu(P)$. In particular, $\lambda_0(P), \lambda_1(P), \ldots, \lambda_{\mu(P)-1} > 0$ is a partition of n.*

Proof. By Theorem 2.3, the number of antichains in a minimal antichain partition of $S(P) \cup M(P)$ is the same as $\lambda_0(P)$, the size of the canonical antichain partition of P. Hence, $\lambda_1(P)$, the size of a minimal antichain partition of $S(P)$, is at most $\lambda_0(P)$. The same argument shows the other inequalities. The sum over the $\lambda_i(P)$ is computed easily by using $|S(P)| = |P| - \lambda_0(P)$ and induction. □

Let P be a planar set of n points. We associate two tableaux $\mathbf{P}(P)$ and $\mathbf{Q}(P)$ (the $\mathbf{P}$- and $\mathbf{Q}$-*symbol of* P) with P in the following way. The kth rows of $\mathbf{P}(P)$, $k \geq 0$, are the y-coordinates of the right outgoing lines of $S^k(P)$ in increasing order. The kth row of $\mathbf{Q}(P)$, $k \geq 0$, are the x-coordinates of the top outgoing lines of $S^k(P)$ in increasing order. Compare the 5 and 3 outgoing lines of the first two layers of Figure 5 with the first two rows of the Young tableaux in Figure 6. According to Lemma 2.4, $\mathbf{P}(P)$ and $\mathbf{Q}(P)$ have $\lambda_i(P)$ cells in their ith row from below and $|P|$ cells altogether. Hence, the shape of the tableaux $\mathbf{P}(P)$ and $\mathbf{Q}(P)$ is a Young shape. We

[1] In the classical theory these are the numbers $1, \ldots, n$ which give a correspondence between pairs of Young tableaux and permutations. In the present context it is convenient to allow real entries.

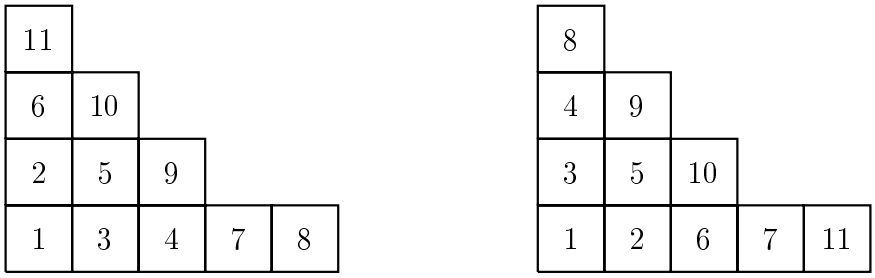


FIG. 6. *The* **P***- and* **Q***-symbol of point set* P *of Figure* 5.

denote the number of cells in the ith column (from the left) with $\lambda_i^*(P)$. Obviously, $\lambda_0^*(P) \geq \lambda_1^*(P) \geq \cdots \geq \lambda_{\ell-1}^*(P)$ and $\sum_{0 \leq i < \ell} \lambda_i^*(P) = |P|$, where $\ell = \lambda_0(P)$. The $\lambda_i^*(P)$ is the *conjugate partition* of the $\lambda_i(P)$ for the integer $|P|$.

Our first observation about the **P**- and **Q**-symbol concerns the *inverse* P^{-1} *of* P, which is the point set that is obtained from P by the transposition $(x, y) \to (y, x)$, i.e., by reflection on the diagonal line $x = y$. Obviously, the corresponding **P**- and **Q**-symbols are simply interchanged.

THEOREM 2.5. *For the inverse* P^{-1} *of a point set* P, $\mathbf{P}(P^{-1}) = \mathbf{Q}(P)$ *and* $\mathbf{Q}(P^{-1}) = \mathbf{P}(P)$. □

THEOREM 2.6 (Robinson–Schensted). *The tableaux* $\mathbf{P}(P)$ *and* $\mathbf{Q}(P)$ *of a point set* P *are Young tableaux. Moreover, for any two Young tableaux* $\mathbf{P}$ *and* $\mathbf{Q}$ *with the same shape there exists a point set* P *with* $\mathbf{P} = \mathbf{P}(P)$ *and* $\mathbf{Q} = \mathbf{Q}(P)$*; i.e., there is a bijection between point sets and pairs of Young tableaux with the same shape.*

The reader interested in the proof of this theorem and in a more comprehensive treatment of geometric approaches to the theory of the Young tableaux is referred to Viennot [14] and Wernisch [15].

3. Maximum *k*-chains. Suppose a subset C_S of the skeleton $S(P)$ of a planar point set P is given and let $C_1, \ldots, C_k$ be the canonical chain partition of C_S. The existence of such a chain partition implies that C_S is a k-chain (the converse is false; a k-chain can have a partition into fewer chains). We define the *ith region of* C_S, for $2 \leq i \leq k$, to be the intersection of the right shadow of C_{i-1} with the complement of the right shadow of C_i, i.e., the region between the jump lines of C_{i-1} and C_i, containing the jump line of C_{i-1} but excluding that of C_i (see Figure 7). The *first region* is the complement of the right shadow of C_1, and the $(k+1)$*st region* is the right shadow of C_k. These $k+1$ regions partition the whole plane.

A description of the main steps of an algorithm computing k-chains can be given with this concept of the region (a more detailed description can be found in Figure 9). The reader may want to visualize the following steps on Figure 7.

1. Compute the skeleton $S(P)$ of a planar point set P.
2. Compute recursively a $(k-1)$-chain C_{k-1} of $S(P)$.
3. For all regions R defined by C_{k-1}, extract a maximum chain from P intersected with R. The union of all chains is a maximum k-chain for P.

To demonstrate the correctness of the approach we need another definition. An antichain A of P and its jump line are said to *cross region* R *well* if $R \cap L(A) = R \cap L(A \cap R)$. That is, A crosses R well exactly if the jump line $L(A)$ enters R vertically and leaves R horizontally (see Figure 8).

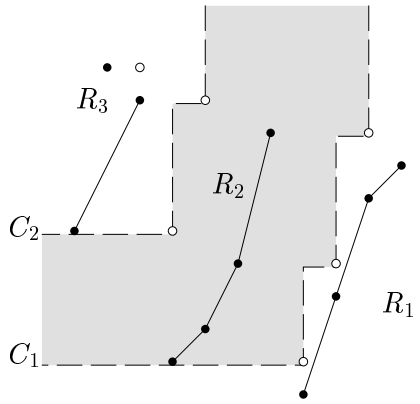


FIG. 7. *Three regions defined by a maximum* 2*-chain of the skeleton each containing one chain.*

The next lemma expresses the key property of antichains that cross well that makes them useful for our purposes.

LEMMA 3.1. *Let R be a region of $C_S \subseteq S(P)$ and let $A_1, \dots, A_\lambda$ be the canonical antichain partition of P. If I is the set of all indices i such that A_i is crossing R well, then the collection $\{A_i \cap R \mid i \in I\}$ is the canonical antichain partition of the underlying point set $\bigcup_{i\in I} A_i \cap R$.*

Proof. Let R be the ℓth region and let $p = L(A_i) \cap L_\diagdown(C_\ell)$ and $p' = L(A_j) \cap L_\diagdown(C_\ell)$ with $i < j$. Note that any two points on $L_\diagdown(C_\ell)$ are comparable. Since $L(A_j)$ is in the shadow of $L(A_i)$ we cannot have p' dominated by p; hence p' dominates p. Since A_i is crossing R well, the line segment of $L(A_i \cap R)$ that ends in p is vertical. The same holds for $L(A_j \cap R)$ and p'. As the two jump lines are disjoint we obtain $p_x < p'_x$. Therefore, we can extend $L(A_i \cap R)$ and $L(A_j \cap R)$ by vertical half lines without introducing an intersection.

A similar argument shows that the y-coordinates of $q = L(A_i) \cap L_\diagdown(C_{\ell-1})$ and $q' = L(A_j) \cap L_\diagdown(C_{\ell-1})$ are related by $q_y < q'_y$. Hence, the corresponding right half lines do not cross. Altogether the jump lines $L(A_i \cup R)$ are pairwise disjoint and, by Lemma 2.1, $\{A_i \cap R \mid i \in I\}$ is the canonical antichain partition of the underlying set. □

LEMMA 3.2. *Let $C_S \subseteq S(P)$ and let A be an antichain of the canonical partition of P. If m is the number of skeleton points on A that are in C_S, then the number of regions of C_S crossed well by A is at least $m + 1$.*

Proof. Let c be a point of C_S on the jump line L of A. Let p and q be the points of A to the left and below c that define it (see Figure 8). Then it is obvious that L leaves the region containing p horizontally and enters that of q vertically. Of course, the top outgoing line of L is vertical and the right outgoing line is horizontal.

Note that if L leaves one region vertically the region of the next point of A to the right is entered vertically too. Now consider the $m+1$ sections of L from left to right before, between, and after its m points in C_S (possibly m equals 0). Since in each such section A enters its first region vertically and leaves its last region horizontally, there must be some region crossed well by A in between. □

LEMMA 3.3. *Let C_S be a $(k-1)$-chain in the skeleton $S(P)$ of P and let λ be the height of P. Taking a maximum chain of $P \cap R$ in each region R of C_S yields a k-chain of P of size at least $|C_S| + \lambda$.*

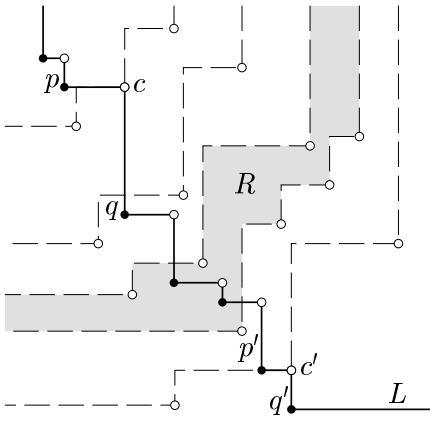


FIG. 8. *L crosses some region R well in the section between c and c'.*

Proof. Let $A_1, \ldots, A_\lambda$ be the canonical antichain partition of P. Each point of C_S is a skeleton point of exactly one antichain A_i. If m_i is the number of skeleton points of A_i in C_S, for $1 \leq i \leq \lambda$, then, according to Lemma 3.2, the antichains A_i cross the regions well in altogether at least $\sum_{1 \leq i \leq \lambda}(m_i + 1) = |C_S| + \lambda$ sections. On the other hand, by Lemma 3.1, each such section crossing some region well contributes one more point to the maximum chain of that region. □

Note that Lemma 3.3 does not require the $(k-1)$-chain C_S to be maximum. We now show a kind of reverse to Lemma 3.3.

LEMMA 3.4. *Let C be a k-chain in P. There exists a $(k-1)$-chain in the skeleton $S(P)$ with size at least $|C| - \lambda$, where λ is the height of P.*

Proof. By Theorem 2.3, P is the right down skeleton of $S(P) \cup M(P)$. The chains $M_R(P)$ and $M_T(P)$ of marginal points both have size λ. Hence, we may reason similarly as in the proof of Lemma 3.3 after reflection of all points on the diagonal $x = -y$, i.e., after the transformation $T(x, y) = (-x, -y)$ of the plane. The right down skeleton P is thus transformed to a skeleton $T(P)$ of $T(S(P) \cup M(P))$. The image $T(C)$ of the k-chain $C \subseteq P$ is a k-chain in the skeleton $T(P)$ defining $k + 1$ regions. Observe that $T(M_R)$ lies in the first region and $T(M_T)$ lies in the $(k + 1)$st region of $T(C)$ and that both are maximum chains of length λ. We apply Lemma 3.3 and obtain a $(k + 2)$-chain C'_S in $T(S(P) \cup M(P))$ with $|C'_S| \geq |C| + \lambda$. According to the above observation, we may further assume that $T(M_R)$ and $T(M_T)$ are in C_S. When these two chains are removed from C'_S we obtain a $(k-1)$-chain $C_S \subseteq T(S(P))$ of size at least $|C| - \lambda$ and $T(C_S) \subseteq S(P)$ is the $(k-1)$-chain searched for. □

We denote the k-chain in P obtained from a $(k-1)$-chain C_S in $S(P)$ according to Lemma 3.3 by $\sigma_k(C_S)$.

THEOREM 3.5. *Let C_1 be a maximum chain of $S^{k-1}(P)$ and $C_i = \sigma_i(C_{i-1})$ for $2 \leq i \leq k$; then C_k is a maximum k-chain in P.*

Proof. By induction, suppose that C_{k-1} is a maximum $(k-1)$-chain in $S(P)$ and let λ be the height of P. If there were a k-chain $C \subseteq P$ with more points than $\sigma_k(C_{k-1})$ then C would have more than $|C_{k-1}| + \lambda$ points, according to Lemma 3.3. Hence, by Lemma 3.4, there would be a $(k-1)$-chain of size larger than $|C_{k-1}|$ in $S(P)$, a contradiction. □

Note that the proof of the above theorem also shows that the number of additional

```
MAXMULTICHAIN(P, k)
C_S ← ∅; C ← ∅;
if k ≥ 2 then
    S ← skeleton(P);
    M ← marginal points(P);
    dispose P;
    C_S ← MAXMULTICHAIN(S, k − 1);
    P ← right down skeleton(S ∪ M);
R_1, ..., R_l ← regions(C_S);
partition P into P_i ← P ∩ R_i, 1 ≤ i ≤ l;
for i ← 1 to l do
    C ← C ∪ maximum chain(P_i);
return C;
```

FIG. 9. *Algorithm* MAXMULTICHAIN.

points in each application of σ_ℓ to a maximum $(\ell-1)$-chain of $S^{k-\ell+1}(P)$ is equal to the height of $S^{k-\ell}(P)$ for $2 \le \ell \le k$. Hence, we obtain the following corollary that is part (1) of Greene's theorem for permutations (see Theorem 1.1).

COROLLARY 3.6. *A maximum k-chain of a point set P has size $\sum_{0\le i<k} \lambda_i(P)$ where $\lambda_i(P)$ is the height of $S^i(P)$.* □

We are now prepared to provide an algorithm MAXMULTICHAIN (see Figure 9) that, given a point set P and some k, computes a maximum k-chain of P. Some remarks about this algorithm are in order. To dispose P means to release any memory space holding the points of P, which is necessary to keep the space requirement small. Recall that a maximum $(k-1)$-chain may consist of fewer than $k-1$ chains. This happens if C_S equals S and can be partitioned into less than $k-1$ chains. Hence, there may be fewer than k regions of C_S. If C_S is empty (e.g., when $k=1$), we assume that R_1 is the whole plane.

The partitioning of P according to the regions R_i of C_S can be done with a single sweep from left to right halting at every point of P. The sweep line L contains its intersection with all the right layers of C_S and is initialized to the y-coordinates of the left outgoing lines of these layers. Now a point $p \in P$ is easily assigned to its region. If the skeleton point immediately above p is in C_S, then the height of the intersection point of the corresponding right layer with the sweep line has to be adapted.

THEOREM 3.7. *Algorithm* MAXMULTICHAIN(P,k) *computes a maximum k-chain for a point set P of size n in time $O(kn\log n)$ and linear space.*

Proof. According to Theorem 2.2, the skeleton, marginal points, and the canonical chain partition of a point set of size n can be computed in $O(n\log n)$ time. The partitioning of P described above takes the same amount of time. The computation of the maximum chains in each region take, again by Theorem 2.2, time $O(\sum_{i=1}^{l} |P_i|\log(|P_i|))$ which is $O(n\log n)$. Since these estimations hold true in each recursive step, we have an overall time $O(kn\log n)$.

As far as the space requirement is concerned, the main problem is the computation of a new skeleton in each recursive step. But P is disposed and only its skeleton together with the marginal points is retained. The number of marginal points in the ith step equals twice the number $\lambda_i(P)$ of layers of the ith skeleton $S^i(P)$, $0 \le i \le k$. Thus, the amount of space that is needed for k recursions is $O(2\sum_{i=0}^{k} \lambda_i(P) + |S^k(P)|)$

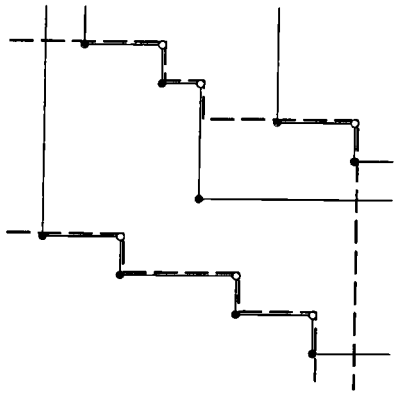


FIG. 10. *The Δ-operator.*

which, by Lemma 2.4, is $O(|P|)$. □

4. Maximum k-antichains. In this section we prove some assertions made by Viennot [13] leading to an algorithm that efficiently computes k-antichains of a point set P.

Let A_S be a k-antichain in the skeleton $S(P)$ and let $A_{S,1}, \ldots, A_{S,k}$ be the canonical antichain partition of A_S. It is easily seen that the intersection of P with the right down jump line $L_{\swarrow}(A_{S,i})$ is an antichain in P. Let us denote the k-antichain $\bigcup_{i=1}^{k} L_{\swarrow}(A_{S,i}) \cap P$ of P by $\Delta(A_S)$ (see Figure 10). On the other hand, given a k-antichain A with canonical partition $A_1, \ldots, A_k$ we define the k-antichain $\delta(A) = \bigcup_{i=1}^{k} L(A_i) \cap S(P)$ of $S(P)$. Recall that a strict k-antichain is one that cannot be covered by less than k antichains.

LEMMA 4.1. *Let P be a planar point set.*

1. *If A_S is a strict k-antichain of $S(P)$, then $\Delta(A_S)$ is a k-antichain of P of size at least $|A_S| + k$.*
2. *If A is a strict k-antichain of P, then $\delta(A)$ is a k-antichain of $S(P)$ of size at least $|A| - k$.*
3. *If A_S is a strict maximum k-antichain, then equality holds in item 1 of this lemma and $\Delta(A_S)$ is a strict maximum k-antichain, too.*

Proof. Let $A_{S,1}, \ldots, A_{S,k}$ be the canonical right down antichain partition of A_S. Each skeleton point $s \in S(P)$ has two defining points $p_L(s), p_D(s) \in P$, one with the same y-coordinate to the left, the other with the same x-coordinate below. If we walk along a right down jump line $L_{\swarrow}(A_{S,i})$ from left to right we find between any two consecutive skeleton points s_1, s_2 of $A_{S,i}$ at least one of the defining points $p_D(s_1)$ or $p_L(s_2)$ on the jump line. Otherwise, the defining point $p_D(s_1)$ would have a y-coordinate smaller than that of s_2, and point $p_L(s_2)$ would have x-coordinate smaller than that of s_1. But this implies that two layers of the canonical layer structure of P intersect, which is impossible (see Figure 11). Since the left and down defining point $p_L(s_L)$ and $p_D(s_R)$ of the leftmost and rightmost skeleton points s_L and s_R of $A_{S,i}$ are always on the right down jump line, $P \cap L_{\swarrow}(A_{S,i})$ contains at least one point more than $A_{S,i}$. Summing over all $A_{S,i}$ we get the first inequality of the lemma.

The second inequality is obtained similarly. One may extend P by the two marginal chains and use the transformation $T(x, y) = (-x, -y)$. By Theorem 2.3,

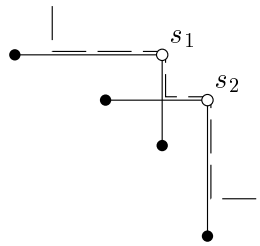


FIG. 11. *Intersecting layers.*

$T(P)$ is the skeleton of $T(S(P) \cup M(P))$. Hence, by the above argument, $|\delta(A)| \geq |\Delta(T(A))| - 2k \geq |T(A)| - k$ since the two chains $T(M_T)$ and $T(M_R)$ contribute at most $2k$ points to the k-antichain $\Delta(T(A))$ in $T(S(P) \cup M(P))$.

Suppose A_S is a strict maximum k-antichain and let $\Delta(A_S)$ be a strict k'-antichain with $k' \leq k$. Then $\delta(\Delta(A_S))$ is a k'-antichain, hence a k-antichain, of size at least $|\Delta(A_S)| - k' \geq |A_S| + k - k'$ in $S(P)$ and $k' = k$ since A_S is maximum. Consequently, if A' is a k-antichain of P, $|\Delta(A_S)| - k \geq |A_S| \geq \delta(A') \geq |A'| - k$, which implies that $\Delta(A_S)$ is a maximum k-antichain. □

THEOREM 4.2. *Let P be a point set and let $k \leq \lambda_0(P)$. There is an ℓ with $0 \leq \ell < \mu(P)$ and $\lambda_{\ell+1}(P) \leq k < \lambda_\ell(P)$. The ℓth skeleton $S^\ell(P)$ contains a strict maximum k-antichain A_ℓ, and $\Delta^\ell(A_\ell)$ is a maximum k-antichain of P of size $|S^{\ell+1}(P)| + k\ell = \sum_{i=0}^{k} \lambda_i^*(P)$.*

Proof. Since, by Lemma 2.4, the $\lambda_i(P)$ are decreasing in i and $\lambda_{\mu(P)}(P) = 0$, there is an ℓ such that the inequalities are satisfied. $S^{\ell+1}(P)$ itself is a strict maximum $\lambda_{\ell+1}(P)$-antichain and $\Delta(S^{\ell+1}(P))$ is a strict maximum $\lambda_{\ell+1}(P)$-antichain of $S^\ell(P)$. The size of $\Delta(S^{\ell+1}(P))$ is $|S^{\ell+1}(P)| + \lambda_{\ell+1}(P)$ by Lemma 4.1. Take a maximum chain C in $S^\ell(P)$; it has size $\lambda_\ell(P)$. Since $\Delta(S^{\ell+1}(P))$ intersects C in at most $\lambda_{\ell+1}(P)$ points and $S^\ell(P)$ has size $|S^{\ell+1}(P)| + \lambda_\ell(P)$, there is no point of $S^\ell(P)$ outside $C \cup \Delta(S^{\ell+1}(P))$ and we may add $k - \lambda_{\ell+1}(P)$ arbitrary points of $S^\ell(P)$ to $\Delta(S^{\ell+1}(P))$ to get a strict maximum k-antichain A_ℓ in $S^\ell(P)$ of size $|S^{\ell+1}(P)| + k$. Now induction and application of Lemma 4.1 shows the theorem. □

An algorithm computing a maximum k-antichain of P for given P and k is now easy to provide. For algorithm MAXMULTIANTICHAIN see Figure 12.

THEOREM 4.3. *Algorithm* MAXMULTIANTICHAIN(P, k) *computes a maximum k-antichain for a point set P of size n in time $O((n^2/k) \log n)$ and linear space.*

Proof. Since the algorithm simply mimics the proof of Theorem 4.2, it certainly computes a maximum k-antichain. The computation of the skeleton, marginal points, and the number of layers takes time $O(n \log n)$. The Δ operator is implemented straightforwardly. For a sweep line going from right to left computing the right down layers of A_S may halt at points of P, too, and check whether they lie on a layer or not. Hence, the time needed for one recursive step is $O(n \log n)$. According to Theorem 4.2, a maximum k-antichain has size $|S^{\ell+1}| + k\ell \leq n$. Thus, the number $\ell + 1$ of recursions is bounded by $n/k + 1$. That the amount of space needed remains linear is seen as in the proof of Theorem 3.7. □

5. Maximum weighted k-chains. Given some weight $w\colon P \to \mathbf{R}$ on the points of a set P, we define the weight of a k-chain as the sum of the weights of its points. A *maximum weighted* k-chain has maximum weight among all weights of k-chains of P. Such maximum weighted k-chains can be found in a similar way as maximum

```
MAXMULTIANTICHAIN(P, k)
λ ← number of layers of (P);
if λ ≤ k then
    return (P, λ);
else
    S ← skeleton(P);
    M ← marginal points(P);
    dispose P;
    (A_S, λ') ← MAXMULTIANTICHAIN(S, k);
    P ← right down skeleton(S ∪ M);
    A ← Δ(A_S);
    if λ' < k then
        add k − λ' arbitrary points of P to A;
    return (A, k);
```

FIG. 12. *Algorithm* MAXMULTIANTICHAIN.

k-chains. Unfortunately, the corresponding algorithm is efficient only if k is small.

In the following assume that the weights w are positive integers. With this assumption the weighted case can be simulated by the unweighted one. Although the algorithms work on the weighted point set itself, the proofs of correctness are based on the following idea. We expand each point $p \in P$ into a tiny chain $C(p)$ of $w(p)$ points, where tiny means that the chain is contained within a tiny box of sidelength ϵ where ϵ is less than the minimum distance in x- or y-coordinates of the points of P (recall that we assume all points to have different x- and y-coordinates). Denote the expanded set of points by P'. Now consider the skeleton $S(P')$ of P'. Let p, q be two points of P with $p_x < q_x$. It is easily seen that if there are any skeleton points in $S(P')$ having their defining points in the tiny chains $C(p)$ and $C(q)$, then all skeleton points with this property again fit into a box B of sidelength ϵ. In this case, locate a *weighted skeleton point* between p and q (i.e., at (q_x, p_y)) and give it a weight equal to the number of skeleton points contained in box B. The resulting set of weighted skeleton points is the *weighted skeleton* $S(P, w)$.

The weighted skeleton $S(P, w)$ can also be obtained without resorting to set P' of multiplied points. We translate the actions of Algorithm SKELETON (see Figure 3 of section 2) that construct the skeleton $S(P')$ of P' with sweep line L' into actions of an Algorithm SKELETON-WEIGHTED (see Figure 13) that construct the corresponding weighted skeleton $S(P, w)$ of P with sweep line L. As a by-product Algorithm SKELETON-WEIGHTED also computes a maximum weighted chain of P.

If the y-coordinates of a set M of markers on L' differ by an amount smaller than ϵ, then they correspond to a *weighted marker* m on L with weight $W(m) = |M|$. The insertion of a point $p \in P$ with weight $w(p)$ in L corresponds to the $w(p)$ insertions of points from $C(p) \subseteq P'$ into L'. Let m be the next marker above p. If $|C(p)|$ is greater than or equal to the number $W(m)$ of markers on L' that correspond to m, then $W(m)$ new skeleton points in $S(P')$ are generated. Hence, we have to generate a new skeleton point in $S(P, w)$ of weight $W(m)$ and remove marker m. If $|C(p)| > W(m)$ there remain $W = |C(p)| - W(m)$ points of $C(p)$ for insertion. Hence, the next marker on L is searched and the procedure is repeated until there is

```
SKELETON-WEIGHTED(P, w)
insert dummy d_u in L with W(d_u) ← +∞ at height +∞;
link(d_u) ← nil;
S ← ∅;
for each p ∈ P from left to right do
    insert a new marker m' in L with m'_y ← p_y;
    W(m') ← w(p);
    point(m') ← p;
    m ← next marker below p on L;
    link(p) ← point(m);
    m ← next marker above p on L;
    W ← w(p);
    while W ≥ W(m) do
        add skeleton point (L_x, m_y) with weight W(m) to S;
        W ← W − W(m);
        remove m from L;
        m ← next marker above p on L;
    if m ≠ d_u and W > 0 then
        add skeleton point (L_x, m_y) with weight W to S;
        W(m) ← W(m) − W;
v ← point(m), m uppermost marker on L; C ← {v};
while link(v) ≠ nil do
    v ← link(v); C ← C ∪ {v};
return S, C;
```

FIG. 13. *Algorithm* SKELETON-WEIGHTED.

no further marker on L or there is one marker m_0 with $W(m_0) > W$. Comparing with the corresponding situation in P' we find the necessary action. A skeleton point of weight W is generated and the weight of marker m_0 is updated to $W(m_0) - W$. With this kind of consideration it can be verified that Algorithm SKELETON-WEIGHTED yields the weighted skeleton of (P, w).

For the maximum weighted chain computation observe that either all or none of the points of $C(p)$, for some $p \in P$, are contained in a maximum chain in P'. Thus, a maximum chain in P' corresponds to a maximum weighted chain in P and vice versa. As will be seen later the same holds true of maximum weighted k-chains in P and maximum k-chains in P'.

The weighted skeleton thus computed has many points with an equal x- or y-coordinate, and we want to compute the weighted skeleton of a skeleton repeatedly. Thus, we perturb them according to the simple procedure mentioned in section 2 before we use them in any further computation. Consequently, we may assume that all coordinates of points of the input instance are different.

Computing the weighted skeleton $S(P, w)$ with algorithm SKELETON-WEIGHTED takes time $O((|P|+|S(P,w)|)\log(|P|+|S(P,w)|))$. In contrast to the unweighted case, the weighted skeleton may contain more points than the original point set. Fortunately, there cannot be many more such points.

LEMMA 5.1. *The number of weighted skeleton points in $S(P, w)$ is at most twice the number of points in P.*

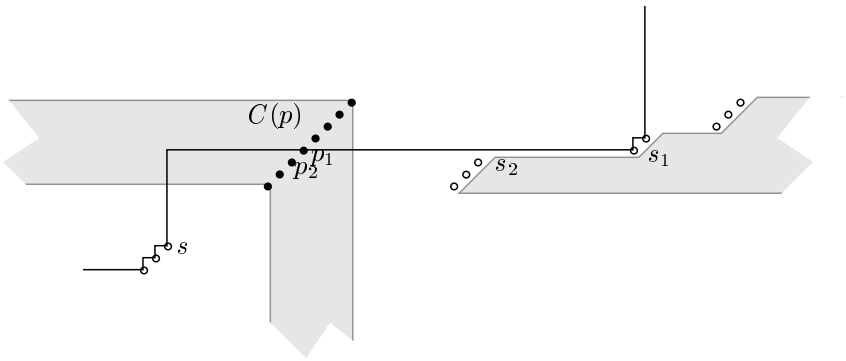


FIG. 14. *This cannot happen—a jump line of C_S separating $C(p)$. Gray areas do not contain points of P' or $S(P')$.*

Proof. If a skeleton point $s \in S(P, w)$ has no other skeleton point s' above, it is assigned to its defining point $p \in P$ below. Otherwise, it is assigned to its defining point to the left. Note that in the second case skeleton point s was generated under the condition $W \geq W(m)$; hence, m was removed and, consequently, there is no skeleton point s' to the right of s. A point $p \in P$ gets assigned the highest skeleton point above it or the furthest skeleton point to its right or both. Therefore, no point p gets assigned more than two skeleton points. □

The interesting fact now is that Algorithm MAXMULTICHAIN (see Figure 9 of section 3) may be used nearly without changes to compute a maximum weighted k-chain. The only problem is that we have not yet defined what the marginal points of a weighted point set should be. But we can do without marginal points if we resign to dispose any set of points. This does no harm since the space requirement will be high anyway.

THEOREM 5.2. *Let C_S be a maximum weighted $(k-1)$-chain in $S(P, w)$. Then taking a maximum weighted chain of points $P \cap R$ in each of the k regions R of C_S yields a maximum weighted k-chain of P.*

Proof. By induction, a maximum weighted $(k-1)$-chain C_S of $S(P)$ corresponds to a maximum $(k-1)$-chain C'_S of the skeleton $S(P')$ of P'. The selection of a maximum chain in the point sets $P' \cap R$ for each region R of the k regions of C'_S would give a maximum k-chain of P'. This does not immediately yield a maximum weighted k-chain of P since some points of the chain $C(p) \subseteq P'$ replacing a point p of P might fall in one region and some in another. We claim that this cannot happen. Since there is no skeleton point in the ϵ box containing $C(p)$, it is separated either by a vertical or horizontal segment of a jump line of some chain. It cannot be vertical, for there is no skeleton point below $C(p)$.

Now suppose that a horizontal segment between two skeleton points s and s_1 separates $C(p)$ (see Figure 14). The y-coordinate of s_1 equals the y-coordinate of some point $p_1 \in C(p)$. Let p_2 be the immediate predecessor of p_1 in $C(p)$. Since there is no skeleton point to the left of $C(p)$, the point s has to be dominated by p_2 and by s_2, the skeleton point on the outgoing line of p_2. Also $s_2 < s_1$; hence, $C_S \cup \{s_2\}$ is a $(k-1)$-chain. If s_2 is not a member of C_S this contradicts the maximality of C_S. On the other hand, s_2 cannot be a member of C_S, since C_S has a decomposition into $(k-1)$ noncrossing jump lines, one of them joining s and s_1. A jump line containing

s_2, however, has to leave s_2 upward and hence crosses the jump line of s and s_1.

We have thus proved that there is a correspondence between the regions R'_i of C'_S and the regions R_i of C_S, for $1 \le i \le k$, in such a way that a chain $C(p)$ is completely contained in R'_i iff p is contained in R_i. Applying Theorem 3.5 to the expanded point set, we obtain that all maximum weighted chains from regions R_i together form a maximum weighted k-chain. □

In analogy to the unweighted case, we set $S^0_w(P) = P$ and let $S^k_w(P) = S(S^{k-1}_w(P), w)$ be the *kth weighted skeleton* of P. We denote the weighted k-chain in P obtained from a weighted $(k-1)$-chain C_S in $S(P, w)$ according to Theorem 5.2 by $\rho_k(C_S)$.

THEOREM 5.3. *Let C_1 be a maximum weighted chain of $S^{k-1}_w(P)$ and $C_\ell = \rho_\ell(C_{\ell-1})$ for $2 \le \ell \le k$; then C_k is a maximum weighted k-chain in P.* □

THEOREM 5.4. *A maximum weighted k-chain of a weighted point set P can be obtained in $O(2^k|P| \log(2^k|P|))$ time and $O(2^k|P|)$ space.*

Proof. As was already pointed out, to construct the weighted skeleton for a weighted point set P takes time $O(|P| \log |P|)$. The point location of points P in the regions defined by a $(k-1)$-chain C_S can be done with the help of a sweep line in time $O(|P| \log |C_S| + |C_S|)$. This amounts to a total running time of $O(\sum_{1 \le i \le k} 2^i|P| \times \log(2^i|P|))$, since $|S^i_w| \le 2^i|P|$ by Lemma 5.1. The bound on the space is given by $O(\sum_{1 \le i \le k} 2^i|P|)$. □

A maximum weighted k-chain of a point set P with rational or real weights can be computed by the very same algorithm. The proof of correctness then requires some additional standard rescaling and approximation arguments.

As the algorithm of this section makes sense only for small values of k it would have been nice to have a complementary method for large k. Unfortunately, we have not been able to extend the algorithm of section 4 to the weighted case so that the running time remains independent from the weights.

REFERENCES

[1] S. EVEN, A. PNUELI, AND A. LEMPEL, *Permutation graphs and transitive graphs*, J. Assoc. Comput. Mach., 19 (1972), pp. 400–410.

[2] A. FRANK, *On chain and antichain families of partially ordered sets*, J. Combin. Theory Ser. B, 29 (1980), pp. 176–184.

[3] M. L. FREDMAN, *On computing the length of longest increasing subsequences*, Discrete Math., 11 (1975), pp. 29–35.

[4] F. GAVRIL, *Algorithms for maximum k-colorings and maximum k-coverings of transitive graphs*, Networks, 17 (1987), pp. 465–470.

[5] C. GREENE, *An extension of Schensted's theorem*, Adv. Math., 14 (1974), pp. 254–265.

[6] C. GREENE, *Some partitions associated with a partially ordered set*, J. Combin. Theory Ser. A, 20 (1976), pp. 69–79.

[7] C. GREENE AND D. J. KLEITMAN, *The structure of Sperner k-families*, J. Combin. Theory Ser. A, 20 (1976), pp. 41–68.

[8] D. E. KNUTH, *The Art of Computer Programming* III. *Sorting and Searching*, Vol. 3, Addison–Wesley, Reading, MA, 1973.

[9] R. D. LOU AND M. SARRAFZADEH, *An optimal algorithm for the maximum 3-chain problem*, SIAM J. Comput., 22 (1993), pp. 976–993.

[10] R. D. LOU, M. SARRAFZADEH, AND D. T. LEE, *An optimal algorithm for the maximum two-chain problem*, SIAM J. Discrete. Math., 5 (1992), pp. 285–304.

[11] J. MATOUŠEK AND E. WELZL, *Good splitters for counting points in triangles*, J. Algorithms, 13 (1992), pp. 307–319.

[12] M. SARRAFZADEH AND R. D. LOU, *Maximum k-covering of weighted transitive graphs with applications*, Algorithmica, 9 (1993), pp. 84–100.

[13] G. Viennot, *Une forme géométrique de la correspondance de Robinson-Schensted*, in Combinatoire et Représentation du Groupe Symétrique, D. Foata, ed., Lecture Notes in Math. 579, Springer, New York, 1977, pp. 29–58.

[14] G. Viennot, *Chain and antichain families, grids and young tableaux*, in Orders: Description and Roles, Math. Stud. 99, North–Holland, Amsterdam, 1984, pp. 409–463.

[15] L. Wernisch, *Dominance Relation on Point Sets and Aligned Rectangles*, Dissertation, Freie Universität Berlin, 1994.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 210–236

FAST ALGORITHMS FOR CONSTRUCTING t-SPANNERS AND PATHS WITH STRETCH t*

EDITH COHEN†

Abstract. The distance between two vertices in a weighted graph is the weight of a minimum-weight path between them (where the weight of a path is the sum of the weights of the edges in the path). A path has *stretch* t if its weight is at most t times the distance between its end points. We present algorithms that compute paths of stretch $2 \leq t \leq \log n$ on undirected graphs $G = (V, E)$ with nonnegative weights. The stretch t is of the form $t = \beta(2 + \epsilon')$, where β is integral and $\epsilon' > 0$ is at least as large as some fixed $\epsilon > 0$. We present an $\tilde{O}((m + k)n^{(2+\epsilon)/t})$ time randomized algorithm that finds paths between k specified pairs of vertices and an $\tilde{O}((m + ns)n^{2(1+\log_n m+\epsilon)/t})$ deterministic algorithm that finds paths from s specified sources to all other vertices (for any fixed $\epsilon > 0$), where $n = |V|$ and $m = |E|$. This improves significantly over the slower $\tilde{O}(\min\{k, n\}m)$ exact shortest paths algorithms and a previous $\tilde{O}(mn^{64/t} + kn^{32/t})$ time algorithm by Awerbuch et al. [*Proc. 34th IEEE Annual Symposium on Foundations of Computer Science*, IEEE, Piscataway, NJ, 1993, pp. 638–647]. A t-spanner of a graph G is a set of weighted edges on the vertices of G such that distances in the spanner are not smaller and within a factor of t from the corresponding distances in G. Previous work was concerned with bounding the size and efficiently constructing t-spanners. We construct t-spanners of size $\tilde{O}(n^{1+(2+\epsilon)/t})$ in $\tilde{O}(mn^{(2+\epsilon)/t})$ expected time (for any fixed $\epsilon > 0$), which constitutes a faster construction (by a factor of $n^{3+2/t}/m$) of sparser spanners than was previously attainable. We also provide efficient parallel constructions. Our algorithms are based on pairwise covers and a novel approach to construct them efficiently.

Key words. shortest paths, graph spanners, parallel algorithms

AMS subject classifications. 68Q20, 68Q22, 68Q25, 05C85, 90C27

PII. S0097539794261295

1. Introduction.

The shortest-path problem amounts to finding paths of minimum weight between specified pairs of vertices on a weighted input graph. We consider the relaxed version of computing paths with *stretch* t, where the ratio between the weight of the path and the minimum-weight path connecting the same two vertices is at most t. We provide algorithms that compute paths of stretch t (typically $2 < t < \log n$) on weighted undirected graphs much more efficiently than known previously. A related concept is t-spanners of weighted graphs. A t-spanner is a weighted graph on the same set of vertices such that the distances between pairs of vertices in the spanner are not smaller and within a factor of t of the corresponding distances on the original graph. Numerous recent works focused on bounding the size of t-spanners, constructing them efficiently, and applying them to other problems (see, e.g., Peleg and Ullman [12], Peleg and Schäffer [11], Althöfer et al. [1], and Chandra et al. [4]). We obtain significantly more efficient parallel and sequential constructions of sparse t-spanners along with improved upper bounds on their size. We note that our stretch-t paths algorithm essentially computes a sparse t-spanner and produces paths that consist of spanner edges. Our algorithms are favorable in scenarios where one needs to find short paths but has severe limitations on resources (e.g., time or number of processors) that prohibit the use of shortest-paths algorithms or when there is a flat high cost associated with each edge that is being used for some short path, and,

*Received by the editors January 10, 1994; accepted for publication (in revised form) December 10, 1996; published electronically June 15, 1998. An extended abstract appeared in *Proc. 34th IEEE Annual Symposium on Foundations of Computer Science*, 1993.
http://www.siam.org/journals/sicomp/28-1/26129.html
†AT&T Labs-Research, Florham Park, NJ 07932 (edith@research.att.com).

hence, it is desirable to find a set of short paths that utilizes only a small subset of the edges (note that shortest paths may require the use of all the edges). In addition, our algorithms are suited to find short paths on line where we allow a small amount of preprocessing and are supplied on line with query pairs of vertices. Furthermore, they can be adapted to find short paths on line on dynamic networks, where weighted edges are inserted or deleted much faster than known shortest-paths algorithms.

We consider undirected graphs $G = (V, E)$ where V is the set of vertices and E the set of edges. Associated with the edges are positive weights $w : E \to R_+$. We denote $n = |V|$ and $m = |E|$. The weight of a path is the sum of the weights of the edges on the path. A *shortest path* between $\{u_1, u_2\} \subset V$ is a path of minimum weight among all paths connecting u_1 and u_2. The *distance* between $\{u_1, u_2\} \subset V$, denoted $\mathrm{dist}\{u_1, u_2\}$, is the weight of the respective shortest path. A path between $\{u_1, u_2\} \subset V$ has *stretch* t if the ratio between the weight of the path and the distance between $\{u_1, u_2\}$ is at most t. We use the $\tilde{O}$ asymptotic notation to eliminate polylogarithmic factors in n, e.g., $\tilde{O}(f) \equiv O(f \operatorname{polylog} n)$. The parallel algorithms and resource bounds discussed in this paper are for the EREW PRAM model.

1.1. Our results. We obtain the following results. Consider a weighted graph G, a set of query pairs $Q \subset V \times V$, and a stretch factor $2 < t = O(\log n)$. The stretch t is of the form $t = \beta(2+\epsilon')$, where β is integral and $\epsilon' > 0$ is at least as large as some fixed $\epsilon > 0$. We present bounds for computing for all $(v, u) \in Q$, stretched distances $\tilde{\mathrm{dist}}\{v, u\}$ such that

$$\mathrm{dist}\{v, u\} \le \tilde{\mathrm{dist}}\{v, u\} \le t\mathrm{dist}\{v, u\} \ ,$$

with the following tradeoffs between the stretch t, the time, and the work performed:

- deterministically,

(1) when Q contains all the pairs consisting of one vertex from a set S of sources and any other vertex;
by a sequential algorithm for any fixed $\epsilon > 0$ and t such that $t/((2+\epsilon)(1+\log_n m))$ is integral, in (see Theorem 6.6) $O(n^{(2+\epsilon)(1+\log_n m)/t}(m \log n + |S| n \log n))$ time;

- with probability $1 - O(1/\operatorname{poly}(n))$. These are Las Vegas algorithms. The randomization is on the expected termination time and not on the correctness.

(2) By a sequential algorithm for any fixed $\epsilon > 0$ and t such that $t/(2+\epsilon)$ is integral in (see section 6) $O((m + |Q|)n^{(2+\epsilon)/t} t \log^2 n)$ time.

(3) For a fixed $0 \le \delta \le 1$, any fixed $\epsilon > 0$, and $\beta = O(\log n)$ (see Theorem 2.8) $t = (2\beta)^{\lceil 1/\delta \rceil}(1+\epsilon)$ in $O(n^\delta \beta^2 \log n)$ time using $\tilde{O}(n^{1+2/\beta} + (m + |Q|)n^{1/\beta})$ work.

In an on-line problem, where we perform some preprocessing and the query pairs (or query sources) are given on line, the algorithm in (1) needs $O(n^{(2+\epsilon)(1+\log_n m)/t} m \log n)$ preprocessing time and $O(n^{(2+\epsilon)(1+\log_n m)/t} n \log n)$ time per source, and the algorithm in (2) needs $O(mn^{(2+\epsilon)/t} t \log^2 n)$ preprocessing time and $O(n^{(2+\epsilon)/t} t \log^2 n)$ time per query pair.

Graph spanners. We construct t-spanners of size (number of edges) $\tilde{O}(n^{1+(2+\epsilon)/t})$ (for any $\epsilon > 0$ and t such that $t/(2+\epsilon)$ is integral). These spanners can be constructed by a randomized algorithm that runs in $\tilde{O}(mn^{(2+\epsilon)/t})$ time. We present a deterministic algorithm that finds t-spanners of size $O(n^{1+(2+\epsilon)(1+\log_n m)/t})$ (for any $\epsilon > 0$ and t such that $t/((2+\epsilon)(1+\log_n m))$ is integral) and runs in $O(mn^{(2+\epsilon)(1+\log_n m)/t})$ time. We show that in parallel, t-spanners of size $O(n^{1+(2+\epsilon)/t})$ (for any $\epsilon > 0$ and t such that $t/(2+\epsilon/2)$ is integral) can be constructed in $O(R\beta^2 \log^2 n)$ expected time

using $O(n^{1/\beta} m\beta \log^2 n)$ work or, alternatively, in $O(\beta \log^3 n)$ expected time using $O(n^3 \beta \log n)$ work, where $R = w_{\max}/w_{\min}$ and $\beta = t/(2 + \epsilon/2)$.

1.2. Previous work. Peleg and Schäffer[11] proved the following bounds on the size of spanners of unweighted graphs. They showed that for infinitely many values of n there exist n-vertex graphs such that all their t-spanners contain $\Omega(n^{1+1/t})$ edges. They also gave a polynomial algorithm that computes t-spanners with $O(n^{1+4/t})$ edges. For weighted graphs, Althöfer et al. [1] gave a simple greedy algorithm that constructs t-spanners of size $O(n^{1+2/(t-1)})$. In a subsequent paper, Chandra et al. [4] showed that the greedy algorithm of [1] has an $O(n^{3+4/(t-1)})$ time implementation. Hence, we tighten the known upper bound on the size of t-spanners and present a significantly more efficient sequential and parallel constructions than previously known. The key to achieving these improvements is the fact that all previous constructions of t-spanners were based on solving many instances of the exact shortest-paths problem, where in this paper we avoid that.

The fastest known sequential algorithm for computing exact shortest paths from a set of sources S to all other vertices requires $\tilde{O}(|S|m)$ time using weighted breadth-first search (BFS). (See, e.g., [8].) Awerbuch, Berger, Cowen, and Peleg [2] presented a faster algorithm for computing paths of stretch $t > 64$. They showed that paths with stretch $64 \leq t = O(\log n)$ between k pairs of vertices can be obtained in $\tilde{O}((mt + nt^2)n^{64/t} + ktn^{32/t})$ time. Hence, in particular, paths with stretch t from a set of sources S to all other vertices can be obtained in $\tilde{O}((mt + nt^2)n^{64/t} + |S|tn^{1+32/t})$ time. Note that when $t = O(\log n)$ the algorithm produces paths with stretch $O(\log n)$ and runs in time $\tilde{O}(m + |S|n)$.

The best-known parallel shortest path algorithms generally perform significantly more work (product of time and number of processors) than their sequential counterparts. The best-known work bound for an $\mathcal{NC}$ algorithm for the (weighted) single-source shortest-paths problem is $\tilde{O}(n^3)$ (the algorithm essentially solves the more general all-pairs problem). Other algorithms exhibit tradeoffs between work and time. Spencer [13] presented an algorithm for the single-source problem that runs in $\tilde{O}(\delta)$ time using $\tilde{O}(n^3/\delta^2 + m)$ work, where the ratio of the largest edge weight to the smallest edge weight is polynomial in n. The author gave an algorithm that computes shortest paths from $|S|$ sources in $\tilde{O}(\delta)$ time using $O(|S|n^2 + n^3/\delta^3)$ work [6]. The algorithm computes shortest paths if the ratio of the largest edge weight to the smallest edge weight is polynomial in n and otherwise can be adapted to find paths whose weights are within a factor of $(1+1/\operatorname{poly}(n))$ from the respective distances. In addition, the author showed that paths with stretch $(1+\epsilon)$ (for a fixed ϵ) can be computed in $\tilde{O}(\delta)$ time using $\tilde{O}(|S|(n^2/\delta + m) + n^3/\delta^2)$ work. Another work-time tradeoff for the single-source shortest paths problem was given by Klein and Sairam [10] who presented a randomized algorithm that finds paths with stretch $(1 + \epsilon)$ (for a fixed ϵ) in $\tilde{O}(n^{0.5})$ time using $\tilde{O}(mn^{0.5})$ work.

Note that the tradeoffs obtained by our sequential randomized algorithm (see (2)) improve over the algorithm of Awerbuch et al. [2]. The tradeoffs obtained by our deterministic sequential algorithm (see (1)) improve over the algorithm of Awerbuch et al. for multisource short-paths problems. The improvements reduce the time bound significantly and are particularly meaningful in the more interesting range where the stretch is small. As a concrete example consider computing paths between $O(m)$ arbitrary pairs of vertices. The best shortest-paths algorithm for this problem runs in time $\tilde{O}(mn)$ (essentially solves the all-pairs problem). Our randomized algorithm in time $\tilde{O}(mn^{0.5})$ finds paths with stretch 4. The Awerbuch et al. algorithm cannot

guarantee paths of stretch 4 and in time $\tilde{O}(mn^{0.5})$ computes paths with stretch 128. Implications of our parallel tradeoffs are that for any fixed $\epsilon > 0$, paths from s sources to all other vertices with a fixed constant stretch can be computed in $O(n^{\epsilon})$ time using $O((m+sn)n^{\epsilon})$ work and with a polylogarithmic stretch, in $O(n^{\epsilon})$ time using $\tilde{O}(m+sn)$ work.

1.3. Overview. The main tool that we use to compute spanners and paths of stretch t are *pairwise covers* of graphs and efficient novel constructions of these covers. Pairwise covers are a modification of sparse neighborhood covers (see Awerbuch and Peleg [3]). Sparse neighborhood covers were employed by Awerbuch et al. [2] to obtain a sequential algorithm for paths of a specified stretch. Pairwise covers turn out to be a more correct notion for computing paths of stretch t. Our constructions allow for significantly improved sequential bounds. In addition, we present parallel algorithms based on efficient parallel cover constructions. The construction of neighborhood covers in [2] seems to be inherently sequential.

A pairwise cover of a graph is defined with respect to parameters W and β. It consists of a collection of subsets of vertices (*clusters*) with the following properties: the sum of the sizes of the clusters is $\tilde{O}(n^{1+1/\beta})$, the maximal distance between any pair of vertices which are in the same cluster is at most $2\beta W$, and for every pair of vertices of distance at most W from each other there exists at least one cluster in the cover that contains both vertices. The value of β determines a tradeoff between the maximum diameter of a cluster (that we would like to be as small a multiple of W as possible) and the total size of the cover (that we would like to be as close to $O(n)$ as possible). For any $\beta \geq 2$ and a fixed $\epsilon > 0$, a logarithmic number of pairwise covers for different values of W can be used to compute paths with stretch $t = 2(1+\epsilon)\beta$. The work performed depends on the size of the cover. To achieve efficient parallel constructions of pairwise covers, we refine the definition to be with respect to an additional "path size" parameter ℓ. For a parameter $1 \leq \ell \leq n$, the ℓ-limited distance between a pair of vertices is the weight of the minimum-weight path connecting them among paths consisting of at most ℓ edges (this terminology is from [10]). In pairwise covers with parameter ℓ, we have a relaxed condition that pairs of vertices where the ℓ-limited distance between them is at most W must be both contained in at least one cluster. This relaxed requirement allows us to construct pairwise covers efficiently in parallel in time $\tilde{O}(\ell)$. The most technical and novel part of this paper is the construction of pairwise covers. It is also the only place where randomization is used.

We sketch our t-spanner constructions. For any $\beta \geq 2$ and a fixed $\epsilon > 0$, we construct a logarithmic number of pairwise covers for different values of W. We use $W \in \{(1+\epsilon)^i W_{\min} | i > 0\}$, where $W_{\min}$ is the smallest path weight. Our construction of pairwise covers has the property that for every cluster, the algorithm computes a shortest-path tree rooted at a vertex of the cluster such that distances from the root to all other vertices in the cluster are at most β. Hence, distances on the tree between any two vertices in the cluster are bounded by 2β. Roughly, for $t = 2(1+\epsilon)\beta$, the algorithm constructs a t-spanner by considering each cluster in each of the covers and augmenting the spanner with the edges of the respective shortest-paths tree. Our spanner contains a path of stretch t between every pair of vertices. We can obtain t-spanners with the additional property of size-2 paths if the property that the spanner is a subgraph of the original graph is compromised. This can be achieved if for each cluster, instead of adding the edges of the tree to the spanner, we add edges from the root to all other vertices in the cluster with appropriate weights. For every pair of

vertices, our spanner contains a path of stretch t that uses at most two edges. We comment that the sparse neighborhood covers constructed by Awerbuch et al. [2] can similarly be used to obtain spanners, but the resulting spanners have very large size compared with previous spanner algorithms.

The paper is organized as follows. In section 2 we define pairwise covers, state the resource bounds for computing them, and employ them to compute stretch-t paths. In section 3 we discuss the computation of a pairwise cover with respect to parameters W and ℓ. We reduce the problem to finding a cover with respect to a set of integral weights and parameters $\ell' = n$ and $W' = O(\ell)$. In section 4 we discuss computing neighborhoods of vertices up to specified distances using work linear in the size of the neighborhood. Sections 5 and 6 contain the most novel part of the paper. In section 5 we present a parallel algorithm for computing a pairwise cover in integral weighted graphs. The running time of the algorithm is linearly dependent on W, and the work performed is roughly the sum over the clusters of the number of edges incident at cluster vertices. The neighborhoods of section 4 are used in section 5 as the clusters constituting a cover. In section 6 we discuss sequential stretch-t paths algorithms. A randomized sequential algorithm is obtained as a simplified version of the parallel randomized algorithm introduced in previous sections. We note that for the sequential randomized algorithm Sections 2.3, 2.4, 3, and 4 may be skipped. We also present a deterministic algorithm that is based on deterministic construction of covers that is fairly different from the randomized construction used in section 5. Section 7 is concerned with applications to graph spanners. In section 8 we discuss computing stretch-t distances on dynamic networks, where edges are inserted and deleted. In section 9 we discuss issues and open problems that arise from this work.

We remark that although the algorithms presented in this paper compute distances, a concise representation of paths with corresponding weights can be produced within the same resource bounds.

Notation. Consider an undirected graph $G = (V, E)$, with positive weights $w : E \to R_+$ associated with the edges. The *size* of a path is the number of edges in the path. For $\ell \in \mathcal{N}$, an ℓ-limited path is a path of size at most ℓ. We use the terms *shortest-path* and *minimum-weight* path interchangeably for a path of minimum weight. For a subset of edges $E' \subseteq E$, we denote $w(E') = \sum_{e \in E'} w(e)$. For a subset of vertices $V' \subseteq V$, we denote by $\mathcal{E}(V') = \{(u_1, u_2) \in E | u_1 \in V' \wedge u_2 \in V'\}$ the edges in the subgraph induced by V'. For a subset of edges $E' \subseteq E$, two vertices $\{u_1, u_2\} \subseteq V$, an integer $1 \leq \ell \leq n$, and a weight function $\hat{w} : E' \to R_+$, we denote by $\mathrm{dist}^{\ell}_{E',\hat{w}}\{u_1, u_2\}$ the weight, according to $\hat{w}$, of the minimum weight path in E' between u_1 and u_2 that consists of at most ℓ edges. (When ℓ is omitted, presume $\ell = n$, and when E' is omitted, presume $E' = E$. Generally, when the weights $\hat{w}$ are clear from the context they are omitted from the subscript.) For a subset $V' \subseteq V$, a vertex $v \in V'$, an integer $1 \leq \ell \leq n$, and a scalar $W \in R_+$, denote by $N^{\ell}_{W}(V', v) \subset V'$ the set of all the vertices $u \in V'$ such that there exists a path from u to v of size at most ℓ and weight at most W in the subgraph induced by V'. (When $V' = V$, we omit the first parameter. If ℓ is omitted, presume $\ell = n$.) When we state that an algorithm terminates with probability of success $1 - O(1/\operatorname{poly}(n))$, it mean that we can replace $\operatorname{poly}(n)$ by any polynomial by adjusting constant factors in the resource bounds of the algorithm. Note that if algorithms with probability of success $1 - O(1/\operatorname{poly}(n))$ are called as subroutines a polynomial number of times, then all of them terminate successfully with probability $1 - O(1/\operatorname{poly}(n))$.

2. Using pairwise covers for stretched distances. In this section we define pairwise covers and employ them to compute stretch-t paths. We note that in the context of shortest-path computations, the assumption that the weights are strictly positive does not limit the generality since (i) edges of zero weight can be contracted (the contractions can be performed well within the resource bounds of the algorithms presented here by using a connected components algorithm), and (ii) edges of negative weight induce negative cycles (by traversing the edge back and forth) that implies that all the distances between vertices that are in the same connected component as the negative weight edge are unbounded from below.

REMARK 2.1. *We make the following assumption throughout the paper and argue that it does not limit generality. We assume that the ratio of the largest to smallest edge weight in a graph is* $O(\text{poly}(n))$. *(That is,* $\max_{e\in E} w(e)/\min_{e\in E} w(e) = O(\text{poly}(n))$.*) We justify this assumption by using the following reduction of Klein and Sairam* [10] (*see also* [6]) *when the weights are general. Klein and Sairam had shown how to compute for a weighted input graph* G *a collection of weighted graphs with a total number of vertices* $O(n \log n)$ *and total size* $O(m \log n)$ *such that* (i) *in each of the graphs in the collection, the ratio of the largest to smallest edge weights is* $O(\text{poly}(n))$ *and* (ii) *a shortest-path problem on the original graph can be translated efficiently to a shortest-path problem on the graphs of the collection. A path in* G *obtained this way is guaranteed only to have stretch* $(1 + 1/\text{poly}(n))$, *but since in this paper we obtain paths of stretch no better than* 2, *we can safely allow that. The computation involved in obtaining this collection amounts to applying a connected components algorithm and is well within the resource bounds of the algorithms presented here.*

This section is arranged as follows. In subsection 2.1 we define pairwise covers. A pairwise cover of a graph is, roughly, a collection of subsets of vertices (clusters) defined with respect to a scalar $W \in R_+$, an integer ℓ (the *path size* of the cover), and a parameter β. The clusters are such that every pair of vertices with ℓ-limited distance of at most W must be contained in at least one cluster. The parameter β determines a tradeoff between the maximum diameter of a cluster and the size of the cover. In later sections we present algorithms for computing a pairwise cover. We shall see that the parallel running time depends linearly on the path size ℓ, and hence we would like to use covers with small path sizes. In subsection 2.2 we present an algorithm that computes stretched ℓ-limited distances from a set of query vertices to all other vertices. The algorithm utilizes a logarithmic number of pairwise covers with path size ℓ but with different values for the parameter W. Recall that the time bound of our cover algorithm depends linearly on the path size ℓ, and hence we would like to avoid using the algorithm for large values of ℓ. Unfortunately, however, ℓ-limited distances for $\ell < n$ may not yield any information about the actual distances in the graph. To overcome this difficulty we seek a method to replace the current set of edges E by a set of weighted edges E' such that for some $\ell << n$, the ℓ-limited distances in E' are within some factor of the distances with respect to E. The following definition formalizes this concept.

DEFINITION 2.2. *Consider a graph* $G = (V, E)$ *with weights* $w : E \to R_+$, *and a set* $E' \subset V \times V$ *of edges with weights* $w' : E' \to R_+$. *We say that the weighted graph* (V, E') (ℓ, t)-approximates G *if for every pair of vertices* $\{u_1, u_2\} \subset V$, *the following inequalities hold:*

$$\text{dist}_E\{u_1, u_2\} \le \text{dist}_{E'}\{u_1, u_2\} \le \text{dist}^{\ell}_{E'}\{u_1, u_2\} \le t\text{dist}_E\{u_1, u_2\} \ .$$

We refer to the parameter ℓ as the *path size* of the approximation and to the parameter t as the *stretch*. Note that since minimum-weight paths are of size at most n, any graph $(n,1)$-approximates itself. In subsection 2.3 we introduce an algorithm that for an input graph G produces a graph G' that (ℓ,t)-approximates G for some $\ell << n$. The algorithm utilizes pairwise covers with typical path size of n^μ (for small $\mu > 0$) and exhibits a tradeoff between the magnitude of μ and the stretch factor. In subsection 2.4 we combine the algorithms of subsections 2.2 and 2.3 to obtain stretched distances: initially we apply the algorithm of subsection 2.3 to the graph G and obtain an approximation G' of G. Subsequently, we apply the algorithm of subsection 2.2 to G'.

2.1. pairwise covers. Pairwise covers are a modification of sparse neighborhood covers that were introduced by Awerbuch and Peleg [3]. Sparse neighborhood covers were employed by Awerbuch et al. [2] for stretch-t path computations.

DEFINITION 2.3 (pairwise cover). *Consider a graph $G = (V,E)$ with weights $w : E \to R_+$, integers $1 \leq \ell \leq n$ and $\beta = O(\log n)$, and scalars $\rho \in R_+$ and $0 \leq \hat{\epsilon} < 1/2$. A* pairwise $(\ell,\beta,\rho,\hat{\epsilon})$-cover *(for brevity, $(\ell,\beta,\rho,\hat{\epsilon})$-cover) of G is a collection of sets of vertices $X_1,\ldots,X_k$* (clusters) *and vertices $v_1,\ldots,v_k$ (where for $1 \leq i \leq k$, $v_i \in X_i$ is the* center *of the cluster X_i) such that*

1. *$\forall \{u,v\} \subset V$, such that $\mathrm{dist}^\ell\{u,v\} \leq \rho/(1+\hat{\epsilon})$, $\exists i$ such that $\{u,v\} \subset X_i$,*
2. *$\forall i$, $\forall u \in X_i$, $\mathrm{dist}\{v_i,u\} \leq \beta\rho$, and*
3. *$\forall v \in V$, $|\{i | v \in X_i\}| = O(n^{1/\beta}\beta \log n)$. (Every vertex belongs to $O(n^{1/\beta}\beta\log n)$ clusters.)*

In sections 3–5 we present an algorithm that computes an $(\ell,\beta,\rho,\hat{\epsilon})$-cover within the following bounds.

THEOREM 2.4. *An $(\ell,\beta,\rho,\hat{\epsilon})$-cover of G can be computed with probability $1 - O(1/\mathrm{poly}(n))$ in $O(\ell\hat{\epsilon}^{-1}\beta^2 \log n)$ time using $O(n^{1/\beta}m\hat{\epsilon}(\log n)/(\ell\beta))$ processors. If we allow $O(\ell\hat{\epsilon}^{-1}n^\alpha\beta^2)$ time (for a fixed $\alpha > 0$), the cover algorithm uses $O(\beta n^{1/\beta}m \log n)$ work.*

REMARK 2.5. *We remark that a simpler sequential version of the algorithm produces an $(n,\beta,\rho,0)$-cover in $O(mn^{1/\beta}\beta\log n)$ time. The sequential version is immediate after presenting the parallel algorithm, and hence it is omitted.*

The probabilistic behavior manifests itself either in not terminating within the stated resource bounds or by producing a set of clusters for which property 3 of Definition 2.3 does not hold. Note that properties 1 and 2 always hold and property 3 can be easily verified. Hence, we can assume that if the algorithm terminates within the stated resource bounds it returns a valid cover. The algorithm can be modified to compute for each cluster a shortest-paths tree rooted at the center vertex. The tree distances from the center to other vertices in the cluster are at most $\beta\rho$.

2.2. Obtaining stretched limited distances. The inputs to the following algorithm are a graph G with weights $w : E \to R_+$, integers $1 \leq \ell \leq n$ and $\beta \geq 1$, a scalar $0 \leq \hat{\epsilon} \leq 1/2$, and a set of query pairs $Q \subset V \times V$. The algorithm computes for every $\{v,u\} \in Q$ a stretched distance $\tilde{\mathrm{dist}}\{v,u\}$ such that

$$\mathrm{dist}_E\{v,u\} \leq \tilde{\mathrm{dist}}\{v,u\} \leq 2\beta(1+\hat{\epsilon})^2\mathrm{dist}^\ell_E\{v,u\} \ .$$

We remark that the algorithm employs pairwise covers in a similar manner to the use of sparse neighborhood covers made by the Awerbuch et al. [2] algorithm.

ALGORITHM 2.6 (compute distances).

1. $w_{\min} \leftarrow \min_{e \in E} w(e)$
   $w_{\max} \leftarrow \max_{e \in E} w(e)$
   $r \leftarrow \lceil \log_{1+\hat{\epsilon}}(n w_{\max}/w_{\min}) \rceil$
   For $i = 0, \ldots, r$: $w_i \leftarrow w_{\min}(1+\hat{\epsilon})^i$
2. *For* $i = 0, \ldots, r$ *do (in parallel):*
   (a) *Compute* $(\ell, \beta, w_i, \hat{\epsilon})$*-cover* χ_i
   (b) *For each vertex* $v \in V$, $x(i,v) \leftarrow \{X \in \chi_i | v \in X\}$.
   (The sets $x(i,v)$ *are sorted lists of indexes of clusters.)*
3. *For all* $\{v,u\} \in Q$: $\tilde{\text{dist}}\{v,u\} \leftarrow 2\beta w_i$,
   where $i \leftarrow \min\{j | x(j,u) \cap x(j,v) \neq \emptyset\}$.

Correctness. Consider a pair of vertices $\{v,u\} \in Q$. Let $i = \min\{j | x(j,u) \cap x(j,v) \neq \emptyset\}$. It suffices to show that

1. $\text{dist}_E\{v,u\} \leq 2\beta w_i = \tilde{\text{dist}}_E\{v,u\}$ and
2. $\tilde{\text{dist}}_E\{v,u\}/(2\beta(1+\hat{\epsilon})^2) = w_i/(1+\hat{\epsilon})^2 \leq \text{dist}^{\ell}_E\{v,u\}$.

Let $X \in \chi_i$ be such that $\{u,v\} \subset X$. (The existence of such an X is immediate from the selection of i.) The first inequality follows from property 2 of a cover (see Definition 2.3). We prove that the second inequality holds. If $i = 0$, the proof is immediate since for any two vertices $\{u_1,u_2\}$, $\text{dist}^{\ell}_E\{u_1,u_2\} \geq \text{dist}_E\{u_1,u_2\} \geq w_0$. It follows from property 1 of a cover that if some j is such that $\text{dist}^{\ell}_E\{v,u\} \leq w_j/(1+\hat{\epsilon})$, then there exists $X \in \chi_j$ for which $\{v,u\} \subset X$, and hence, $x(j,u) \cap x(j,v) \neq \emptyset$. For every $j < i$, $x(j,u) \cap x(j,v) \neq \emptyset$. Therefore, $\text{dist}^{\ell}_E\{v,u\} > w_j/(1+\hat{\epsilon})$. Thus, taking $j = i-1$ we get $\text{dist}^{\ell}_E\{v,u\} > w_{i-1}/(1+\hat{\epsilon}) = w_i/(1+\hat{\epsilon})^2$.

Complexity. Recall (see Remark 2.1) that we assume $w_{\max}/w_{\min} = O(\log n)$ and therefore $r = O(\log n/\log(1+\hat{\epsilon})) = O(\hat{\epsilon}^{-1}\log n)$. Consider performing step 2 for one value $0 \leq i \leq r$. It follows from Theorem 2.4 that with probability $1 - O(1/\text{poly}(n))$, the construction of the cover χ_i can be performed in $O(\ell\hat{\epsilon}^{-1}\log n\beta^2)$ time using $O(mn^{1/\beta}\hat{\epsilon}/(\ell\beta))$ processors. (Thus, using $O(mn^{1/\beta}\beta\log n)$ work.) Hence, with probability $(1-1/\text{poly}(n))^r = 1 - O(1/\text{poly}(n))$, the computation of all the covers $\chi_1, \ldots, \chi_r$ terminates within $O(\ell\hat{\epsilon}^{-1}\log n\beta^2)$ time and $O(\hat{\epsilon}^{-1}mn^{1/\beta}\beta\log^2 n)$ work. It follows from standard PRAM techniques that the construction of the sets $x(i,v)$ (for all $v \in V$) can be performed in $T = \Omega(\log n)$ time using $O(\sum_{X \in \chi_i}|X|(\log n)/T) = O(n^{1+1/\beta}\beta\log^2 n/T)$ processors or, when allowing $\Omega(n^{\alpha})$ time for a fixed $\alpha > 0$, using $O(n^{1+1/\beta}\beta\log n)$ work. (Since property 3 of Definition 2.3 implies that $\sum_{X \in \chi_i}|X| = O(n^{1+1/\beta}\beta\log n)$.) The computation of the stretched distances (step 3) can be performed in $T = \Omega(r)$ time using

$$O\left(\sum_{\{v,u\} \in Q}\sum_{i}(|x(i,u)| + |x(i,v)|)/T\right) = O(\hat{\epsilon}^{-1}|Q|n^{1/\beta}\beta\log^2 n/T)$$

processors (since property 3 asserts that for all $v \in V$ and $1 \leq i \leq r$, $|x(i,v)| \leq n^{1/\beta}\beta\log n$). Hence, Algorithm 2.6 can be implemented to run in $O(\ell\hat{\epsilon}^{-1}\log n\beta^2)$ time using $O((m+|Q|)n^{1/\beta}\hat{\epsilon}^{-1}\beta\log^3 n)$ work and have probability of success $1 - O(1/\text{poly}(n))$ (or when allowing $\Omega(\ell\hat{\epsilon}^{-1}n^{\alpha}\beta^2)$ time using $O((m+|Q|)n^{1/\beta}\hat{\epsilon}^{-1}\beta\log^3 n)$ work).

2.3. Obtaining approximations with small path size. In this subsection we present an algorithm that produces approximations (in the sense of Definition 2.2) with small path sizes for input graphs. The algorithm inputs a weighted graph $G =$

(V,E), with weights $w : E \to R_+$, integers $1 \leq \ell' \leq n$ and $\beta \geq 1$, and a scalar $0 \leq \hat{\epsilon} \leq 1/2$. The algorithm computes a set of edges $E' \subset V \times V$ with weights $w' : E' \to R_+$ such that

1. $|E'| = O(\hat{\epsilon}^{-1} n^{1+1/\beta} \beta \log^2 n)$, and
2. for every $\ell' \leq \ell \leq n$ and every pair of vertices $\{u_1, u_2\} \subset V$ the following holds:

$$\mathrm{dist}_E\{u_1,u_2\} \leq \mathrm{dist}_{E'}\{u_1,u_2\}$$

$$\leq \mathrm{dist}^{2\lceil \ell/\ell' \rceil}_{E'}\{u_1,u_2\} \leq 2\beta(1+\hat{\epsilon})^2 \mathrm{dist}^{\ell}_{E}\{u_1,u_2\} \ .$$

Hence, the graph (V,E') is a $(2\lceil n/\ell' \rceil, 2\beta(1+\hat{\epsilon})^2)$-approximation of G. Furthermore, if $G=(V,E)$ is an (ℓ,t)-approximation of some graph $(V,\hat{E})$ for some parameters ℓ and t, then the graph (V,E') produced by the algorithm is a $(2\lceil \ell/\ell' \rceil, 2t\beta(1+\hat{\epsilon})^2)$-approximation of $(V,\hat{E})$. We first present the algorithm and analyze its resource bounds. Later on we study and analyze the resource bounds of repeated applications of the algorithm.

ALGORITHM 2.7 (approximation with small path size).

1. $w_{\min} \leftarrow \min_{e \in E} w(e)$
   $w_{\max} \leftarrow \max_{e \in E} w(e)$
   $r \leftarrow \lceil \log_{1+\hat{\epsilon}}(n w_{\max}/w_{\min}) \rceil$
   For $i = 0, \ldots, r$: $w_i \leftarrow w_{\min}(1+\hat{\epsilon})^i$
   $E' \leftarrow \emptyset$

2. *For* $i = 0, \ldots, r$ *do (in parallel):*
   (a) *Compute* $(\ell', \beta, w_i, \hat{\epsilon})$*-cover* χ_i
   (b) *For each cluster* $X \in \chi_i$ *with center* $v \in X$*:*
   add to E' *edges of weight* $w_i\beta$ *from* v *to all other vertices in* X*:*
   $E' \leftarrow E' \bigcup_{X \in \chi_i} \{(v,u) | u \in X\}$
   $\forall u \in X, w'((v,u)) \leftarrow w_i\beta$

Correctness. We first bound the size of E'. Observe that when executing step 2 for some value $0 \leq i \leq r$, at most $\sum_{X \in \chi_i} |X-1|$ edges are produced for E'. Hence, it follows from property 3 of Definition 2.3 and the assumption that $r = O(\hat{\epsilon}^{-1} \log n)$ (see subsection 2.2) that when the algorithm terminates $|E'| = O(\hat{\epsilon}^{-1} n^{1+1/\beta} \beta \log^2 n)$. It remains to show that distances in (V,E') comply with the stated inequalities. It follows from the construction of E' and property 2 of Definition 2.3 that if $e = (u_1,u_2) \in E'$ then $w'(e) \geq \mathrm{dist}_E\{u_1,u_2\}$. Hence, for every pair of vertices $\{u_1,u_2\} \subset V$, $\mathrm{dist}_E\{u_1,u_2\} \leq \mathrm{dist}_{E'}\{u_1,u_2\}$. (Thus, the leftmost inequality holds.) We show that the rightmost inequality holds. Consider a pair of vertices $\{u_1,u_2\} \subset V$ and let $D = \mathrm{dist}^{\ell'}_E\{u_1,u_2\}$. We show that there exists a path p of size at most 2 between u_1 and u_2 in E' such that $w'(p) \leq 2\beta(1+\hat{\epsilon})^2 D$. Let $i = \min\{j | \exists X \in \chi_j \text{ s.t. } \{u_1,u_2\} \subset X\}$. We have $D > w_i/(1+\hat{\epsilon})^2$ (by an argument similar to the one given in the correctness analysis of Algorithm 2.6). Let $X \in \chi_i$ be such that $\{u_1,u_2\} \subset X$ and let v be the center of X. We assume (without loss of generality) that if $v \in \{u_1,u_2\}$ then $v = u_2$. By definition of E', the edges $e_1 = (v,u_1)$ and (if $u_2 \neq v$) $e_2 = (v,u_2)$ are contained in E' and are such that $w'(e_1) = w'(e_2) = \beta w_i$. Hence (if $u_2 \neq v$), the edges e_1 and e_2 constitute a path p of size 2 between u_1 and u_2 in E' of weight $w'(p) = 2\beta w_i$. If $u_2 = v$, then the edge $e_1 = (u_1,u_2) \in E'$ constitutes a path p of size 1 and weight $w'(e_1) = \beta w_i$ between u_1 and u_2. Since $D > w_i/(1+\hat{\epsilon})^2$, we conclude that $w'(p) \leq 2\beta(1+\hat{\epsilon})^2 D$.

Consider a path p of size at most ℓ in E. To conclude the correctness proof, it suffices to show that there exists a path p' of size at most $2\lceil \ell/\ell' \rceil$ in E' such that $w'(p') \leq 2\beta(1+\hat{\epsilon})^2 w(p)$. Treat p as a list of edges and consider a partition of p to $k \leq \lceil \ell/\ell' \rceil$ segments $p_1, \ldots, p_k$, each of size at most ℓ'. By the argument given above, for each segment p_i $(1 \leq i \leq k)$ there is a corresponding path p'_i in E' of size at most 2 between the end vertices of p_i such that $w'(p'_i) \leq 2\beta(1+\hat{\epsilon})^2 w(p_i)$. Denote $p' = \bigcup_{i=1}^{k} p'_i$. Note that the size of p' is at most $2k \leq 2\lceil \ell/\ell' \rceil$ and that

$$w'(p') = \sum_{i=1}^{k} w'(p'_i) \leq 2\beta(1+\hat{\epsilon})^2 \sum_{i=1}^{k} w(p_i) = w(p) \ .$$

Complexity. The complexity of the algorithm is dominated by the computation of the covers at step 2. The algorithm performs step 2 in parallel for $r = O(\hat{\epsilon}^{-1} \log n)$ values of i. Hence, it follows from Theorem 2.4 that Algorithm 2.7 can be implemented to run with probability of success $1 - O(1/\operatorname{poly}(n))$ in $O(\ell' \hat{\epsilon}^{-1} \log n \beta^2)$ time using $O(\hat{\epsilon}^{-1} m n^{1/\beta} \beta \log^3 n)$ work (or, alternatively, in $\Omega(\ell' \hat{\epsilon}^{-1} n^{\alpha} \beta^2)$ time using $O(\hat{\epsilon}^{-1} m n^{1/\beta} \beta \log^2 n)$ work).

Repeated applications of Algorithm 2.7. Algorithm 2.7 can be applied repeatedly to obtain approximations of smaller path size but larger stretch. In each run, the algorithm is applied to the set of edges produced by the previous run. Suppose a graph (V, E) is an (ℓ, t)-approximation of some graph $(V, \hat{E})$. Consider k (for meaningful results we always have $k = o(\log n)$) repeated applications of Algorithm 2.7 starting with (V, E). Denote by E_i the set of edges computed by the ith run of the algorithm (where $E_0 \equiv E$). The graph (V, E_i) is the input for the $i+1$st run of the algorithm. Note that for $i > 0$, $|E_i| = O(\hat{\epsilon}^{-1} n^{1+1/\beta} \beta \log^2 n)$. Hence, the resulting resource bounds for k repeated applications of the algorithm are $O(k\ell' \hat{\epsilon}^{-1} \log n \beta^2)$ time using $O(\hat{\epsilon}^{-1} m n^{1/\beta} \beta \log^2 n + k\hat{\epsilon}^{-2} n^{1+2/\beta} \beta^2 \log^4 n)$ work. The set E_k produced by the kth run of the algorithm is such that (V, E_k) constitutes a $(2^k \lceil \ell/(\ell')^k \rceil, t(1+\hat{\epsilon})^{2k}(2\beta)^k)$-approximation of $(V, \hat{E})$. We typically use the above where $\hat{\epsilon}$ and k are fixed constants. We remark that under some conditions we can get further improvements on the above bounds. Note that the work bound stated above may have an imbalance between the two terms. If $m >> n^{1+1/\beta}$ we can "afford" smaller values of β and within the same resource bounds achieve smaller stretch. Suppose we use for the parameter β the values β_1 and $\beta_2 = 1/(\log_n m - 1)$ alternatingly in repeated applications of the algorithm. In the first run, that is applied to the graph (V, E), we use β_1. We assume that k and $\hat{\epsilon}$ are fixed. It follows that the algorithm runs in $O(\ell' \log n(\beta_1^2 + \beta_2^2))$ time using $O(m n^{1/\beta_1} \log^4 n)$ work. The graph (V, E_k) is a $(2^k \lceil \ell/(\ell')^k \rceil, t 2^k \beta_1^{\lceil k/2 \rceil} \beta_2^{\lfloor k/2 \rfloor} (1+\hat{\epsilon})^{2k})$-approximation of $(V, \hat{E})$.

2.4. Computing stretched distances. Observe that any weighted graph is an $(n, 1)$-approximation of itself. We can obtain stretch-t paths in a graph (V, E) by (i) applying Algorithm 2.7 repeatedly to obtain an approximation (V, E') of (V, E) with small path size, and (ii) applying Algorithm 2.6 to (V, E').

THEOREM 2.8. *For a weighted graph $G = (V, E)$, a set of query pairs $Q \subset V \times V$, a scalar $\beta \geq 1$, and fixed $0 \leq \epsilon \leq 1$ and $0 \leq \delta \leq 1$, we can obtain paths with stretch*

$$(2\beta)^{\lceil 1/\delta \rceil}(1+\epsilon)$$

between all pairs of vertices in Q in $O(n^{\delta} \log n \beta^2)$ time using $O(n^{1+2/\beta} \beta^2 \log^4 n + (m + |Q|) n^{1/\beta} \beta \log^2 n)$ work with probability of success $1 - O(1/\operatorname{poly}(n))$.

Proof. We choose $k = \lceil 1/\delta \rceil - 1$ and $\hat{\epsilon} < (1+\epsilon)^{(-2\lceil 1/\delta \rceil)} - 1$. We apply k repeated application of Algorithm 2.7 to (V, E) with $\ell' = 2n^{\delta}$. Denote by (V, E') the resulting graph (obtained with probability $1 - O(1/\operatorname{poly}(n))$. It follows (see subsection 2.3) that (V, E') constitutes an approximation of (V, E) with path size $\lceil n^{\delta} \rceil$ and stretch

$$(2\beta(1+\hat{\epsilon})^2)^k.$$

The graph (V, E') is obtained in $O(n^{\delta} \log n \beta^2)$ time using

$$O(n^{1+2/\beta}\beta^2 \log^4 n + mn^{1/\beta}\beta \log^2 n)$$

work. We apply Algorithm 2.6 to (V, E') with parameter $\ell = \lceil n^{\delta} \rceil$ and query set Q. We return the stretched distances found by Algorithm 2.6. It is easy to verify that these distances are within a factor of

$$(2\beta(1+\hat{\epsilon})^2)^{\lceil 1/\delta \rceil} \leq (2\beta)^{\lceil 1/\delta \rceil}(1+\epsilon)$$

of the corresponding shortest distances in G. The resource bounds of the application of Algorithm 2.6 are $O(n^{\delta} \log n \beta^2)$ time using $O((|E'| + |Q|)n^{1/\beta}\beta \log^2 n)$ work. By combining the above we obtain the stretch and resource bounds claimed in the statement of the theorem.

It follows that for any fixed $\epsilon > 0$, by choosing $\delta < \epsilon$ and $\beta > 2/\epsilon$, paths with constant stretch can be obtained in $O(n^{\epsilon})$ time using $O(n^{\epsilon}(|Q| + n))$ work. In [6] the author presented an algorithm that for an input graph $G = (V, E)$ and parameter $\mu \geq 0.5$ produces a graph that is a $(6n^{1-\mu} \log^3 n, 1 + 1/\operatorname{poly}(n))$-approximation of G. The graph obtained has $O(m + n^{1+\mu} \log n)$ edges and the algorithm uses $O(n^{1+2\mu} \log n)$ work and runs in $\Omega(\log^2 n \log^* n)$ time, We utilize this to improve on the tradeoffs stated in Theorem 2.8. We obtain the following theorem.

THEOREM 2.9. *For $0.5 \leq \mu \leq 1$ and fixed constants $0 \leq \delta \leq 1$ and $\epsilon > 0$, we can obtain (with probability $1 - O(1/\operatorname{poly}(n))$) paths with stretch*

$$(2\beta)^{\lceil (1-\mu)/\delta \rceil}(1+\epsilon)$$

between all pairs of vertices in a given set $Q \subset V \times V$ in $O(n^{\delta} \log n \beta^2)$ time using

$$O(n^{1+2\mu} \log n + n^{1+2/\beta}\beta^2 \log^4 n + (m + n^{1+\mu} \log n + |Q|)n^{1/\beta}\beta \log^2 n)$$

work.

Proof. We use a simple modification of the proof of Theorem 2.8 where we start with the graph G obtained by applying the algorithm of [6] with parameter μ.

3. Computing an $(\ell, \beta, \rho, \hat{\epsilon})$-cover. In this section we reduce the problem of finding an $(\ell, \beta, \rho, \hat{\epsilon})$-cover in a graph $G = (V, E)$ with weights $w : E \to R_+$ to a problem of finding a $(\beta, 2\ell\hat{\epsilon}^{-1})$-cover in (V, E) with respect to some integral weights $\bar{w} : E \to \mathcal{N}$. We define (β, W)-cover of a weighted graph to be an $(n, \beta, W, 0)$-cover of the graph (see Definition 2.3). The weights $\bar{w}$ are obtained by scaling and rounding up the weights w. The reduction is based on a technique and ideas from Klein and Sairam [10]. The weights $\bar{w}$ are computed as follows.

ALGORITHM 3.1 (generate integral weights $\bar{w}$).

1. $\pi \leftarrow 2\ell\hat{\epsilon}^{-1}/\rho$
2. *For all $e \in E$, let $\bar{w}(e) \leftarrow \lceil \pi w(e) \rceil$ $(e \in E)$*

Complexity. Using standard PRAM techniques, Algorithm 3.1 can be performed in $O(\log n)$ time using $O(m)$ work.

The weights $\bar{w} : E \to \mathcal{N}$ have the following properties.

PROPOSITION 3.2. *For every pair of vertices, $\{u_1, u_2\} \subset V$,*

1. $\pi \text{dist}^{\ell}_{E,w}\{u_1, u_2\} \geq \text{dist}_{E,\bar{w}}\{u_1, u_2\} - \ell$ *and*
2. $\pi \text{dist}_{E,w}\{u_1, u_2\} \leq \text{dist}_{E,\bar{w}}\{u_1, u_2\}$.

Proof. Observe that for every path p of size $k = |p|$,

$$\bar{w}(p) - k \leq \pi w(p) \leq \bar{w}(p) \ .$$

Part 2 is immediate from the right inequality. We apply the left inequality to prove part 1. Let $\hat{p}$ be the path that minimizes $w(\hat{p})$ among all paths of size at most ℓ between u_1 and u_2. We have

$$\pi \text{dist}^{\ell}_{E,w}\{u_1, u_2\} = \pi w(\hat{p}) \geq \bar{w}(\hat{p}) - |p| \geq \text{dist}_{E,\bar{w}}\{u_1, u_2\} - \ell \ .$$

DEFINITION 3.3 ((β, W)-covers of graphs). *For a given graph $G = (V, E)$ with integral weights $w : E \to \mathcal{N}$, a scalar $W \geq 1$ and an integer $\beta = O(\log n)$, a (β, W)-cover of G is a collection of sets of vertices $X_1, \ldots, X_k$ (clusters) and vertices $v_1, \ldots, v_k$ where $v_i \in X_i$ (centers) such that*

1. *$\forall \{u, v\} \subset V$ such that $\text{dist}\{u, v\} \leq W$, $\exists i$ such that $\{u, v\} \subset X_i$,*
2. *$\forall i$ for all $u \in X_i$, $\text{dist}\{v_i, u\} \leq \beta W$, and*
3. *for every $v \in V$, $|\{i | v \in X_i\}| = O(n^{1/\beta} \beta \log n)$.*

In section 5 we present a randomized algorithm that computes a (β, W)-cover of an input graph G with integral edge weights. The algorithm runs in $O(W\beta^2 \log n)$ time using $O(n^{1/\beta} m/(W\beta))$ processors and computes a cover with probability $1 - O(1/\text{poly}(n))$.

We discuss computing an $(\ell, \beta, \rho, \hat{\epsilon})$-cover in a graph $G = (V, E)$ with weights $w : E \to R_+$. Consider applying Algorithm 3.1 to obtain the weights $\bar{w} : E \to \mathcal{N}$. Define $W = \pi\rho = 2\ell\hat{\epsilon}^{-1}$. Let $X_1, \ldots, X_k$ with corresponding centers $v_1, \ldots, v_k$ be a (β, W)-cover of (V, E) with respect to the weights $\bar{w}$.

PROPOSITION 3.4. *The sets $X_1, \ldots, X_k$ with corresponding centers $v_1, \ldots, v_k$ constitute an $(\ell, \beta, \rho, \hat{\epsilon})$-cover of G.*

Proof. We prove that condition 1 of Definition 2.3 holds for $X_1, \ldots, X_k$. Consider a pair of vertices $\{u_1, u_2\} \subset V$ such that $\text{dist}^{\ell}_{E,w}\{u_1, u_2\} \leq \rho/(1 + \hat{\epsilon})$. It follows from Proposition 3.2 that

$$\text{dist}_{E,\bar{w}}\{u_1, u_2\} \leq \pi \text{dist}^{\ell}_{E,w}\{u_1, u_2\} + \ell \leq W(1/(1 + \hat{\epsilon}) + \hat{\epsilon}/2) \leq W \ .$$

Hence, from condition 1 of Definition 3.3, for some i, $\{u_1, u_2\} \subset X_i$. We show that condition 2 of Definition 2.3 holds. Consider $1 \leq i \leq k$ and $u \in X_i$. It follows from Proposition 3.2 that $\pi \text{dist}_{E,w}\{v_i, u\} \leq \text{dist}_{E,\bar{w}}\{v_i, u\}$. Condition 2 of Definition 3.3 asserts that $\text{dist}_{E,\bar{w}}\{v_i, u\} \leq \beta W \leq \pi\beta\rho$. Hence, $\text{dist}_{E,w}\{v_i, u\} \leq \beta\rho$ and thus condition 2 holds. The validity of condition 3 is immediate from condition 3 of Definition 3.3.

Proposition 3.4 asserts that an $(\ell, \beta, \rho, \hat{\epsilon})$-cover of G with weights w can be computed within the same resource bounds as a $(\beta, 2\ell\hat{\epsilon}^{-1})$-cover of G with weights $\bar{w}$. The proof of Theorem 2.4 follows using the bounds for computing a (β, W)-cover stated above.

4. Computing neighborhoods. In this section we present a parallel algorithm for computing neighborhoods of vertices in a graph $G = (V, E)$ with integral weights $w : E \rightarrow \mathcal{N}$. The k-neighborhood of a vertex v, $N_k(v)$ is the set of vertices of distance at most k from v. The time depends linearly on k (the radius of the neighborhood) and the work performed is linear in $\mathcal{E}(N_k(v))$, the number of edges between vertices in the neighborhood.

We assume that the input is such that at each vertex, the incident edges are sorted into a list of "buckets" according to increasing weights. (Note that this can be achieved in $O(\log^2 n)$ time and either $O(m \log n)$ work using comparison-based sorting or $O(m)$ work, since weights are integral and for the purposes of this paper it suffices to consider weights that are of size at most $W = O(\hat{\epsilon}^{-1}n)$.) We prove the following.

PROPOSITION 4.1. *For a set of integers $k_1, \ldots, k_r$, and a set of vertices $s_1, \ldots, s_r$, the computation of $N_{k_i}(s_i)$ for $i = 1. \ldots, r$ can be performed using p processors in time*

$$O\left((\max_i k_i \log n) + \sum_{i=1}^{r} |\mathcal{E}(N_{k_i}(s_i))|(\log n)/p\right) .$$

Or, alternatively, for any fixed $\alpha > 0$, in

$$\text{time } \Omega(\max_i k_i n^{\alpha}) \text{ and work } O\left(\sum_{i=1}^{r} |\mathcal{E}(N_{k_i}(s_i))|\right) .$$

The algorithm essentially amounts to performing, in parallel for s_i $(1 \leq i \leq r)$ a weighted parallel BFS computation from s_i while considering only paths of weight at most k_i.

We present a weighted parallel BFS algorithm from a vertex s in G. The algorithm is a straightforward generalization of a parallelization of the standard sequential BFS algorithm used by Ullman and Yannakakis [14]. A weighted version of the parallel BFS algorithm was also used by Klein and Sairam [10].

The algorithm consists of iterations where in the ith iteration $(i \geq 0)$ the algorithm computes the set $\text{level}_i(s) \subset V$ of all vertices of distance i from s. The algorithm terminates when a stopping condition is met.

ALGORITHM 4.2 (weighted Parallel BFS from s).

1. $\text{level}_0(s) \leftarrow \{s\}$
   $F = \{s\}$
   $i \leftarrow 0$
2. **Repeat:**
   $i \leftarrow i + 1$
   $\text{level}_i(s) \leftarrow \{v \in V \setminus F | \exists j < i \exists u \in \text{level}_j(s) \text{ s.t. } (v, u) \in E \wedge w(v, u) = i - j\}$
   $F = F \cup \text{level}_i(s)$
   Until: *stopping condition is satisfied*

Correctness. It is easy to verify that after the ith iteration,

$$\text{level}_i(s) = \{u \in V | \text{dist}_E\{s, u\} = i\} .$$

Complexity. We outline an EREW PRAM implementation of the algorithm and analyze the resulting resource bounds.

We assume that at each vertex $u \in V$ there is a list of "buckets" containing a partition of the vertices adjacent to u. The partitioning of adjacent vertices to buckets is done by sorting the edges incident at u. Adjacent vertices are placed in buckets

according to the weight of the edge to the vertex u. The buckets are maintained in a list, according to increasing weights, such that each bucket has a pointer to the nonempty bucket of next larger weight. For a vertex u and integer j, we denote by $B(u,j)$ the bucket of adjacent vertices to u through edges of weight j. The algorithm also maintains a partition of the vertices in F as a list of sets A_j ($j > 0$). The partition is modified in the course of the algorithm. At the termination of iteration i, a vertex $u \in \text{level}_j(s)$ (for $j < i$) is placed in the set A_{c_u+j}, where c_u is the smallest weight of an edge incident at u such that $c_u + j > i$. Note that at iteration i it suffices to examine vertices in A_{i+1} to determine the $i+1$st level.

We specify the actions of the algorithm in the ith iteration. The algorithm examines (in parallel) the vertices in A_i. For each $u \in A_i \cap \text{level}_j(s)$, the algorithm scans the vertices in $B(u, i-j)$. For each $u' \in B(u, i-j)$ the algorithm does as follows. If $u' \notin F$, it is placed in $\text{level}_i(s)$. The vertex u' is placed in a set $A_{i'}$, where $i' = i + \min\{j' | B(u', j') \neq \emptyset\}$. The vertex u is replaced in a different set $A_{i''}$ ($i'' > i$), according to the weight of the next nonempty bucket on its bucket list $i'' = i + \min\{j' > i - j | B(u, j') \neq \emptyset\}$. It follows from standard EREW PRAM techniques that the ith iteration can be performed in $O(\log n)$ time using $\sum_{j<i} \sum_{u \in \text{level}_j(s)} |B(u, i-j)| \log n$ work or, alternatively, in $\Omega(n^\alpha)$ time using $\sum_{j<i} \sum_{u \in \text{level}_j(s)} |B(u, i-j)|$ work.

Consider performing instances of the above computation in parallel for a set of sources $s_1, \ldots, s_r$, where the computation for a source s_i is done for up to $k_i \leq \ell$ ($1 \leq i \leq r$) iterations. The ith iterations (for $0 \leq i \leq \ell$) of all instances are performed in parallel. Hence, with p available processors, the ith iterations of all instances can be performed in

$$O\left(\log n + \sum_{h=1}^{r} \sum_{j<i} \sum_{u \in \text{level}_j(s_h)} |B(u, i-j)|(\log n)/p\right)$$

time or alternatively in

$$O\left(n^\alpha + \sum_{h=1}^{r} \sum_{j<i} \sum_{u \in \text{level}_j(s_h)} |B(u, i-j)|/p\right)$$

time. For $1 \leq j \leq r$, denote by F_j the set F generated by the instance of the algorithm for the source s_j. Note that for all $1 \leq h \leq r$,

$$\sum_{i \leq r} \sum_{j<i} \sum_{u \in \text{level}_j(s_h)} |B(u, i-j)| = 2|\mathcal{E}(F_h)| \ .$$

It follows that the ℓ iteration of Algorithm 4.2 can be performed in time

$$O\left(\ell \log n + \sum_{s \in S} |\mathcal{E}(F_s)|(\log n)/p\right)$$

(or, alternatively, in time $O(\ell n^\alpha)$ and work $O(\sum_{s \in S} |\mathcal{E}(F_s)|/p)$). This concludes the proof of Proposition 4.1.

REMARK 4.3 (computing neighborhoods sequentially). *Consider a graph with weights $w : E \to R_+$, a scalar ρ, and a vertex s. The goal is to compute $N_{k\rho}(s)$ where*

k is the first j such that $N_{j\rho}(s)$ satisfies some desired easy-to-check property. Assume that G is represented in a way that the edges incident at each vertex can be accessed in order according to increasing weight. A simple modification of the sequential BFS algorithm can compute $N_{k\rho}(s)$ in $|\mathcal{E}(N_{k_i\rho}(s_i))|$ time. The algorithm uses $O(k+n)$ additional storage for maintaining a partition of the vertices into sets $A_j (1 \le j \le k)$. The latter partition is used in a similar fashion to the partition maintained by the parallel algorithm stated above.

5. Computing a (β, W)-cover. In this section we present and analyze an algorithm that computes a (β, W)-cover χ for an input graph $G = (V, E)$ with integral weights $w : E \to \mathcal{N}$.

DEFINITION 5.1. *For $V' \subset V$, $v \in V'$, and an integer $k \ge 0$, define*

1. $\mathrm{core}(v, k, V') = N_{kW}(V', v)$ *and*
2. $\mathrm{cluster}(v, k, V') = N_{(k+1)W}(V', v)$.

The following proposition states some easy-to-verify properties.

PROPOSITION 5.2.

1. *For $V' \subset V$, $k \in \mathcal{N}$, and $\{v_1, v_2\} \subset V'$: $v_1 \in \mathrm{core}(v_2, k, V')$ if and only if $v_2 \in \mathrm{core}(v_1, k, V')$. Similarly, $v_1 \in \mathrm{cluster}(v_2, k, V')$ if and only if $v_2 \in \mathrm{cluster}(v_1, k, V')$.*
2. *For $V' \subset V$, $k \in \mathcal{N}$, and $u \in \mathrm{cluster}(v, k, V')$, $\mathrm{dist}_E\{v, u\} \le \mathrm{dist}_{\mathcal{E}(V')}\{v, u\} \le (k+1)W$.*
3. *For integers $k_2 < k_1$, subsets $V_2 \subseteq V_1 \subseteq V$, and $v \in V_2$:*

$$\mathrm{core}(v, k_2, V_2) \subseteq \mathrm{cluster}(v, k_2, V_2) \subseteq \mathrm{core}(v, k_1, V_1) \subseteq \mathrm{cluster}(v, k_1, V_1) \ .$$

ALGORITHM 5.3 (compute a (β, W)-cover χ).

1. $V_1 \leftarrow V$, $\chi \leftarrow \emptyset$
2. *For $i = 1, \ldots, \beta$:*
   (a) *Choose uniformly at random a set $S_i \subset V_i$ of size $\lceil Cn^{i/\beta}|V_i|(\log n)/n \rceil$ ($C \ge 1$ is some constant, where the value of C determines the probability that the algorithm terminates correctly.)*
   If $|V_i| < \lceil Cn^{i/\beta}|V_i|(\log n)/n \rceil$ choose $S_i \leftarrow V_i$
   Let $k_i \leftarrow \beta - i$
   (b) *Perform, in parallel for $v \in S_i$, a computation of* $\mathrm{core}(v, k_i, V_i)$, $\mathrm{cluster}(v, k_i, V_i)$.
   (*Use the algorithm of section* 4.)
   (c) $\chi \leftarrow \chi \cup \{\mathrm{cluster}(v, k_i, V_i) | v \in S_i\}$
   (d) $V_{i+1} \leftarrow V_i \setminus \bigcup_{v \in S_i} \mathrm{core}(v, k_i, V_i)$
   If $V_{i+1} = \emptyset$, stop.

Figure 1 illustrates the clusters produced by the algorithm. In the first iteration, we select a small number of neighborhoods with a large radius. In following iterations, we select larger number of neighborhoods with smaller radiuses. The algorithm is such that with high probability these neighborhoods do not contain many vertices.

Note that the condition $|V_i| < \lceil Cn^{i/\beta}|V_i|(\log n)/n \rceil$ must occur for some i. When $|V_i| < \lceil Cn^{i/\beta}|V_i|(\log n)/n \rceil$, the current iteration is last. Since $S_i = V_i$, for all $v \in V_i$, $v \in \mathrm{core}(v, k_i, V_i)$ and hence $V_{i+1} = \emptyset$. Therefore, we have $V \equiv V_1 \supset V_2 \supset \cdots \supset V_t = \emptyset$, where $i = t - 1$ in the last iteration.

In the remaining part of this section we prove the following theorem.

THEOREM 5.4. *With probability $1 - O(1/\mathrm{poly}(n))$, Algorithm* 5.3 *is*

1. *correct, that is, χ constitutes a (β, W)-cover of G; and*

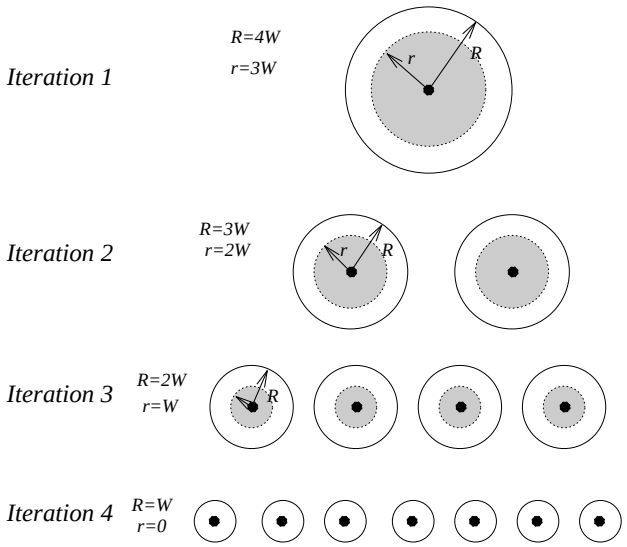


FIG. 1. *Clusters produced by the algorithm with* $\beta = 4$.

2. *can be implemented to run in* $O(W\beta^2 \log n)$ *time using* $O(n^{1/\beta}m/(W\beta))$ *processors; hence, performing* $O(n^{1/\beta}m\beta \log n)$ *work.*

PROPOSITION 5.5. *For every pair of vertices* $\{u_1, u_2\} \subset V$, *if* $\text{dist}_E\{u_1, u_2\} \leq W$, *then* $\exists X \in \chi$ *such that* $\{u_1, u_2\} \subset X$.

Proof. Let $\{u_1, u_2\} \subset V$ be such that $\text{dist}_E\{u_1, u_2\} \leq W$ and let p be a path in (V, E) of weight at most W between u_1 and u_2. During this proof we interchangeably refer to p as the path as a whole or as the set of vertices of p. Since $\{u_1, u_2\} \subset p$, it suffices to show that $p \subset X$ for some $X \in \chi$. Let j be such that $p \subset V_j$ but $p \not\subset V_{j+1}$. Note that j is well defined since the sets V_i $(i \geq 1)$ are monotonically decreasing and include the set of all vertices and the empty set. Denote by χ_j the assignment of χ at the end of the jth iteration. We show that $\exists X \in \chi_j$ such that $p \subset X$. Let $u \in p$ be such that $u \notin V_{j+1}$. It follows from the algorithm that $u \in \text{core}(s, k_j, V_j)$ for some $s \in S_j$. Hence, from the definition, $N_W(V_j, u) \subset \text{cluster}(s, k_j, V_j)$. Since p is of weight at most W and $p \subset V_j$, it follows that $p \subset N_W(V_j, u)$. Hence,

$$\{u_1, u_2\} \subset p \subset N_W(V_j, u) \subset \text{cluster}(s, k_j, V_j) \in \chi \ .$$

PROPOSITION 5.6. *For every* $v \in V$ *and* $1 \leq i \leq \beta$,

$$\text{core}(v, k_{i+1}, V_{i+1}) \subseteq \text{cluster}(v, k_{i+1}, V_{i+1}) \subseteq \text{core}(v, k_i, V_i) \subseteq \text{cluster}(v, k_i, V_i) \ .$$

Proof. The proof is immediate from part 3 of Proposition 5.2.

PROPOSITION 5.7. *With probability* $1 - O(1/\text{poly}(n))$, *for* $j > i$

$$\forall v \in V_j, |\text{cluster}(v, k_j, V_j)| < n^{1-i/\beta} \ .$$

Proof. If i is the last iteration, $V_{i+1} = \emptyset$ and the proposition follows. Otherwise, $|S_i| = \Omega(n^{i/\beta}|V_i|(\log n)/n)$. Since $\text{cluster}(v, k_j, V_j) \subset \text{core}(v, k_i, V_i)$ (see Proposition 5.6) and $V_j \subset V_i$,

$$\{v \in V_j \mid |\text{cluster}(v, k_j, V_j)| \geq n^{1-i/\beta}\} \subset \{v \in V_i \mid |\text{core}(v, k_i, V_i)| \geq n^{1-i/\beta}\} \ .$$

It is therefore sufficient to prove that with probability $1 - O(1/\operatorname{poly}(n))$,

$$\{v \in V_i| \ |\mathrm{core}(v, k_i, V_i)| \geq n^{1-i/\beta}\} \cap V_{i+1} = \emptyset \ .$$

Consider a vertex $v \in V_i$ such that $|\mathrm{core}(v, k_i, V_i)| \geq n^{1-i/\beta}$. We prove that with probability $1 - O(1/\operatorname{poly}(n))$, $v \in \bigcup_{s \in S_i} \mathrm{core}(s, k_i, V_i)$ (and, hence, using part 1 of Proposition 5.2, $v \notin V_{i+1}$). Since the elements of S_i are chosen uniformly at random, for $s \in S_i$ we have

$$\mathrm{Prob}\{v \in \mathrm{core}(s, k_i, V_i)\} = |\mathrm{core}(v, k_i, V_i)|/|V_i| \geq n^{1-i/\beta}/|V_i| \ .$$

Hence, since the elements of S_i are chosen independently,

$$\begin{aligned}&\mathrm{Prob}\{v \in V_{i+1}\} = \mathrm{Prob}\left\{v \notin \textstyle\bigcup_{s \in S_i} \mathrm{core}(s, k_i, V_i)\right\} \leq (1 - n^{1-i/\beta}/|V_i|)^{|S_i|} \\ &= (1 - n^{1-i/\beta}/|V_i|)^{\Omega(|V_i| n^{i/\beta - 1} \log n)} = O(1/\operatorname{poly}(n)) \ .\end{aligned}$$

To conclude the proof note that

$$\begin{aligned}&\mathrm{Prob}\left\{\{v \in V_i| \ |\mathrm{core}(v, k_i, V_i)| \geq n^{1-i/\beta}\} \cap V_{i+1} \neq \emptyset\right\} \\ &\leq \textstyle\sum_{v \in V_i} \mathrm{Prob}\{|\mathrm{core}(v, k_i, V_i)| \geq n^{1-i/\beta} \wedge v \in V_{i+1}\} \leq |V_i|/\operatorname{poly}(n) \leq O(1/\operatorname{poly}(n)) \ .\end{aligned}$$

PROPOSITION 5.8. *With probability* $1 - O(1/\operatorname{poly}(n))$, *for all* $e \in E$ *and all* $v \in V$,

$$\begin{aligned}|\{X \in \chi | e \in \mathcal{E}(X)\}| &= O(n^{1/\beta} \beta \log n) \ \textit{and} \\ |\{X \in \chi | v \in X\}| &= O(n^{1/\beta} \beta \log n) \ .\end{aligned}$$

Proof. The claim for edges follows from the claim for vertices. Consider the ith iteration and a vertex v. It follows from Proposition 5.7 that with probability $1 - O(1/\operatorname{poly}(n))$,

$$\text{for all } u \in V_i, \ \ |\mathrm{cluster}(u, k_i, V_i)| \leq n^{1-(i-1)/\beta} \ .$$

Using part 1 of Proposition 5.2 we have that for $u \in V_i$, $v \in \mathrm{cluster}(u, k_i, V_i)$ if and only if $u \in \mathrm{cluster}(v, k_i, V_i)$. Hence, for any $u \in V_i$ chosen uniformly at random

$$\mathrm{Prob}\{v \in \mathrm{cluster}(u, k_i, V_i)\} \leq |\mathrm{cluster}(v, k_i, V_i)|/|V_i| \leq n^{1-(i-1)/\beta}/|V_i| \ .$$

Therefore, the expected number of vertices $s \in S_i$ for which $v \in \mathrm{cluster}(s, k_i, V_i)$ is bounded by $|S_i| n^{1-(i-1)/\beta}/|V_i| = O(n^{1/\beta} \log n)$. From Chernoff bound [5] we obtain

$$\mathrm{Prob}\{|\{s \in S_i | v \in \mathrm{cluster}(s, k_i, V_i)\}| = O(n^{1/\beta} \log n)\} \geq 1 - O(1/\operatorname{poly}(n)) \ .$$

Since $\beta = O(\log n) = O(\operatorname{poly} n)$, it follows that for all $v \in V$,

$$\mathrm{Prob}\left\{\sum_{i=1}^{\beta} |\{s \in S_i | v \in \mathrm{cluster}(s, k_i, V_i)\}| = O(\beta n^{1/\beta} \log n)\right\} \geq 1 - O(1/\operatorname{poly}(n)) \ .$$

Since $|V| = O(n)$, we can deduce that with probability $1 - O(1/\operatorname{poly}(n))$,

$$\forall v \in V, \sum_i |\{s \in S_i | v \in \mathrm{cluster}(s, k_i, V_i)\}| = O(n^{1/\beta} \beta \log n) \ .$$

COROLLARY 5.9. *With probability* $1 - O(1/\operatorname{poly}(n))$,

$$\sum_{X \in \chi} |\mathcal{E}(X)| = O(n^{1/\beta} m\beta \log n) \text{ and}$$
$$\sum_{X \in \chi} |X| = O(n^{1+1/\beta}\beta \log n) .$$

Proof. Note that

$$\sum_{X \in \chi} |\mathcal{E}(X)| = \sum_{e \in E} |\{X \in \chi | e \in X\}| \text{ and } \sum_{X \in \chi} |X| = \sum_{v \in V} |\{X \in \chi | v \in X\}| .$$

The proof follows using Proposition 5.8.

We conclude the proof of Theorem 5.4. To prove the correctness of Algorithm 5.3 we verify that with probability $1 - O(1/\operatorname{poly}(n))$, χ constitutes a (β, W)-cover of G. We show that χ satisfies properties 1 and 2 of Definition 3.3, and with probability $1 - O(1/\operatorname{poly}(n))$, χ satisfies property 3 of the definition. Proposition 5.5 establishes that property 1 holds for χ. The validity of property 2 follows from the fact that for all i, $k_i \leq \beta - 1$. Proposition 5.8 asserts that χ satisfies property 3 (with probability $1 - O(1/\operatorname{poly}(n))$).

We furnish resource bounds for Algorithm 5.3. The computation of clusters in step 2b assumes that the graph is represented in a certain format (see section 4). This representation allowed us to compute clusters with work proportional to the combined size of the clusters computed (instead of having to look at all the edges in the graph). We remark that this representation can be easily generated and revised (for the subgraphs induced by V_i where $i > 0$) within the resource bounds of the algorithm. Consider a single iteration of Algorithm 5.3. The computation in step 2b dominates the complexity. It follows from Proposition 4.1 that the computation of step 2b in the ith iteration can be performed in

$$O\left(W\beta \log n + \sum_{s \in S_i} |\mathcal{E}(\text{cluster}(s, k_i, V_i))|(\log n)/p\right)$$

time using p processors (and in $O(\sum_{s \in S_i} |\mathcal{E}(\text{cluster}(s, k_i, V_i))|)$ work, when allowing $O(W\beta n^{\alpha})$ time). Hence, since there are at most β iterations, the algorithm can be performed in

$$O\left(W\beta^2 \log n + \sum_{x \in \chi} |\mathcal{E}(X)|(\log n)/p\right)$$

time using p processors. By substituting the bound from Corollary 5.9 we deduce that with probability $1 - O(1/\operatorname{poly}(n))$, the algorithm can be performed in $O(W\beta^2 \log n)$ time using $p = O(n^{1/\beta} m(\log n)/(W\beta))$ processors. The work performed by the algorithm is $O(n^{1/\beta} m\beta \log^2 n)$. When allowing $O(W\beta^2 n^{\alpha})$ time (for fixed α), the work is reduced to $O(n^{1/\beta} m\beta \log n)$.

6. Sequential algorithms. This section is concerned with sequential stretch-t paths algorithms. We first discuss a sequential version of the randomized algorithm presented in previous sections. In the main part of this section we present a deterministic algorithm. Our sequential algorithms are both simpler than the previous Awerbuch et al. [2] algorithm and significantly improve the tradeoffs between the stretch and the running time.

6.1. A randomized sequential algorithm. In subsection 2.2 we presented a parallel algorithm (Algorithm 2.6) that computes stretched ℓ-limited distances, where $1 \leq \ell \leq n$. Hence, when $\ell = n$, the algorithm produces stretch-t paths. The resulting algorithm inputs a graph G, with weights $w : E \rightarrow R_+$, integer $\beta \geq 1$, a scalar $0 \leq \hat{\epsilon} \leq 1/2$, and a set of query pairs $Q \subset V \times V$. The algorithm runs in time $O((m + |Q|)n^{1/\beta}\beta \log^2 n)$ and with probability $1 - O(1/\operatorname{poly}(n))$ computes for all $\{v, u\} \in Q$, stretched distance $\tilde{\operatorname{dist}}\{v, u\}$ such that

$$\operatorname{dist}_E\{v, u\} \leq \tilde{\operatorname{dist}}\{v, u\} \leq 2\beta(1 + \hat{\epsilon})^2 \operatorname{dist}_E\{v, u\} \ .$$

A simplified sequential algorithm with slightly worse time bounds and slightly better stretch can be obtained by avoiding the reduction to small integral weights performed in section 3. Small integral weights are necessary for efficient parallel computation of neighborhoods but are not crucial in the sequential version. The simplified algorithm applies a sequential version of Algorithm 2.6 and uses $(n, \beta, w_i, 0)$-covers instead of $(\ell, \beta, w_i, \hat{\epsilon})$-covers (for $i = 1, \ldots, r$). $(n, \beta, w_i, 0)$-covers can be computed directly by Algorithm 5.3. In the implementation of Algorithm 5.3 we disregard the assumption that the weights are integral and utilize a slightly modified version of Dijkstra's algorithm to compute neighborhoods. If edges are initially sorted according to their weight at the vertices, we can compute a neighborhood with V' vertices in time $O(|\mathcal{E}(V')| + |V'| \log |V'|)$ (using Fibonnaci heaps). The simplified algorithm inputs a graph G, with weights $w : E \rightarrow R_+$, integer $\beta \geq 1$, a scalar $0 \leq \hat{\epsilon} \leq 1/2$, and a set of query pairs $Q \subset V \times V$. The algorithm runs in time $O((m + n \log n + |Q|)n^{1/\beta}\beta \log^2 n)$ and with probability $1 - O(1/\operatorname{poly}(n))$ computes for all $\{v, u\} \in Q$, stretched distance $\tilde{\operatorname{dist}}\{v, u\}$ such that

$$\operatorname{dist}_E\{v, u\} \leq \tilde{\operatorname{dist}}\{v, u\} \leq 2\beta(1 + \hat{\epsilon}) \operatorname{dist}_E\{v, u\} \ .$$

6.2. The deterministic algorithm. We present a deterministic stretch-t paths algorithm. For a weighted graph $G = (V, E)$ and a set of sources $S \subset V$, the algorithm computes stretched distances for all pairs of vertices in $S \times V$. For an integer β and any fixed $\epsilon > 0$, the algorithm obtains stretched distances to within a factor of $2(1 + \log_n m)\beta(1 + \epsilon)$ using $\tilde{O}(n^{1/\beta}(m + n|S|))$ time. For the all-pairs problem (when $|S| = n$), the running time is $\tilde{O}(n^{2+1/\beta})$. A modification of the algorithm achieves stretch $4\beta(1 + \epsilon)$ using $\tilde{O}(n^{2+1/\beta})$ time. For dense graphs $\log_n m \approx 2$, and hence the modification yields improved stretch (by a factor of $2/3$) within comparable time bounds. In contrast, the Awerbuch et al. [2] algorithm obtains stretch 32β using $\tilde{O}(n^{2/\beta}m + n^{1+1/\beta}|S|)$ time. We remark, however, that the Awerbuch et al. [2] algorithm is more general since stretched distances between any k specified pairs of vertices can be obtained in $\tilde{O}(n^{2/\beta}m + n^{1/\beta}k)$ time.

We use the following definition for pairwise covers. Note that it differs from Definition 2.3 that was used for the randomized algorithm.

DEFINITION 6.1. *Consider a graph $G = (V, E)$ with weights $w : E \rightarrow R_+$, an integer $\beta = O(\log n)$, and a scalar $\rho \in R_+$. A* pairwise (β, ρ)-cover *of G is a collection*

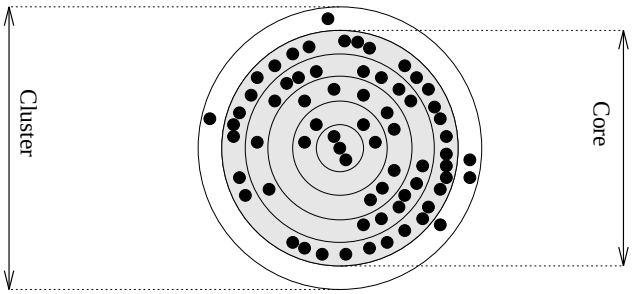


FIG. 2. *Example of a cluster and respective core for* $n^{1/\beta} < 2$.

of sets of vertices $X_1, \ldots, X_k$ (clusters) *and vertices* $v_1, \ldots, v_k$, *where* $v_i \in X_i$ *is the* center *of* X_i, *such that*

1. $\forall \{u, v\} \subset V$ *such that* $\text{dist}\{u, v\} \leq \rho$ $\exists i$ *such that* $\{u, v\} \subset X_i$,
2. $\forall i$, $\forall u \in X_i$, $\text{dist}\{v_i, u\} \leq (1 + \log_n m)\beta\rho$, *and*
3. $\sum_{i=1}^{k} |X_i| = O(n^{1+1/\beta})$.

We claim that a (β, ρ)-cover of G can be computed in $O(mn^{1/\beta})$ time. We first discuss the structure of the clusters we choose to constitute the cover.

6.3. Structure of clusters. We consider a graph $G = (V, E)$ with weights $w : E \to R_+$ and parameters β and $\rho \in R_+$. We discuss the structure and properties of subsets of vertices we call *clusters*. We use carefully chosen clusters as the building blocks of a (β, ρ)-cover.

A cluster is a subset of vertices that is defined with respect to two parameters: a subset of vertices $V' \subset V$ and a vertex $v \in V'$ (the *center* of the cluster). The cluster comprises a neighborhood of the center v in the graph $(V', \mathcal{E}(V'))$ up to a certain distance (that is an integral multiple of ρ). The *core* of a cluster is the subset of cluster vertices whose neighborhoods up to distance ρ in $(V', \mathcal{E}(V'))$ are contained in the cluster. For the parameters (v, V'), we define $\text{core}(v, V')$ and $\text{cluster}(v, V')$ to be the sets obtained by performing the following computation.

1. $\text{cluster}(v) \leftarrow \{v\}$
2. $i \leftarrow 1$
3. **Repeat:**
   (a) $\text{core}(v) \leftarrow \text{cluster}(v)$
   (b) $\text{cluster}(v) \leftarrow \{u \in V' | \text{dist}_{\mathcal{E}(V')}\{v, u\} \leq i\rho\}$
   (c) $i \leftarrow i + 1$
4. **Until:** $|\text{cluster}(v)| \leq n^{1/\beta}(|\text{core}(v)| + 1) \wedge$
   $|\mathcal{E}(\text{cluster}(v))| \leq n^{1/\beta}(|\mathcal{E}(\text{core}(v))| + 1)$

Figure 2 illustrates a cluster. The shells correspond to neighborhoods of the center that are integral multiples of ρ. In the example, $n^{1/\beta} < 2$. Hence the cluster stops growing when annexing the next shell does not double the number of vertices. The external shell defines the cluster, and the previous shell defines the core.

PROPOSITION 6.2. *At termination,* $i \leq (1 + \log_n m)\beta - 1$.

Proof. Note that the size of at least one of $\text{cluster}(v)$ or $\mathcal{E}(\text{cluster}(v))$ increases by more than a factor of $n^{1/\beta}$ in every iteration except for the last one. Observe that $|\text{cluster}(v)| \leq |V'| \leq n$ and $|\mathcal{E}(\text{core})| \leq m$. Thus, $|\text{cluster}(v)|$ can increase by a factor of more than $n^{1/\beta}$ at most $\beta - 1$ times and $|\mathcal{E}(\text{cluster}(v))|$ can increase by a factor

of more than $n^{1/\beta}$ at most $\beta \log_n m - 1$ times. Hence, the stopping condition occurs within $(1 + \log_n m)\beta - 1$ iterations.

Consider a subset $V' \subset V$ and $v \in V'$.

COROLLARY 6.3. *For all* $u \in \text{cluster}(v, V')$,

$$\text{dist}_E\{u, v\} \leq \text{dist}_{\mathcal{E}(V')}\{u, v\} \leq (1 + \log_n m)\beta\rho \ .$$

The following proposition is immediate.

PROPOSITION 6.4.

1. *For all* $u \in \text{core}(v, V')$, $N_W(V', u) \subset \text{cluster}(v, V')$.
2. $|\text{cluster}(v, V')| \leq n^{1/\beta}(|\text{core}(v, V')| + 1)$.
3. $|\mathcal{E}(\text{cluster}(v, V'))| \leq n^{1/\beta}(|\mathcal{E}(\text{core}(v, V'))| + 1)$.

We discuss the construction of clusters. We assume that the edges incident at each vertex are initially sorted according to their weight. Remark 4.3 asserts that a simple modification of the standard BFS algorithm yields the following bound: for all $V' \subset V$ and $v \in V'$, the sets $\text{cluster}(v, V')$ and $\text{core}(v, V')$ can be computed in $O(|\mathcal{E}(\text{cluster}(v, V'))|)$ time.

6.4. The cover algorithm. The following algorithm produces a pairwise cover.

ALGORITHM 6.5 (compute a (β, ρ)-cover).

1. $V' \leftarrow V$, $\chi \leftarrow \emptyset$

- **Main Loop:**

2. *Choose* $v \in V'$
   Compute $\text{cluster}(v, V')$ *and* $\text{core}(v, V')$.
   $\chi \leftarrow \chi \cup \{\text{cluster}(v, V')\}$
   $V' \leftarrow V' \setminus \text{core}(v, V')$
   If $V' \neq \emptyset$, *go to step* 2

We show that χ is indeed a (β, ρ)-cover as in Definition 6.1. Property 2 is immediate from Corollary 6.3. Property 3 follows from Proposition 6.4 and the fact that the cores of the clusters in χ are disjoint. In order to prove that property 1 holds, consider a pair of vertices $\{v_1, v_2\} \subset V$ such that $\text{dist}_E\{v_1, v_2\} \leq \rho$. Let p be a set of vertices constituting a path between v_1 and v_2 of weight at most ρ. Consider the first iteration where $p \cap \text{core}(v, V') \neq \emptyset$ (V' and v denote the respective assignments of values to V' and v during this iteration). It follows that $p \subset V'$. By definition, $\text{cluster}(v, V')$ contains all vertices of distance at most ρ from $\text{core}(v, V')$ in $\mathcal{E}(V')$. Hence, since $\mathcal{E}(p) \subset \mathcal{E}(V')$ and $p \cap \text{core}(v, V') \neq \emptyset$, we have $p \subset \text{cluster}(v, V')$. Therefore, $\{v_1, v_2\} \subset p \subset \text{cluster}(v, V')$. The running time of the algorithm is dominated by the construction of the clusters. We assume that at each vertex the incident edges are sorted according to weight. We argued above that the computation of a cluster $X \in \chi$ can be performed in $O(|\mathcal{E}(X)|)$ time. Hence, using property 3 of Definition 6.1, we obtain the following bound on the running time

$$O\left(\sum_{X \in \chi} |\mathcal{E}(X)|\right) = O(n^{1/\beta} m) \ .$$

6.5. Computing stretched distances. A cover as in Definition 6.1 can be used in a similar manner to Algorithm 2.6 of subsection 2.2 to compute stretched distances from a set of sources $S \subset V$ to all other vertices.

THEOREM 6.6. *For a weighted graph* G, *an integer* $\beta \geq 1$, *a fixed* $\epsilon \geq 0$, *and a set of query sources* $S \subset V$, *the following algorithm computes for all* $v \in S$ *and for*

all $u \in V$, stretched distance $\tilde{\text{dist}}(v,u)$ such that

$$\text{dist}_E(v,u) \le \tilde{\text{dist}}(v,u) \le 2\beta(1+\log_n m)(1+\epsilon)\text{dist}_E(v,u)$$

in $O(n^{1/\beta}m\log n + |S|n^{1+1/\beta}\log n)$ time.

ALGORITHM 6.7 (compute distances).

1. $w_{\min} \leftarrow \min_{e\in E} w(e)$
   $w_{\max} \leftarrow \max_{e\in E} w(e)$
   $r \leftarrow \lceil \log_{1+\hat{\epsilon}}(nw_{\max}/w_{\min})\rceil$
   For $i = 0,\ldots,r$: $w_i \leftarrow w_{\min}(1+\hat{\epsilon})^i$
2. *For $i = 0,\ldots,r$ do (in parallel):*
   (a) *Compute (β, w_i)-cover χ_i*
   (b) *For each vertex $v \in S$, $x(i,v) \leftarrow \bigcup\{X \in \chi_i | v \in X\}$.*
3. *For all $v \in S$:*
   For all $u \in V$:
   $\tilde{\text{dist}}(v,u) \leftarrow 2(1+\log_n m)\beta w_i$, *where* $i \leftarrow \min\{j | u \in x(j,v)\}$.

The correctness follows from arguments similar to the ones given in subsection 2.2. The computation amounts to first computing $r = O(\epsilon^{-1}\log n)$ covers for different values of ρ and, second, for every source $v \in S$ computing $x(j,v)$ for $1 \le j \le r$. The computation per source can be performed in time linear in the sum of the sizes of the covers. Recall that the computation of covers using Algorithm 6.5 assumes that the edges are sorted according to weight at vertices. Hence, the running time is $O(m\log n)$ for the sorting of edges, $O(\epsilon^{-1}n^{1/\beta}m\log n)$ for computing the covers, and $O(n^{1+1/\beta}\epsilon^{-1}\log n)$ time per source.

REMARK 6.8. *Consider relaxing the termination conditions in the definition of clusters and not requiring that $|\mathcal{E}(\text{cluster}(v))| \le n^{1/\beta}(|\mathcal{E}(\text{core}(v))|+1)$. We obtain improved stretch (the diameter of cluster is shorter by a factor of $(1+\log_n m)$ but has worse time bounds). The tradeoffs for stretched all-pairs distances in dense graphs improve. We obtain an $O(n^{2+2/\beta}\log n)$ time algorithm for computing stretched distances to within a factor of 2β between all pairs of vertices.*

7. Applications to graph spanners. Consider a weighted graph $G = (V,E)$. A t-spanner of G is a weighted graph (V,E') such that for all pairs $\{u_1,u_2\} \subset V$ we have

$$\text{dist}_E\{u_1,u_2\} \le \text{dist}_{E'}\{u_1,u_2\} \le t\text{dist}_E\{u_1,u_2\}\ .$$

We utilize our constructions of pairwise covers to obtain t-spanners for weighted graphs. We present a randomized construction of spanners of size (number of edges) $O(n^{1+(2+\epsilon)/t})$ in time $O(mn^{(2+\epsilon)/t})$ (for any $\epsilon > 0$ and t such that $t/(2+\epsilon)$ is integral). We give a deterministic algorithm that finds t-spanners of size $O(n^{1+(2+\epsilon)(1+\log_n m)/t})$ and runs in $O(mn^{(2+\epsilon)(1+\log_n m)/t})$ time (for any $\epsilon > 0$ and t such that $t/((1+\log_n m)(2+\epsilon))$ is integral). In addition, we discuss parallel constructions of spanners.

7.1. Computing spanners sequentially. We show how to obtain sparse (small size) spanners. Our spanners are constructed from a collection of pairwise covers (see Definitions 3.3 and 6.1). Let $R = w_{\max}/w_{\min}$. Using considerations similar to Remark 2.1, we may assume that $R = O(\text{poly}(n))$. Let $r = \lceil \log_{1+\epsilon} R\rceil$. It follows that $r = O(\epsilon^{-1}\log n)$. Let $\beta = t/(2+\epsilon/2)$ if the randomized construction of covers is used (see Definition 3.3) and let $\beta = t/((1+\log_n m)(2+\epsilon/2))$ if the deterministic construction is used (see Definition 6.1). For each $0 \le i \le r$, let χ_i be a (β, w_i)-cover of

G , where $w_i = w_{min}(1+\epsilon)^i$. The t-spanner $S \subset E$ is constructed as follows. For each $0 \leq i \leq r$ and each cluster $X \in \chi_i$, add to S the edges of the partial BFS tree used to construct X. It follows immediately from the properties of our construction of covers that for every edge $e \in E$ there is a path in S of weight at most $tw(e)$ connecting the end points of e. Hence, for every path $p \subset E$, there is a path in S of weight at most $tw(p)$. Therefore, S indeed comprises a t-spanner. To bound the size of the spanner S, note that the number of edges in S is bounded by $\sum_{i=0}^{r} \sum_{X \in \chi_i} (|X| - 1)$. Hence, it follows from the properties of covers that $|S| = O(rn^{1+1/\beta}\beta \log n)$. To obtain a time bound for the spanner algorithm, note that it amounts to computing r covers. The time bounds for computing a single cover are $O(mn^{1/\beta})$ time for the deterministic cover algorithm and $O(mn^{1/\beta} \log n)$ time for the randomized cover algorithm.

Additional properties of our spanners. The spanner S consists of a collection of subtrees $\mathcal{T}$ of the graph G. The following stronger statement holds: for any two vertices $\{u_1, u_2\} \subset V$, the minimum over all the trees $T \in \mathcal{T}$ such that $\{u_1, u_2\} \subset T$ of the distance between u_1 and u_2 in T, is within a factor t of $\text{dist}_E\{u_1, u_2\}$. Furthermore, if the covers produced by the randomized construction are used, then every vertex is contained in at most $O(rn^{1/\beta}\beta \log n)$ subtrees in $\mathcal{T}$. Consider a variant where we do not require the spanner S to consist of edges from E, but we want limited distances on the spanner (using small size paths) to be within a small factor of distances on the graph. Construct a spanner S' as follows. Take $R = nw_{\max}/w_{\min}$ and $r = \lceil \log_{1+\epsilon} R \rceil = O(\epsilon^{-1} \log n)$. For each $0 \leq i \leq r$, let χ_i be a (β, w_i)-cover of G, where $w_i = w_{min}(1+\epsilon)^i$. For each $0 \leq i \leq r$, each cluster $X \in \chi_i$ with respective center $u(X)$, and each $v \in X$, include in S' an edge $(u(X), v)$ of weight equal to the distance from $u(X)$ to v in X. The spanner S' has a stronger property that the 2-limited distances in S' are within a factor t of the respective distances in G.

7.2. Computing spanners in parallel. We provide three methods for computing spanners in parallel. In this subsection we assume that $\beta = t/(2 + \epsilon/2)$ and use Definition 3.3 for covers.

Straightforward parallel implementation. We describe a parallel implementation of the sequential algorithm above that has the same work bound and runs in time linearly dependent on $R = w_{\max}/w_{\min}$. Hence, this implementation is of interest when R is sufficiently small. In particular, the algorithm is in $\mathcal{NC}$ for unweighted graphs. Note that for the covers computed by the spanner algorithm we have $w_{\min} \leq w_i \leq (1+\epsilon)w_{\max}$ $(0 \leq i \leq r)$. By definition, for any cluster $X \in \chi_i$, the distance from the center of X to any vertex $v \in X$ is at most βw_i, and, hence, the partial BFS tree generated while computing X has path size of $O(R\beta)$. Consider an application of Algorithm 5.3 to compute χ_i, where in step 2b we compute clusters by a weighted parallel BFS algorithm (see section 4). It follows that the cover algorithm runs in expected time $O(R\beta^2 \log n)$ and performs $O(n^{1/\beta} m\beta \log n)$ work. Hence, t-spanners of size $O(n^{1+(2+\epsilon)/t})$ can be obtained in $O(R\beta^2 \log^2 n)$ expected time using $O(n^{1/\beta} m\beta \log^2 n)$ work.

Using (exact) all-pairs shortest paths. We showed above that computing a spanner amounts to concurrently computing $r = O(\log n)$ covers. Consider a single application of Algorithm 5.3 to compute a cover. The complexity of the algorithm is dominated by step 2b. Note that step 2b can be performed (with worse work bounds) as follows. First compute all-pairs distances on the graph induced by the vertices V_i. The sets $\text{core}(v, k_i, V_i)$ and $\text{cluster}(v, k_i, V_i)$ for $v \in S_i$ can be easily generated using the all-pairs distances table. The parallel all-pairs shortest-paths computation can be performed in $O(\log^2 n)$ time with $O(n^3)$ work on an EREW PRAM. The generation of the sets $\text{core}(v, k_i, V_i)$ and $\text{cluster}(v, k_i, V_i)$ for $v \in S_i$ can be performed within the same time

bounds using work proportional to the sum of the sizes of these sets. Note that the resource bounds of the all-pairs shortest paths computation dominate all other steps. A shortest-paths computation is performed in each one of the β iterations of Algorithm 5.3. It follows that a pairwise cover can be computed in $O(\beta \log^2 n)$ time using $O(\beta n^3)$ work. Hence, t-spanners of size $O(n^{1+(2+\epsilon)/t})$ can be obtained in $O(\beta \log^3 n)$ expected time using $O(n^3 \beta \log n)$ work. We remark that although the work bound is worse than the sequential time bound achieved above, it is better than even the sequential time bounds of previous algorithms.

Tradeoffs between time and stretch. This method is based on the same technique used to obtain a fast parallel stretch-t paths algorithm (see subsection 2.3). We sketch the algorithm. Consider some parameter $\ell' < R$, and let $k = \lceil \log_{\ell'/2} R \rceil$. We repeatedly apply k times a version of Algorithm 2.7, with parameter ℓ', where in step 1 we use $r \leftarrow \lceil \log_{1+\hat{\epsilon}} R \rceil$. The above guarantees that if $\mathrm{dist}_E\{u_1, u_2\} \leq w_{\max}$ then for all $\ell' \leq \ell$,

$$\mathrm{dist}_{E'}^{2\lceil \ell/\ell' \rceil}\{u_1, u_2\} \leq 2\beta(1+\hat{\epsilon})^2 \mathrm{dist}_E^{\ell}\{u_1, u_2\} \ ,$$

where E' is the set of edges obtained by applying the algorithm to (V, E). Note that the running time is $\tilde{O}(k\ell')$. We consider the BFS trees obtained by the covers computed at the last repetition of the algorithm and "expand" them back to consist of original edges. The resulting spanners are t^k-spanners, but their size and the work performed is the same as for the sequential t-spanner algorithm.

8. Stretch t paths on dynamic networks. The dynamic shortest-paths problem amounts to resolving on-line distance queries between pairs of vertices when weighted edges (and possibly new vertices) are inserted to or deleted from the graph. Unfortunately, even when only insertions are permitted, the currently best-known solution amounts to the brute force method of applying Dijkstra's $O(m + n \log n)$ time algorithm to resolve each distance query. The pairwise covers and stretch-t paths algorithms introduced here enable us to obtain more efficient algorithms for dynamic stretch-t distances. We consider a dynamic stretch-t paths problem where we are allowed to insert edges (while possibly introducing new vertices), delete recently inserted edges, and perform stretch-t distance queries. Another relevant setting is when some large graph is fixed, but we allow insertions to and deletions from a small set of new edges. The distance queries are performed with respect to the augmented graph.

Consider the sequential version of Algorithm 2.6 (see subsection 6.1). In the first part of the algorithm, it produces a set of pairwise covers as to be able to answer on-line distance queries. The first part takes $\tilde{O}(mn^{(2+\epsilon)/t})$ time. Consequently, in the second part of the algorithm, each distance query takes $\tilde{O}(n^{(2+\epsilon)/t})$ time. Our dynamic algorithms utilize Algorithm 2.6. Note that since Algorithm 2.6 is randomized, the time bounds mentioned below are for the expected running time.

We first consider the following problem: the graph G is fixed, and there is a dynamic set of new weighted edges I. Some of the edges in I may introduce new vertices. We denote by $V(I)$ the vertices in I. The operations allowed are to insert edges to I, delete edges from I, and ask stretch-t distance queries between vertices in $V \cup V(I)$, with respect to the graph $(V \cup V(I), E \cup I)$. We sketch an algorithm for this problem and analyze its performance. The initialization step is to apply the first part of Algorithm 2.6 to the fixed graph G and generate a collection of pairwise covers. The algorithm dynamically maintains a graph $G_I = (V_I, E_I)$ as follows. $V_I = V(I) \cup C$, where C is a collection of all centers of clusters that contain vertices from $V(I)$. The set E_I contains I and for each occurrence of a vertex $v \in V(I)$ in a cluster X with

center $c \in C$, E_I contains the edge (v, c) weighted by the radius of X. Recall (see Definition 2.3) that each vertex $v \in V$ belongs to $\tilde{O}(n^{(2+\epsilon)/t})$ clusters. Hence, $|C| = \tilde{O}(|V(I)|n^{(2+\epsilon)/t})$, and $E_I \setminus I = \tilde{O}(|V(I)|n^{(2+\epsilon)/t})$. It is easy to see that an insertion or deletion of an edge takes $O(1)$ time, if $V(I)$ is not modified, and $\tilde{O}(n^{(2+\epsilon)/t})$ time otherwise. To perform a stretch-t distance query for a pair $\{u_1, u_2\} \subset V \cup V(I)$, we proceed as follows. We utilize the second part of Algorithm 2.6, applied to the set of pairwise covers produced earlier, to compute stretch-t distances on G from u_1 and from u_2 to every vertex in $V(I) \cap V$. We perform a single-source shortest-paths computation, rooted at u_1, on G_I augmented as follows: by the vertices $\{u_1, u_2\}$ and edges between $\{u_1, u_2\}$ and $V(I) \cap V$, weighted by the stretch-t G-distances computed above. The answer to the query is the distance between u_1 and u_2 in the augmented G_I. It is easy to verify that the distance between u_1 and u_2 in the augmented G_I is within a factor of t of the distance in $(V \cup V(I), E \cup I)$. We bound the time required for performing a query. The stretch-t distances computations in G can be performed in time $\tilde{O}(|V(I) \cap V|n^{(2+\epsilon)/t})$. The single-source shortest-paths computation take time $\tilde{O}(|V(I)| + |E_I||) = \tilde{O}(|I| + |V(I)|n^{(2+\epsilon)/t})$ (using, e.g., Dijkstra's algorithm). Hence, the total query time is $\tilde{O}(|I|n^{(2+\epsilon)/t})$. We remark that the time is improved when $|V(I)| \ll |I|$.

Consider now the following problem. Let k be a parameter. The weighted edges are maintained in a list M. The allowed operations are to insert edges to the head of M, to delete edges that are of distance at most k from the head of M, and to ask t-stretch distance queries. We sketch an algorithm that operates in stages. Each stage amounts to an instance of the problem discussed in the previous paragraph. In the beginning of each stage, we consider a suffix of M denoted by M' that contains all edges in M except the initial $2k$ edges. The set M' constitutes the fixed graph, and we initialize $I = M \setminus M'$. Within a stage, the insertions, deletions, and distance queries are handles as sketched in the previous paragraph. The current stage terminates (and a new stage begins) when there is a request to delete an edge in M' or when $|M \setminus M'| \geq 4k$. It is easy to see that each stage consists of at least k deletions or insertions. Using standard dynamic algorithms methods, we can amortize the cost of the operations such that insertions or deletions take $\tilde{O}((k + m/k)n^{(2+\epsilon)/t})$ time and queries take $\tilde{O}(kn^{(2+\epsilon)/t})$ time. Note that when $k = m^{0.5}$, each operation takes $\tilde{O}(m^{0.5}n^{(2+\epsilon)/t})$ time. In general, k can be determined according to the ratio of edge insertions/deletions and queries.

9. Concluding remarks. We discuss some issues and open problems that arise from this work.

We remark that stretch-n paths can be obtained using a minimum spanning tree (MST). For a pair of vertices, consider the MST path p between them and the heaviest edge e on this path. It is easy to see that the distance between the vertices is bounded from below by $w(e)$ and bounded from above by $w(p) \leq nw(e)$. An MST can be computed sequentially in $O(m \log n)$ time and in parallel in $\mathcal{NC}$ using $\tilde{O}(m)$ work.

The techniques used in this paper and in the previous work of Awerbuch et al. [2] do not seem to generalize to directed graphs. A natural question regards obtaining comparable work-stretch tradeoffs for directed graphs. We conjecture that for directed graphs, obtaining stretch-t paths where t is constant or polylogarithmic is nearly as hard as finding shortest paths.

A result due to Karger, Koller, and Phillips [9] provides some indication that for directed graphs, either comparable time-stretch tradeoffs are not obtainable or stretch-t paths are indeed strictly easier to find than shortest paths. Karger, Koller, and Phillips proved a lower bound of $\Omega(mn)$ on the amount of work required by path-comparison-based algorithm for all-pairs shortest-paths computations. Note, however, that this lower bound is applicable for a restricted class of algorithms (that does not include the algorithms presented in this paper) and is applicable only to the all-pairs problem. Karger conjectured that the lower bound is likely to hold for undirected graphs as well [15].

We point out some additional properties of the pairwise covers constructed in this paper that were not needed for the applications used here. The first property of a (β, w)-cover stated that if two vertices $\{u_1, u_2\} \subset V$ have distance at most w, then at least one cluster in the cover contains all the vertices on some path of weight w between u_1 and u_2. It is easy to prove the stronger property that at least one cluster must contain *all* paths of weight at most w between u_1 and u_2. Moreover, the (β, w)-covers introduced here actually constitute neighborhood covers: for every vertex $u \in V$, the $w/2$ neighborhood of u must be contained in at least one cluster. Recall that pairwise covers were introduced here as a more correct (in the context of short paths computations) and constructible in parallel alternative for the neighborhood covers of Awerbuch and Peleg [3]. This indicates that our constructions may have other applications.

We presented efficient algorithms to produce sparse spanners. Some of the previous work on spanners (see Chandra et al. [4]) is concerned with finding spanners that are not only sparse but in addition have small weight (that is, within a small factor of the weight of the MST). It is likely that our methods can be extended to efficiently generate sparse spanners with small weight.

We presented parallel algorithms that compute stretch-t paths, where t is a constant or polylogarithmic, while performing significantly less work than exact shortest-paths algorithms. This is notable because, typically, parallel shortest-paths algorithms perform much more work than their sequential counterparts. A particular example is the single-source shortest-paths problem on graphs with nonnegative weights. It is solved sequentially in $\tilde{O}(m)$ time, but the currently best-known work bound of a polylog time algorithm is $O(n^3)$. We remark that the author, in a later work [7], that employs our parallel pairwise cover constructions, presented an $\tilde{O}(mn^{\epsilon_0} + s(m + n^{1+\epsilon_0}))$ work polylog-time randomized algorithm that computes paths within $(1 + O(1/\operatorname{polylog} n))$ of shortest path from s source nodes to all other nodes in weighted undirected networks (for any fixed $\epsilon_0 > 0$). Another implication of the work in [7] is efficiently parallelizing the randomized t-spanners, stretch-t paths, and pairwise covers algorithms presented in the current paper. Namely, obtaining polylogarithmic time parallel algorithms that perform work comparable to the respective sequential running times presented here and an additional additive term of $O(mn^{\epsilon_0})$ (where $\epsilon_0 > 0$ is any fixed constant).

Acknowledgments. The author would like to thank Lenore Cowen for presenting the Awerbuch et al. [2] work at Bell Labs and discussing it and the current work with the author, to thank Lenore Cowen, David Johnson, Alex Wang, Uri Zwick, and the anonymous referee for suggestions that improved the write up, and Andrew Goldberg, Monika Rauch, and Peter Shor for useful discussions and bibliographic pointers.

REFERENCES

[1] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares, *On sparse spanners of weighted graphs*, Discrete Comput. Geom., 9 (1993), pp. 81–100.

[2] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg, *Near-linear cost sequential and distributed constructions of sparse neighborhood covers*, in Proc. 34th IEEE Annual Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1993, pp. 638–647.

[3] B. Awerbuch and D. Peleg, *Sparse partitions*, in Proc. 31st IEEE Annual Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1990, pp. 503–513.

[4] B. Chandra, G. Das, G. Narasinhan, and J. Soares, *New sparseness results on graph spanners*, in Proc. 8th Annual ACM Symposium on Computational Geometry, Association for Computing Machinery, New York, 1992, pp. 192–201.

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

[6] E. Cohen, *Using selective path-doubling for parallel shortest-path computations*, in Proc. of the 2nd Israeli Symposium on the Theory of Computing and Systems, IEEE, Piscataway, NJ, 1993, pp. 78–87.

[7] E. Cohen, *Polylog-time and near-linear work approximation scheme for undirected shortest-paths*, in Proc. 26th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1994, pp. 16–26.

[8] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, McGraw–Hill, New York, 1990.

[9] D. R. Karger, D. Koller, and S. Phillips, *Finding the hidden path: Time bounds for all-pairs shortest paths*, in Proc. 32nd IEEE Annual Symposium on Foundations of Computer Science, IEEE, Piscataway, NJ, 1991, pp. 560–568.

[10] P. N. Klein and S. Sairam, *A parallel randomized approximation scheme for shortest paths*, in Proc. 24th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1992, pp. 750–758.

[11] D. Peleg and A. A. Schäffer, *Graph spanners*, J. Graph Theory, 13 (1989), pp. 99–116.

[12] D. Peleg and J. D. Ullman, *An optimal synchronizer for the hypercube*, SIAM J. Comput., 18 (1989), pp. 740–747.

[13] T. H. Spencer, *More time-work tradeoffs for parallel graph algorithms*, in Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, Association for Computing Machinery, New York, 1991, pp. 81–93.

[14] J. D. Ullman and M. Yannakakis, *High-probability parallel transitive closure algorithms*, SIAM J. Comput., 20 (1991), pp. 100–125.

[15] D. R. Karger, personal communication, 1993.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 237–253

SMART SMART BOUNDS FOR WEIGHTED RESPONSE TIME SCHEDULING*

UWE SCHWIEGELSHOHN†, WALTER LUDWIG‡, JOEL L. WOLF§, JOHN TUREK§, AND PHILIP S. YU§

Abstract. Consider a system of independent tasks to be scheduled without preemption on a parallel computer. For each task the number of processors required, the execution time, and a weight are known. The problem is to find a schedule with minimum weighted average response time. We present an algorithm called SMART (which stands for scheduling to minimize average response time) for this problem that produces solutions that are within a factor of 8.53 of optimal. To our knowledge this is the first polynomial-time algorithm for the minimum weighted average response time problem that achieves a constant bound. In addition, for the unweighted case (that is, where all the weights are unity) we describe a variant of SMART that produces solutions that are within a factor of 8 of optimal, improving upon the best known bound of 32 for this special case.

Key words. scheduling, parallel computing, approximate algorithms

AMS subject classifications. 68Q22, 68Q25

PII. S0097539795286831

1. Introduction.

Consider a parallel computer composed of P processors, on which a system τ consisting of M independent tasks is to be scheduled. Suppose that any given task $i \in \{1, \ldots, M\}$ requires $w_i \leq P$ processors for h_i units of time. All of the processors allotted to a task are required to execute that task in unison and without preemption. That is, w_i processors are all required to start task i at the same *starting time* t_i. They will then complete task i at *completion time* $t_i + h_i$. (One can therefore think of the execution of task i as taking place within a rectangle whose height h_i stretches along a *time* axis and whose width w_i stretches along a *processor* axis.) A *schedule* will consist of a starting time t_i for each task i and must be *legal* in the sense that for any time t the number of active processors does not exceed the total number of processors. In other words, $\sum_{\{i|t_i \leq t < t_i + h_i\}} w_i \leq P$ for all t. The classic problem of finding a schedule with minimum *makespan*, defined by $\max_{1 \leq i \leq M}\{t_i + h_i\}$, has been studied extensively in the literature. (The makespan corresponds to the last completion time and thus represents the length of the entire schedule.) The minimum makespan problem is NP-hard in the strong sense [5], and efforts have accordingly focused on finding polynomial-time algorithms whose solution in the worst case is within a fixed multiplicative constant of the optimal solution [4, 1, 3, 9].

In this paper we consider the corresponding problem, also NP-hard in the strong sense [2], where the goal is to minimize the *average response time* instead. The average response time can be written as $\frac{1}{M}\sum_{i=1}^{M}(t_i + h_i)$ and is an important and standard measure in computer performance. Note that *all* completion times count, not just the last completion time. Modulo the factor $\frac{1}{M}$ (which can be removed without affecting the solution to the problem), we are attempting to minimize the *sum of the completion times*, namely, $\sum_{i=1}^{M}(t_i + h_i)$.

* Received by the editors May 30, 1995; accepted for publication (in revised form) December 21, 1996; published electronically June 15, 1998.
http://www.siam.org/journals/sicomp/28-1/28683.html
† University of Dortmund, Dortmund, Germany (uwe@carla.e-technik.uni-dortmund.de).
‡ Computer Science Department, University of Wisconsin, Madison, WI.
§ IBM T.J. Watson Research Center, Yorktown Heights, NY 10598 (jlw@watson.ibm.com, jjt@watson.ibm.com, psyu@watson.ibm.com).

Our approach is analogous to that of the minimum makespan problem literature: we give an algorithm that takes polynomial time and produces schedules whose completion time sums are within a multiplicative constant of optimal. At this time, the best known constant is achieved by an algorithm called SMART [12]. Unfortunately, this constant (32) is not nearly as close to 1 as the best known constant (2) for the minimum makespan problem [4].

In this paper we devise a variant of the original SMART algorithm with an improved bound of 8. (It should be noted that the results of this paper can also be used to improve the bound of the original algorithm to 9.)

We also tackle the more general minimum *weighted* response time problem. That is, we assume that each task i is given a weight u_i, and that the objective is to minimize the weighted average response time $\frac{1}{C}\sum_{i=1}^{M} u_i(t_i + h_i)$, where $C = \sum_{i=1}^{M} u_i$. (Again the factor of $\frac{1}{C}$ can be removed without affecting the solution, so we speak instead of minimizing the weighted sum of the completion times, given by $\sum_{i=1}^{M} u_i(t_i + h_i)$.) When all weights are unity this problem reduces to the unweighted minimum response time special case described above.

We present a generalization of the original SMART algorithm for the minimum weighted response time problem. We show that this algorithm achieves a bound of 8.53. To our knowledge this is the first polynomial-time algorithm for the weighted problem that achieves a constant bound.

In section 2 we recall the original SMART algorithm and deal with the unweighted case. Section 3 deals with the weighted case. In section 4 we give examples to show that the bounds are within less than a factor of 2 of being tight. Section 5 contains conclusions.

2. The unweighted case. We begin by briefly reminding the reader of the original SMART algorithm; then we will present the variant that gives the better bound. SMART belongs to the category of so-called *shelf*-based algorithms.

Shelf solutions can be characterized by the following simple properties, illustrated in the left-hand side of Figure 1: the tasks are assigned to *shelves*, with all tasks on any given shelf having the same starting time. The sum of the required processors for all tasks on a given shelf must not exceed the total number of processors. The first shelf is placed at time zero. The *height* of a shelf is the largest task execution time of any task assigned to that shelf, and the next shelf is placed this height above the previous shelf. (If we think of the tasks as rectangles, the physical analogy becomes clear: rectangles are packed onto shelves, they must fit on the shelves, the first shelf sits on the floor, and each new shelf rests on the highest rectangle of the shelf before it.) If $\tau = \{(h_i, w_i) | i = 1, \ldots, M\}$ is a task system, we can think of a shelf assignment as a surjective function $\mathcal{S} : \{1, \ldots, M\} \to \{1, \ldots, S\}$ such that $\sum_{\mathcal{S}(i)=k} w_i \leq P$ for all $k = 1, \ldots, S$. Let $\mathcal{M}_k = \text{card}\{\mathcal{S}^{-1}(k)\}$ denote the number of tasks on shelf k, so that $\sum_{k=1}^{S} \mathcal{M}_k = M$. Let $\mathcal{H}_k = \max_{\mathcal{S}(i)=k} h_i$ denote the height of shelf k. Observe that the total area in the right-hand side of Figure 1, viewed as an integral, represents the sum of the completion times. Specifically, there is one "column" of unit width for each task. The unshaded portion of a given column has height equal to the execution time of the appropriate task. The shaded portion of the column has height equal to the waiting time of the task, which is the time until the task starts. This waiting time is simply the sum of the heights of the earlier shelves. Thus the entire column has height equal to the completion time of the task.

Consider a task system $\tau = \{(h_i, w_i) | i = 1, \ldots, M\}$. By normalization we will assume, without loss of generality, that the minimum task execution time $\min_{1 \leq i \leq M} h_i$

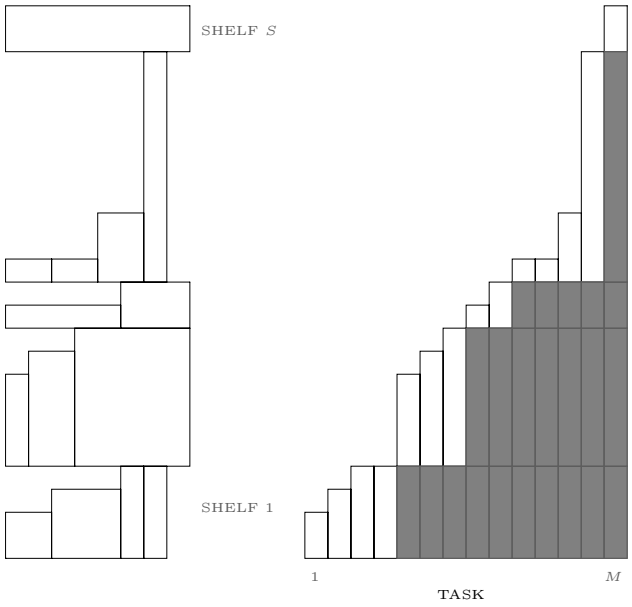


FIG. 1. *Shelf solution and its completion time sum.*

is equal to 1. Let $\hat{h} = \max_{1 \le i \le M} 2^{\lceil \log h_i \rceil}$. We partition τ into (at most) $1 + \log \hat{h}$ components by assigning task i to the component $j \in \{0, \ldots, \log \hat{h}\}$ that satisfies $2^{j-1} < h_i \le 2^j$. Now we assign tasks in each component j to shelves according to the *NFIW* (for *next fit increasing width*) bin packing algorithm [6, 7]: In other words, we reindex the tasks within component j in order of increasing width and assign them in sequence to shelves, which we regard as bins of size P. Each task is assigned to the current shelf, which is initialized to be shelf number one and incremented by one whenever a task does not fit. Now in order to obtain the SMART solution it remains to combine all the shelves in the various partitions into an overall ordering.

Note that for any shelf assignment $\mathcal{S}$, we can view the pair $(\tau, \mathcal{S})$ as a set of shelves. But these shelves can be permuted arbitrarily, resulting in solutions of varying quality. (In contrast, the ordering of shelves is irrelevant when employing a makespan objective function, since the makespan is simply the sum of the heights of the shelves.) The SMART algorithm arranges its shelves in the best way possible, according to the dictates of the following lemma.

LEMMA 2.1. *For any set of shelves $(\tau, \mathcal{S})$, an ordering of these shelves is optimal among all possible such shelf orderings if and only if $\frac{\mathcal{H}_1}{\mathcal{M}_1} \le \cdots \le \frac{\mathcal{H}_S}{\mathcal{M}_S}$.*

We will state and prove a generalization of this lemma in section 3. (See Lemma 3.1.)

Let $\sigma(\tau, \mathcal{S})$ be the schedule that results from ordering the shelves $(\tau, \mathcal{S})$ as specified by Lemma 2.1, and let $c(\tau, \mathcal{S})$ be its completion time sum.

Let s_τ and o_τ denote the objective function values for the SMART and optimal solutions for τ, respectively. In [12] it was shown that the SMART algorithm has time complexity $O(M \log M)$ and satisfies $\frac{s_\tau}{o_\tau} \le 32$. (This term is sometimes called an *approximation factor*.) In fact, using the techniques in section 3 we can now show that $\frac{s_\tau}{o_\tau} \le 9$, a significant improvement. To obtain a further improvement, we now consider a variant of the original algorithm. Specifically, instead of assigning tasks to shelves

via NFIW, we employ *FFIA* (*first fit increasing area*) [6, 7]: in other words, we reindex the tasks within component j in order of increasing area and assign them in sequence to shelves, which we again regard as bins of size P. Each task is assigned to the first shelf on which it fits, and the number of shelves is incremented by one whenever a task does not fit on any preceding shelf. (The rest of the algorithm, including the partitioning and combining, remains intact.) We regard this new algorithm as being in the same family of algorithms as the original one, and we use subscripts to differentiate the two algorithms when necessary: we denote the original and new algorithms by $\text{SMART}_{\text{NFIW}}$ and $\text{SMART}_{\text{FFIA}}$, respectively.

We illustrate the distinction between NFIW and FFIA with a simple example: suppose $P = 8$, and consider a task system in which there are four tasks. Specifically, there are two *narrow* tasks of width 3 and height 2, and two *wide* tasks of width 5 and height just greater than 1. These tasks might thus correspond to the first height component in a larger task system. NFIW would place the two narrow tasks on one shelf, followed by two shelves with a single wide task each. The areas of the wide tasks, however, are smaller than the areas of the narrow tasks. So FFIA would place the two wide tasks on separate shelves and then add one narrow task to each shelf.

The main new result in this section is the following theorem.

THEOREM 2.1. *For any task system $\tau = \{(h_i, w_i) | i = 1, \ldots, M\}$, the $\text{SMART}_{\text{FFIA}}$ algorithm has time complexity $O(M \log M)$ and satisfies $\frac{s_\tau}{o_\tau} \leq 8$.*

The time complexity remains unchanged from that of $\text{SMART}_{\text{NFIW}}$. To prove the bound, we will make use of two more lemmas from [12] (Lemmas 2.2 and 2.5 — part of this lemma is from [8]), as well as a number of entirely new results.

To get an upper bound on $\frac{s_\tau}{o_\tau}$, we will make use of lower bounds on o_τ. We now remind the reader of three such lower bounds.

LEMMA 2.2. *Let $\tau = \{(h_i, w_i) | i = 1, \ldots, M\}$ denote a task system indexed by increasing area $a_i = h_i w_i$. If $A_\tau = \frac{1}{P} \sum_{i=1}^{M} a_i (M - i + 1)$, $H_\tau = \sum_{i=1}^{M} h_i$, and $W_\tau = \frac{1}{P} \sum_{i=1}^{M} a_i$, then the optimal average response time solution o_τ satisfies $A_\tau \leq o_\tau$, $H_\tau \leq o_\tau$, and $A_\tau + \frac{1}{2} H_\tau - \frac{1}{2} W_\tau \leq o_\tau$.*

COROLLARY 2.1. *If a solution to the minimum average response time problem for task system τ has completion time sum v, where $v \leq \beta_1 [A_\tau + \frac{1}{2} H_\tau - \frac{1}{2} W_\tau] + \beta_2 H_\tau$, then $\frac{v}{o_\tau} \leq \beta_1 + \beta_2$.*

Once again, we delay the proof of Lemma 2.2 until section 3, when we will state and prove a more general result. (See Lemma 3.2.)

We will refer to A_τ and H_τ as the *squashed area* and *height* bounds, respectively. We will refer to the third bound as the *combined* bound. Note that the combined bound is tighter than the squashed area bound, since $W_\tau \leq H_\tau$. The height bound can be viewed as the area of the unshaded region in the right-hand side of Figure 1. The squashed area bound and W_τ are illustrated in Figure 2. The term W_τ is given by the area of the right-most of the M columns, while the squashed area bound is given by the total area of all the columns. The definition corresponds to vertical integration. But note that [12] gives an alternative expression corresponding to horizontal integration, namely, $A_\tau = \frac{1}{P} \sum_{j=1}^{M} \sum_{i=1}^{j} a_i$. This leads to the following expression for the squashed area bound, which is useful in situations where the tasks are not necessarily arranged in order of increasing area.

LEMMA 2.3. *Let $\tau = \{(h_i, w_i) | i = 1, \ldots, M\}$ denote a task system, indexed arbitrarily. Then $A_\tau = \frac{1}{P} \sum_{j=1}^{M} \sum_{i=1}^{j} \min(a_i, a_j)$.*

Proof. Observe that $\sum_{j=1}^{M} \sum_{i=1}^{j} \min(a_i, a_j) = \sum_{i \leq j} \min(a_i, a_j)$ is the sum over all pairs of elements of the minimum of each pair. This sum is independent of the way

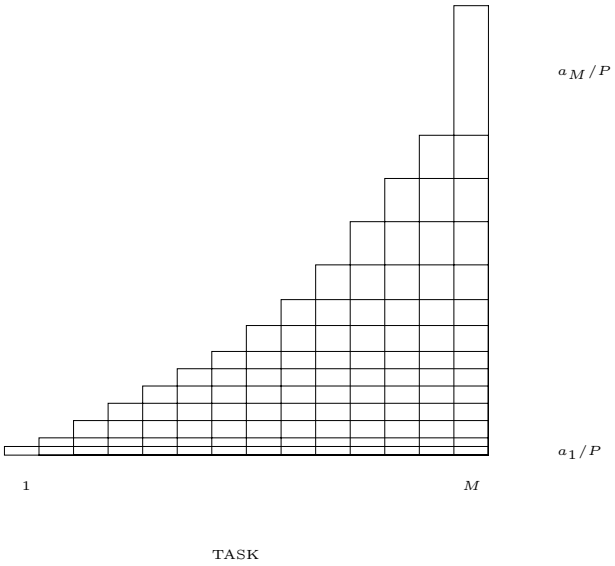


FIG. 2. *The squashed area bound.*

the elements are indexed. Suppose, then, that they are indexed in increasing order, $a_1 \le a_2 \le \cdots \le a_M$. Then $\frac{1}{P}\sum_{j=1}^{M}\sum_{i=1}^{j}\min(a_i, a_j) = \frac{1}{P}\sum_{j=1}^{M}\sum_{i=1}^{j} a_i = A_\tau$. □

In addition to lower bounds on o_τ, we will also need an upper bound on s_τ. To get this bound, we will partition the task system τ into two subsets and also adjust the task heights to facilitate the analysis. We will then consider schedules for the two subsets separately, proving bounds for each. Finally, we will consider the result of combining the two schedules to obtain a schedule for τ.

Consider the shelf assignment that results from applying $\text{SMART}_{\text{FFIA}}$ to the task system τ. We partition the tasks in τ into two subsets as follows. The first subset τ_1 will contain the tasks on the *first shelf* (if any) of each height component — that is, in each height component, the first shelf created by the FFIA packing. The second subset τ_2 will contain the tasks on all *remaining* shelves. (This partitioning is, in a sense, orthogonal to the original partitioning into components based on height. To avoid confusion we shall consistently use the term *component* when referring to the first partitioning, and the term *subset* when referring to the second partitioning. Also note that a similar partitioning into three subsets was given in [12].)

We now create a new task system $\hat{\tau}$, called the *double height* construction, whose tasks are in natural one-to-one correspondence with the tasks in τ. The number of processors required for each task in $\hat{\tau}$ will be identical to that of its counterpart in τ, while the task execution times will be at least as large but less than doubled, according to the following rule. Observe that the new task system $\hat{\tau} = \{(\hat{h}_i, w_i) | i = 1, \ldots, M\}$ will also partition into two subsets $\hat{\tau}_1$ and $\hat{\tau}_2$ corresponding task for task to τ_1 and τ_2. The height $\hat{h}_i$ of a task in the subset $\hat{\tau}_1$ will be $2^{\lceil \log h_i \rceil}$. The height $\hat{h}_i$ of a task in the subset $\hat{\tau}_2$ will be the height $\max_{\{j|\mathcal{S}(j)=\mathcal{S}(i)\}} h_j$ of the shelf containing the corresponding task i in τ. (This is similar to the double construction in [12], although here only the heights are modified, not the widths.)

Observe that all tasks in $\hat{\tau}$ on any given shelf have identical heights. Also note that by construction we have $H_\tau \le H_{\hat{\tau}} \le 2H_\tau$. Corresponding statements hold for the two subsets τ_1 and τ_2.

Let $\mathcal{S}_\tau$ denote the shelf assignment that results from applying $\text{SMART}_{\text{FFIA}}$ to the task system τ. (Then $s_\tau = c(\tau, \mathcal{S}_\tau)$.) To get a bound on s_τ, we will bound $c(\hat{\tau}, \mathcal{S}_\tau)$. First we establish a relationship between these two quantities.

LEMMA 2.4. *Any task system τ satisfies $c(\tau, \mathcal{S}_\tau) \leq c(\hat{\tau}, \mathcal{S}_\tau) + H_\tau - H_{\hat{\tau}}$.*

Proof. Note that the sets of shelves $(\tau, \mathcal{S}_\tau)$ and $(\hat{\tau}, \mathcal{S}_\tau)$ are identical except for the task heights, with each shelf in $(\hat{\tau}, \mathcal{S}_\tau)$ being at least as tall as its counterpart in $(\tau, \mathcal{S}_\tau)$. Let v_τ denote the sum of the completion times of the schedule that results from starting each shelf of $(\tau, \mathcal{S}_\tau)$ at the time the corresponding shelf starts in $\sigma(\hat{\tau}, \mathcal{S}_\tau)$. Since the shelves are potentially reordered and gaps potentially introduced, we have $c(\tau, \mathcal{S}_\tau) \leq v_\tau$. But $c(\hat{\tau}, \mathcal{S}_\tau) - H_{\hat{\tau}} = v_\tau - H_\tau$ represents the sum of the starting times of the tasks in $\sigma(\hat{\tau}, \mathcal{S}_\tau)$ and the new schedule for $(\tau, \mathcal{S}_\tau)$, respectively. The result follows. □

The next step is to bound the quantities $c(\hat{\tau}_1, \mathcal{S}_\tau)$ and $c(\hat{\tau}_2, \mathcal{S}_\tau)$.

LEMMA 2.5. *The partition $\hat{\tau}_1$ satisfies $c(\hat{\tau}_1, \mathcal{S}_\tau) \leq 2H_{\hat{\tau}_1}$.*

We will state and prove a generalization of this lemma in section 3. (See Lemma 3.4.)

LEMMA 2.6. *The partition $\hat{\tau}_2$ satisfies $c(\hat{\tau}_2, \mathcal{S}_\tau) \leq 2A_\tau - 2A_{\tau_1}$.*

The proof of this lemma is fairly elaborate, and we delay it until the end of this section. It is worth noting that Lemma 2.6 is the reason for using FFIA rather than NFIW. We can form analogues of all the other lemmas using $\text{SMART}_{\text{NFIW}}$, but we cannot prove an analogue to Lemma 2.6.

The final step is to get a bound on $c(\hat{\tau}, \mathcal{S}_\tau)$ in terms of $c(\hat{\tau}_1, \mathcal{S}_\tau)$ and $c(\hat{\tau}_2, \mathcal{S}_\tau)$.

LEMMA 2.7. *For any task system τ and any $\delta > 0$, the double height construction $\hat{\tau}$ satisfies $c(\hat{\tau}, \mathcal{S}_\tau) \leq (\delta + 1) \cdot c(\hat{\tau}_1, \mathcal{S}_\tau) + (\frac{1}{\delta} + 1) \cdot c(\hat{\tau}_2, \mathcal{S}_\tau)$.*

Proof. Let y_k denote the completion time of shelf k in its respective schedule, either $\sigma(\hat{\tau}_1, \mathcal{S}_\tau)$ or $\sigma(\hat{\tau}_2, \mathcal{S}_\tau)$. (Recall that all the tasks on each shelf in these schedules complete at the same time.) For each shelf k in $(\hat{\tau}_1, \mathcal{S}_\tau)$, let $y'_k = \delta y_k$, and for each shelf k in $(\hat{\tau}_2, \mathcal{S}_\tau)$, let $y'_k = y_k$. Then construct a schedule of all the shelves in $(\hat{\tau}, \mathcal{S}_\tau)$ by arranging them in order of ascending y'_k. Call this schedule $\mathcal{Z}$, and let z_k denote the completion time of shelf k in $\mathcal{Z}$.

Now consider a shelf k in $(\hat{\tau}_1, \mathcal{S}_\tau)$. Let l be the last shelf in $(\hat{\tau}_2, \mathcal{S}_\tau)$ that is completed before shelf k in $\mathcal{Z}$. Since l comes before k, we have $y_l = y'_l \leq y'_k = \delta y_k$. Then $z_k = y_k + y_l \leq (1 + \delta) y_k$. Therefore, the completion time of each task in $(\hat{\tau}_1, \mathcal{S}_\tau)$ in the schedule $\mathcal{Z}$ is not more than $\delta + 1$ times its completion time in the schedule $\sigma(\hat{\tau}_1, \mathcal{S}_\tau)$.

Next consider a shelf k in $(\hat{\tau}_2, \mathcal{S}_\tau)$. Let l be the last shelf in $(\hat{\tau}_1, \mathcal{S}_\tau)$ that is completed before shelf k in $\mathcal{Z}$. Since l comes before k, we have $\delta y_l = y'_l \leq y'_k = y_k$. Then $z_k = y_k + y_l \leq (1 + \frac{1}{\delta}) y_k$. Therefore, the completion time of each task in $(\hat{\tau}_2, \mathcal{S}_\tau)$ in the schedule $\mathcal{Z}$ is not more than $\frac{1}{\delta} + 1$ times its completion time in the schedule $\sigma(\hat{\tau}_2, \mathcal{S}_\tau)$.

Now the result follows from the fact that reordering the shelves according to Lemma 2.1 will not increase the sum of the completion times. □

Lemma 2.7 is a generalization of Lemma 3.5 in [12]. When $\delta = 1$, notice that both coefficients on the right-hand side become 2, as in the original lemma. Choosing a value of δ different from 1 will ultimately result in reducing the bound in Theorem 2.1 from 9 to 8, as will be seen.

Now we apply the above lemmas and observations together with some bookkeep-

ing to get the following string of inequalities:

$$\begin{aligned}
s_\tau &\le c(\hat{\tau}, \mathcal{S}_\tau) + H_\tau - H_{\hat{\tau}} \\
&\le (\delta+1)\cdot c(\hat{\tau}_1, \mathcal{S}_\tau) + \left(\frac{1}{\delta}+1\right)\cdot c(\hat{\tau}_2, \mathcal{S}_\tau) + H_\tau - H_{\hat{\tau}} \\
&\le 2\left(\frac{1}{\delta}+1\right) A_\tau - 2\left(\frac{1}{\delta}+1\right) A_{\tau_1} + H_\tau + 2(\delta+1)H_{\hat{\tau}_1} - H_{\hat{\tau}} \\
&\le 2\left(\frac{1}{\delta}+1\right) A_\tau + H_\tau + (2\delta+1)H_{\hat{\tau}} - 2(\delta+1)H_{\hat{\tau}_2} - 2\left(\frac{1}{\delta}+1\right) W_{\tau_1} \\
&\le 2\left(\frac{1}{\delta}+1\right) A_\tau + (4\delta+3)H_\tau - 2(\delta+1)H_{\tau_2} - 2\left(\frac{1}{\delta}+1\right) W_{\tau_1} \\
&\le 2\left(\frac{1}{\delta}+1\right) A_\tau + (4\delta+3)H_\tau - 2(\delta+1)W_{\tau_2} - 2\left(\frac{1}{\delta}+1\right) W_{\tau_1} \\
&\le 2\left(\frac{1}{\delta}+1\right) A_\tau + (4\delta+3)H_\tau - 2\min\left(\delta+1, \frac{1}{\delta}+1\right) W_\tau. \qquad (1)
\end{aligned}$$

Now choosing $\delta = \frac{1}{2}$, we get

$$s_\tau \le 6A_\tau + 5H_\tau - 3W_\tau = 6\left(A_\tau + \frac{1}{2}H_\tau - \frac{1}{2}W_\tau\right) + 2H_\tau \le 8o_\tau. \qquad (2)$$

This completes the proof of Theorem 2.1. (Note that choosing $\delta = 1$ yields instead a slightly poorer bound of 9.) We now return to the proofs that we omitted.

Proof of Lemma 2.6. Consider the following schedule of the shelves in $(\hat{\tau}_2, \mathcal{S}_\tau)$. Instead of arranging them in the order dictated by Lemma 2.1, arrange them in the following way. Let $\psi(k) = \min_{\mathcal{S}_\tau(i)=k}\{a_i\}$ be the area of the smallest-area task in τ on shelf k. Note that the area in question is the task's *original* area, not its area in the double height construction. Then reindex the shelves in $(\hat{\tau}, \mathcal{S}_\tau)$ so that $\psi(j) \le \psi(j+1)$ for all j. That is, we are ordering the shelves by increasing area of the least-area task. Take the shelves of $(\hat{\tau}_2, \mathcal{S}_\tau)$ in the order they are indexed, and call this schedule $\mathcal{D}$.

Let g_i denote the completion time of task $i \in \hat{\tau}_2$ in the schedule $\mathcal{D}$. Reordering the shelves according to Lemma 2.1 cannot increase the sum of the completion times, and therefore $c(\hat{\tau}_2, \mathcal{S}_\tau) \le \sum_{i\in\hat{\tau}_2} g_i$.

Reindex the tasks so that $\mathcal{S}_\tau(i) < \mathcal{S}_\tau(j) \Rightarrow i < j$. That is, tasks that appear on lower shelves have lower indices. Then define $\alpha_i = \sum_{j=1}^{i} \min\{a_i, a_j\}$. By Lemma 2.3, $A_\tau = \frac{1}{P}\sum_{i=1}^{M} \alpha_i$.

Our goal is to show that $g_i \le \frac{2}{P}\alpha_i$ for all $i \in \hat{\tau}_2$. From this we can conclude that

$$c(\hat{\tau}_2, \mathcal{S}_\tau) \le \sum_{i\in\hat{\tau}_2} g_i \le \frac{2}{P}\sum_{i\in\hat{\tau}_2}\alpha_i = 2\left(A_\tau - \frac{1}{P}\sum_{i\in\tau_1}\alpha_i\right) \le 2(A_\tau - A_{\tau_1}). \qquad (3)$$

For the analysis, we wish to treat each height component separately, and to this end we introduce the following notation. For any two tasks i and j, we write $i \sim j$ if they are in the same height component, i.e., $\lceil \log_2 h_i \rceil = \lceil \log_2 h_j \rceil$. For a given task i, consider the schedule consisting only of the shelves in $\mathcal{D}$ that contain tasks which are in the same height component as i, with the shelves ordered as they are in $\mathcal{D}$. Note that this is the order in which the shelves are created when the tasks are being packed onto shelves. Call this schedule $\mathcal{D}(i)$. For $i \in \hat{\tau}_2$, let $\tilde{g}_i$ denote the completion

time of task i in $\mathcal{D}(i)$. Also let $\tilde{\alpha}_i = \sum_{j \le i \wedge j \sim i} \min\{a_i, a_j\}$. The following lemma demonstrates a relationship between $\tilde{g}_i$ and $\tilde{\alpha}_i$.

LEMMA 2.8. *$\tilde{g}_i \le \frac{2}{P}\tilde{\alpha}_i$ for all $i \in \hat{\tau}_2$.*

We delay the proof of Lemma 2.8 until the completion of the proof of Lemma 2.6.

Now we are ready to show that $g_i \le \frac{2}{P}\alpha_i$ for all $i \in \hat{\tau}_2$. Pick any task $i_0 \in \hat{\tau}_2$. Suppose that there are r height components, other than the component containing the task i_0 itself, that contain a shelf which comes before i_0 in the schedule $\mathcal{D}$. For each such component $l \in \{1, \dots, r\}$, let i_l be the least-area task on the last shelf of the component l that comes before the shelf containing i_0. Note that the indexing of the shelves and the indexing of the tasks guarantee that $i_l < i_0$ for all $l \in \{1, \dots, r\}$. Also note that $a_{i_l} \le a_{i_0}$ for all $l \in \{1, \dots, r\}$. Now we have

$$\begin{aligned} g_{i_0} &= \sum_{l=0}^{r} \tilde{g}_{i_l} \le \frac{2}{P} \sum_{l=0}^{r} \tilde{\alpha}_{i_l} = \frac{2}{P} \sum_{l=0}^{r} \sum_{j \le i_l \wedge j \sim i_l} \min\{a_{i_l}, a_j\} \\ &\le \frac{2}{P} \sum_{l=0}^{r} \sum_{j \le i_0 \wedge j \sim i_l} \min\{a_{i_0}, a_j\} = \frac{2}{P} \sum_{j=1}^{i_0} \min\{a_{i_0}, a_j\} = \frac{2}{P}\alpha_{i_0}. \end{aligned} \tag{4}$$

Proof of Lemma 2.8. Pick some task $i_0 \in \hat{\tau}_2$. We will show that $\tilde{g}_{i_0} \le \frac{2}{P}\tilde{\alpha}_{i_0}$.

Let $\mathcal{S}_\tau^{-1}(k)$ denote the set of tasks on shelf k. Let $T_{i_0} = \{j \in \hat{\tau} : j \sim i_0\}$ denote the height component containing task i_0, and let $K = \{k : k < \mathcal{S}_\tau(i_0) \text{ and } \mathcal{S}_\tau^{-1}(k) \subseteq T_{i_0}\}$ be the set of shelves containing tasks in the same height component as i_0 that are indexed lower than the shelf containing i_0. These are the shelves that precede the task i_0 in the schedule $\mathcal{D}(i_0)$, plus the single shelf containing tasks from $T_{i_0} \cap \hat{\tau}_1$. We will examine the shelves in K and determine the contribution of each to $\tilde{g}_{i_0}$ and to $\tilde{\alpha}_{i_0}$. For this purpose, we will partition K into three subsets. Some additional notation is required first.

Define the *gap* of shelf k with respect to task i_0 to be $d_k = P - \sum_{\{i \in \mathcal{S}_\tau^{-1}(k) : a_i \le a_{i_0}\}} w_i$. Then d_k is "how much room" is left on shelf k at the time when task i_0 is assigned to a shelf. (If there are tasks on shelf k with the same area as i_0, then d_k may be less than the amount of room that is left on shelf k when i_0 is assigned to a shelf.) Note that $d_k < w_{i_0}$ for all $k \in K$ due to the use of first fit. Also note that there is at most one shelf $k \in K$ such that $d_k \ge \frac{P}{2}$. Let v denote this shelf if it exists; otherwise, let $v = \infty$. Let $k_0 = \min\{k \in K\}$ be the lowest-indexed shelf in T_{i_0}. Then the tasks on the shelf k_0 are in $\hat{\tau}_1$, not in $\hat{\tau}_2$, and so k_0 is not in $\mathcal{D}$ or $\mathcal{D}(i_0)$. Let $K^- = K \setminus \{k_0\}$.

Now we are ready to partition K into three subsets. Let $K_1 = \{k \in K^- : \exists i \in \mathcal{S}_\tau^{-1}(k) \text{ such that } a_i > a_{i_0}\}$ be the set of shelves containing a task larger than i_0 but not including the shelf k_0. Let $K_2 = \{k \in K^- \setminus K_1 : k > v\}$ be the set of shelves that do not contain any tasks larger than i_0 and that are scheduled after the shelf with $d_v \ge \frac{P}{2}$. Finally, let $K_3 = \{k \in K^- \setminus K_1 : k \le v\} \cup \{k_0\}$ be the remaining shelves — the shelf k_0 along with all shelves that do not contain any tasks larger than i_0 and that are scheduled not later than the shelf with $d_v \ge \frac{P}{2}$.

Let $h = \hat{h}_{i_0} = 2^{\lceil log_2 h_{i_0} \rceil}$, and let the height of shelf k be given by $\frac{h}{2} + x_k$, where $0 < x_k \le \frac{h}{2}$. Then the completion time of task i_0 in $\mathcal{D}(i_0)$ does not exceed h plus the sum of the heights of the shelves that precede it in $\mathcal{D}(i_0)$. That is,

$$\tilde{g}_{i_0} \le \sum_{k \in K^-} \left(x_k + \frac{h}{2} \right) + h. \tag{5}$$

We also have

$$\tilde{\alpha}_{i_0} = \sum_{j \le i_0 \wedge j \sim i_0} \min\{a_{i_0}, a_j\} \ge \sum_{k \in K} \sum_{j \in \mathcal{S}_\tau^{-1}(k)} \min\{a_{i_0}, a_j\} + a_{i_0}. \tag{6}$$

Now for $k \in K^-$, let

$$\lambda_k = \frac{2}{P} \sum_{j \in \mathcal{S}_\tau^{-1}(k)} \min\{a_{i_0}, a_j\} - \left(x_k + \frac{h}{2}\right). \tag{7}$$

Then from (5), (6), and (7) we have

$$\frac{2}{P}\tilde{\alpha}_{i_0} - \tilde{g}_{i_0} \ge \sum_{k \in K^-} \lambda_k + \frac{2}{P} \sum_{j \in \mathcal{S}_\tau^{-1}(k_0)} \min\{a_{i_0}, a_j\} + \frac{2}{P} a_{i_0} - h. \tag{8}$$

Now our goal is to show that the quantity in (8) is positive, and, to this end, we consider each of K_1, K_2, and K_3 in turn. Suppose that $k \in K_1$. Then there is at least one task on shelf k with area greater than a_{i_0}, so

$$\begin{aligned} \sum_{j \in \mathcal{S}_\tau^{-1}(k)} \min\{a_{i_0}, a_j\} &\ge a_{i_0} + \sum_{\{j \in \mathcal{S}_\tau^{-1}(k) : a_j \le a_{i_0}\}} a_j \\ &\ge a_{i_0} + \frac{h}{2} \left(\sum_{\{j \in \mathcal{S}_\tau^{-1}(k) : a_j \le a_{i_0}\}} w_j \right) \\ &= a_{i_0} + \frac{h}{2}(P - d_k). \end{aligned} \tag{9}$$

It follows from (7) and (9) that

$$\lambda_k \ge \frac{2a_{i_0}}{P} + h - \frac{hd_k}{P} - \left(x_k + \frac{h}{2}\right) > 0, \tag{10}$$

because $a_{i_0} = h_{i_0} w_{i_0} > \frac{h}{2} d_k$, and $h \ge x_k + \frac{h}{2}$.

Next consider $k \in K_2$. Suppose $q \in \mathcal{S}_\tau^{-1}(k)$. By the definition of K_2, we have $\mathcal{S}_\tau(q) > v$ and $a_q \le a_{i_0}$, and so $w_q > d_v \ge \frac{P}{2}$. Therefore, shelf k contains only one task q. We conclude that

$$\lambda_k = \frac{2}{P} a_q - h_q = \frac{2}{P} h_q w_q - h_q > 0, \tag{11}$$

because $w_q > \frac{P}{2}$.

Finally, consider the shelves in K_3. Let $k_0 < k_1 < \cdots < k_r$ denote all the shelves in K_3. For $q \ge 1$, we have

$$\sum_{j \in \mathcal{S}_\tau^{-1}(k_q)} \min\{a_{i_0}, a_j\} = \sum_{j \in \mathcal{S}_\tau^{-1}(k_q)} a_j \ge \frac{h}{2}(P - d_{k_q}) + x_{k_q} d_{k_{q-1}}. \tag{12}$$

This is because all tasks $j \in \mathcal{S}_\tau^{-1}(k_q)$ have $h_j > \frac{h}{2}$ and $w_j > d_{k_{q-1}}$. The latter is important only for the tallest task on shelf k_q, which has height $\frac{h}{2} + x_{k_q}$.

For $q = 0$, we have

$$\sum_{j \in \mathcal{S}_\tau^{-1}(k_0)} \min\{a_{i_0}, a_j\} \geq \sum_{\{j \in \mathcal{S}_\tau^{-1}(k_0) : a_j \leq a_{i_0}\}} a_j \geq \frac{h}{2} \sum_{\{j \in \mathcal{S}_\tau^{-1}(k_0) : a_j \leq a_{i_0}\}} w_j = \frac{h}{2}(P - d_{k_0}). \tag{13}$$

Then from (7), (12), and (13), we have

$$\begin{aligned}
&\sum_{k \in K_3 \setminus \{k_0\}} \lambda_k + \frac{2}{P} \sum_{j \in \mathcal{S}_\tau^{-1}(k_0)} \min\{a_{i_0}, a_j\} + \frac{2}{P} a_{i_0} - h \\
&\geq \sum_{q=1}^{r} \left(h - \frac{h d_{k_q}}{P} + \frac{2 x_{k_q} d_{k_{q-1}}}{P} - x_{k_q} - \frac{h}{2} \right) + h - \frac{h d_{k_0}}{P} + \frac{2 a_{i_0}}{P} - h \\
&= \sum_{q=1}^{r} \left(\frac{h}{2} - x_{k_q} - \frac{h d_{k_q}}{P} + \frac{2 x_{k_q} d_{k_{q-1}}}{P} \right) - \frac{h d_{k_0}}{P} + \frac{2 a_{i_0}}{P} \\
&= \sum_{q=1}^{r} \left(\frac{h}{2} - x_{k_q} - \frac{h d_{k_{q-1}}}{P} + \frac{2 x_{k_q} d_{k_{q-1}}}{P} \right) - \frac{h d_{k_r}}{P} + \frac{2 a_{i_0}}{P} \\
&= \sum_{q=1}^{r} \left(\frac{h}{2} - x_{k_q} \right) \left(1 - \frac{2 d_{k_{q-1}}}{P} \right) - \frac{h d_{k_r}}{P} + \frac{2 a_{i_0}}{P} \\
&> 0. \qquad (14)
\end{aligned}$$

The last inequality holds because $q-1 < r \leq u$, and so $d_{k_{q-1}} < \frac{P}{2}$ for all $q \in \{1, \ldots, r\}$, and because $a_{i_0} = h_{i_0} w_{i_0} > \frac{h}{2} d_{k_r}$.

Now from (7), (8), (10), (11), and (14), we conclude that $\tilde{g}_{i_0} \leq \frac{2}{P} \tilde{\alpha}_{i_0}$. □

3. The weighted case. Consider a task system $\tau = \{(h_i, w_i) | i = 1, \ldots, M\}$, and suppose now that each task i has weight u_i. In this section our goal is to minimize the weighted sum of the completion times given by $\sum_{i=1}^{M} u_i(t_i + h_i)$. Our algorithm for this problem is a generalization in two ways of the original $\text{SMART}_{\text{NFIW}}$ algorithm. We now describe each of these.

Recall that the original algorithm partitions tasks by height based on powers of 2. More generally, we can base the partitions on any power $\gamma > 1$. In other words, partition τ into components by assigning task i to the component j that satisfies $\gamma^{j-1} < h_i \leq \gamma^j$. If the rest of the algorithms $\text{SMART}_{\text{NFIW}}$ and $\text{SMART}_{\text{FFIA}}$ are unchanged, we obtain parametrized versions γ-$\text{SMART}_{\text{NFIW}}$ and γ-$\text{SMART}_{\text{FFIA}}$ of these algorithms, with the original algorithms corresponding to the special case $\gamma = 2$. We will employ γ-$\text{SMART}_{\text{NFIW}}$ here. (While we could have introduced this generalization in the previous section, we chose not to do so because it turns out not to improve the bound there.)

The second generalization allows us to effectively handle weighted tasks. Specifically, we now compute for each task i a ratio $\frac{w_i}{u_i}$ of width to weight, and we assign tasks in each component to shelves according to a *next fit increasing width to weight* bin packing algorithm. Let $\mathcal{U}_k$ be the total weight of the tasks on shelf k. (Note that if all the weights are unity, then $\mathcal{U}_k = \mathcal{M}_k$.) The shelves are combined in the order dictated by the following lemma.

LEMMA 3.1. *For any set of shelves $(\tau, \mathcal{S})$, an ordering of these shelves is optimal among all possible such shelf orderings if and only if $\frac{\mathcal{H}_1}{\mathcal{U}_1} \leq \cdots \leq \frac{\mathcal{H}_S}{\mathcal{U}_S}$.*

As in section 2, let $\sigma(\tau, \mathcal{S})$ be the schedule that results from ordering the shelves $(\tau, \mathcal{S})$ as specified by Lemma 3.1, and let $c(\tau, \mathcal{S})$ be the weighted sum of the completion times. This method of ordering the shelves amounts to Smith's ratio rule for scheduling tasks of width 1 on a single processor [10] and has the same proof of optimality.

Proof. Suppose that the shelves are not in order of nondecreasing $\frac{\mathcal{H}_k}{\mathcal{U}_k}$. Then there is a shelf k such that $\frac{\mathcal{H}_k}{\mathcal{U}_k} > \frac{\mathcal{H}_{k+1}}{\mathcal{U}_{k+1}}$. If the two shelves are interchanged, then the completion times of the tasks on shelf k will increase by $\mathcal{H}_{k+1}$ and the completion times of the tasks on shelf $k+1$ will decrease by $\mathcal{H}_k$. Therefore, interchanging the shelves decreases the weighted sum of the completion times by $\mathcal{U}_{k+1}\mathcal{H}_k - \mathcal{U}_k\mathcal{H}_{k+1} = \mathcal{U}_k\mathcal{U}_{k+1}(\frac{\mathcal{H}_k}{\mathcal{U}_k} - \frac{\mathcal{H}_{k+1}}{\mathcal{U}_{k+1}}) > 0$. □

The algorithm for weighted tasks is a straightforward generalization of its unweighted counterpart γ-SMART$_{\text{NFIW}}$, and we will refer to it by the same name. We will also retain the notation s_τ and o_τ to denote the weighted completion time sum for the γ-SMART$_{\text{NFIW}}$ and optimal solutions for τ, respectively.

THEOREM 3.1. *For any weighted task system $\tau = \{(h_i, w_i, u_i) | i = 1, \ldots, M\}$, the 1.65-SMART$_{\text{NFIW}}$ algorithm has time complexity $O(M \log M)$ and satisfies $\frac{s_\tau}{o_\tau} \leq 8.53$.*

As in section 2, we will begin with lower bounds on o_τ and then move on to upper bounds on s_τ. In order to obtain lower bounds on the total weighted response time, we extend the definitions of H_τ, A_τ, and W_τ. Specifically, we let $H_\tau = \sum_{i=1}^{M} u_i h_i$ be the weighted sum of the task heights. Suppose that the tasks are ordered such that $\frac{a_i}{u_i} \leq \frac{a_{i+1}}{u_{i+1}}$ for all tasks $i \in \{1, \ldots, M-1\}$. Then we define $A_\tau = \frac{1}{P}\sum_{i=1}^{M} u_i(\sum_{j=1}^{i} a_j)$. Note that we are now ordering the tasks according to Smith's ratio rule [10]. Finally, let $W_\tau = \frac{1}{P}\sum_{i=1}^{M} u_i a_i$. Then Lemma 2.2 and Corollary 2.1 can be extended to weighted tasks.

LEMMA 3.2. *The optimal solution o_τ satisfies $A_\tau \leq o_\tau$, $H_\tau \leq o_\tau$, and $A_\tau + \frac{1}{2}H_\tau - \frac{1}{2}W_\tau \leq o_\tau$.*

Proof. Given a schedule for the task system τ, let $f_\tau = \sum_{i=1}^{M} u_i(t_i + h_i)$ be its weighted completion time sum. Then $H_\tau = \sum_{i=1}^{M} u_i h_i \leq \sum_{i=1}^{M} u_i(t_i + h_i) = f_\tau$. This holds for any schedule of τ, and therefore $H_\tau \leq o_\tau$.

Next we will show that $A_\tau + \frac{1}{2}H_\tau - \frac{1}{2}W_\tau \leq o_\tau$, and from this it also follows that $A_\tau \leq o_\tau$, since $H_\tau \geq W_\tau$. Given a schedule for the task system τ, let $r_i(t)$ denote the fraction of task i that is not yet completed at time t. That is, let

$$r_i(t) = \begin{cases} 1 & \text{if } t \leq t_i, \\ 1 - \frac{t - t_i}{h_i} & \text{if } t_i < t \leq t_i + h_i, \\ 0 & \text{if } t_i + h_i < t. \end{cases} \tag{15}$$

Now let $r(t) = \sum_{i=1}^{M} u_i r_i(t)$ be the weighted number of tasks remaining at time t, including fractional tasks. Then

$$\begin{aligned} \int_0^\infty r(t)dt &= \sum_{i=1}^{M} u_i \int_0^\infty r_i(t)dt \\ &= \sum_{i=1}^{M} u_i \left[\int_0^{t_i+h_i} dt - \int_{t_i}^{t_i+h_i} \frac{t - t_i}{h_i} dt \right] \end{aligned}$$

$$= \sum_{i=1}^{M} u_i \left[(t_i + h_i) - \frac{h_i}{2} \right]$$

$$= f_\tau - \frac{1}{2} H_\tau. \tag{16}$$

Now we seek a lower bound on $r(t)$. Define a "squashed" task set $\dot{\tau}$ corresponding to τ in the following way. For each task $i \in \tau$, there is a corresponding task in $\dot{\tau}$ with width P, height $\frac{a_i}{P}$, and weight u_i. Then the squashed task has the same area and weight as the original task. Consider the schedule in which the tasks in $\dot{\tau}$ are ordered by increasing ratio of area to weight. Let $r^*(t)$ denote the function $r(t)$ for this schedule.

LEMMA 3.3. *For any schedule of τ, the corresponding function $r(t)$ satisfies $r(t) \geq r^*(t)$ for all t.*

We delay the proof of Lemma 3.3 until the completion of the proof of Lemma 3.2.

Let $f^*_{\dot{\tau}}$ be the weighted completion time sum of the squashed schedule. Then from (16) we have

$$\int_0^\infty r^*(t)dt = f^*_{\dot{\tau}} - \frac{1}{2} H_{\dot{\tau}}. \tag{17}$$

Now observe that $f^*_{\dot{\tau}} = A_\tau$ and $H_{\dot{\tau}} = W_\tau$. Then we have

$$f_\tau - \frac{1}{2} H_\tau = \int_0^\infty r(t)dt \geq \int_0^\infty r^*(t)dt = f^*_{\dot{\tau}} - \frac{1}{2} H_{\dot{\tau}} = A_\tau - \frac{1}{2} W_\tau. \tag{18}$$

We conclude that $o_\tau \geq A_\tau + \frac{1}{2} H_\tau - \frac{1}{2} W_\tau$.

Proof of Lemma 3.3. Consider the more general P processor parallel scheduling problem in which task i requires a_i units of processor time to complete, but there are no other restrictions. In other words, preemption without penalty is allowed, and the task execution need not take place within a rectangle of a given height and width. Given an arbitrary solution for this scheduling problem, let $X_i(t)$ denote the units of processor time completed for task i at time t. Then the fraction of task i that is not yet completed at time t can be written as $r_i(t) = 1 - X_i(t)/a_i$, a generalization of (15). We minimize $r(t) = \sum_{i=1}^{M} u_i r_i(t)$ over all possible schedules for τ, including those which are legal for our original rectangle packing scheduling problem. Formally, we wish to minimize $\sum_{i=1}^{M} u_i(1 - X_i(t)/a_i)$ subject to the constraints $X_i(t) \leq a_i$ for each i, and $\sum_{i=1}^{M} X_i(t) \leq Pt$. Removing the constant $\sum_{i=1}^{M} u_i$, we wish equivalently to maximize $\sum_{i=1}^{M} (u_i/a_i) X_i$. This trivial linear program can be solved exactly via a greedy algorithm. The solution corresponds precisely to ordering the squashed tasks by decreasing ratio of weight to area and processing as many of them sequentially as can be accomplished by time t. The resulting cost is $r^*(t)$. □

We now obtain an upper bound on s_τ in the same manner as in section 2. We partition the task system τ in the same way and prove bounds for schedules for the two subsets separately. Then we consider the result of combining the two schedules. However, we make use of a slightly different "double height" construction. In particular, we use the height $\hat{h}_i = \gamma^{\lceil \log_\gamma h_i \rceil}$ for *every* task i, not just those in the first subset $\hat{\tau}_1$. Then with the exception of Lemma 2.6, the relevant results from section 2 (Lemmas 2.4, 2.5, and 2.7) carry over directly to the modified algorithm and double height construction.

As before, we will let $\mathcal{S}_\tau$ denote the shelf assignment that results from applying $\text{SMART}_{\text{NFIW}}$ to the task system τ.

We now present a generalization of Lemma 2.5 incorporating γ.

LEMMA 3.4. *The partition $\hat{\tau}_1$ satisfies $c(\hat{\tau}_1, \mathcal{S}_\tau) \le \frac{\gamma}{\gamma-1} H_{\hat{\tau}_1}$.*

Proof. Consider a schedule in which the shelves in $\hat{\tau}_1$ are arranged in order of increasing height. Recall that the shelf heights are powers of γ, and that there is at most one shelf of each height. Therefore, if a given shelf has height h, then its completion time in this schedule is at most $h + \frac{1}{\gamma}h + \frac{1}{\gamma^2}h + \cdots \le \frac{1}{1-\frac{1}{\gamma}}h = \frac{\gamma}{\gamma-1}h$. Every task on a given shelf has the same height, and so the height of a task is the same as the height of its shelf. It follows that the completion time of a task $i \in \hat{\tau}_1$ does not exceed $\frac{\gamma}{\gamma-1}\hat{h}_i$. Now observe that reordering the shelves according to Lemma 3.1 can only reduce the weighted completion time sum of the schedule, and the result follows. ∎

In place of Lemma 2.6, we have the following pair of lemmas.

LEMMA 3.5. *For any task system τ, the double height construction $\hat{\tau}$ satisfies $A_{\hat{\tau}} \le \gamma A_\tau$ and $H_{\hat{\tau}} \le \gamma H_\tau$.*

Proof. First observe that $\hat{h}_i = \gamma^{\lceil \log_\gamma h_i \rceil} < \gamma^{\log_\gamma h_i + 1} = \gamma h_i$. It follows directly that $H_{\hat{\tau}} < \gamma H_\tau$. Suppose that the tasks are ordered by nondecreasing ratio of area to weight, i.e., $\frac{a_1}{u_1} \le \frac{a_2}{u_2} \le \cdots \le \frac{a_M}{u_M}$. Note that altering the task heights may put the tasks out of order with respect to the ratio of area to weight, but still we have $A_{\hat{\tau}} \le \frac{1}{P}\sum_{i=1}^{M}(\sum_{k=1}^{i}\hat{a}_k)u_i \le \gamma A_\tau$. ∎

LEMMA 3.6. *The partition $\hat{\tau}_2$ satisfies $c(\hat{\tau}_2, S_\tau) \le A_{\hat{\tau}} + A_{\hat{\tau}_2} - A_{\hat{\tau}_1}$.*

Proof. Consider the following schedule of the tasks in $\hat{\tau}_2$. Assign the tasks to shelves as in $\text{SMART}_{\text{NFIW}}$, but then do not arrange the shelves according to Lemma 3.1. Instead, arrange them according to the area to weight ratio of the task on each shelf with the smallest ratio, so that the shelves are in increasing order according to that ratio. Call this schedule $\mathcal{D}$. Let g_i denote the completion time of task $i \in \hat{\tau}_2$ in the schedule $\mathcal{D}$. Reordering the shelves according to Lemma 3.1 cannot increase the sum of the completion times, and, therefore, $c(\hat{\tau}_2, \mathcal{S}_\tau) \le \sum_{i\in\hat{\tau}_2} u_i g_i$. Suppose that the tasks in $\hat{\tau}$ are ordered by increasing ratio of area to weight, i.e., $\frac{\hat{a}_i}{u_i} \le \frac{\hat{a}_{i+1}}{u_{i+1}}$ for all $i \in \{1, \ldots, M-1\}$. Let $\alpha_i = \sum_{k=1}^{i}\hat{a}_k$ and $\beta_i = \sum_{k\le i \wedge k\in\hat{\tau}_2}\hat{a_k}$. Then $A_{\hat{\tau}} = \frac{1}{P}\sum_{i=1}^{M} u_i\alpha_i$ and $A_{\hat{\tau}_2} = \frac{1}{P}\sum_{i=1}^{M} u_i\beta_i$. Our goal is to show that $g_i \le \frac{\alpha_i+\beta_i}{P}$ for all $i \in \hat{\tau}_2$. From this we can conclude that

$$c(\hat{\tau}_2, \mathcal{S}_\tau) \le \sum_{i\in\hat{\tau}_2} u_i g_i \le \frac{1}{P}\sum_{i\in\hat{\tau}_2} u_i(\alpha_i+\beta_i) = A_{\hat{\tau}} - \frac{1}{P}\sum_{i\in\hat{\tau}_1} u_i\alpha_i + A_{\hat{\tau}_2} \le A_{\hat{\tau}} - A_{\hat{\tau}_1} + A_{\hat{\tau}_2}. \tag{19}$$

For the analysis, we wish to treat each height component separately, and to this end we recall the following notation. For any two tasks i and j, we write $i \sim j$ if they are in the same height component, i.e., $\lceil \log_\gamma h_i \rceil = \lceil \log_\gamma h_j \rceil$. For a given task i, consider the schedule consisting only of the shelves in $\mathcal{D}$ that contain tasks which are in the same height component as i, with the shelves ordered as they are in $\mathcal{D}$. Note that this is the order in which the shelves are created when the tasks are being packed onto shelves. Call this schedule $\mathcal{D}(i)$. For $i \in \hat{\tau}_2$, let $\tilde{g}_i$ denote the completion time of task i in $\mathcal{D}(i)$. Also let $\tilde{\alpha}_i = \sum_{j\le i\wedge j\sim i}\hat{a}_j$ and $\tilde{\beta}_i = \sum_{j\le i\wedge j\sim i\wedge j\in\hat{\tau}_2}\hat{a}_j$.

LEMMA 3.7. *$\tilde{g}_i \le \frac{\tilde{\alpha}_i+\tilde{\beta}_i}{P}$ for all $i \in \hat{\tau}_2$.*

We delay the proof of Lemma 3.7 until the completion of the proof of Lemma 3.6.

Now we are ready to show that $g_i \le \frac{\alpha_i+\beta_i}{P}$ for all $i \in \hat{\tau}_2$. Pick any task $i_0 \in \hat{\tau}_2$. Suppose that there are r height components, other than the component containing the task i_0 itself, that contain a shelf which comes before i_0 in the schedule $\mathcal{D}$. For

each such component $l \in \{1, \ldots, r\}$, let i_l be the task with the least area to weight ratio on the last shelf of the component l that comes before the shelf containing i_0. Note that the ordering of the shelves guarantees that $i_l < i_0$ for all $l \in \{1, \ldots, r\}$. Now we have

$$\begin{aligned}
g_{i_0} &= \sum_{l=0}^{r} \tilde{g}_{i_l} \\
&\leq \frac{1}{P} \sum_{l=0}^{r} (\tilde{\alpha}_{i_l} + \tilde{\beta}_{i_l}) \\
&= \frac{1}{P} \sum_{l=0}^{r} \left(\sum_{j \leq i_l \wedge j \sim i_l} \hat{a}_j + \sum_{j \leq i_l \wedge j \sim i_l \wedge j \in \hat{\tau}_2} \hat{a}_j \right) \\
&\leq \frac{1}{P} \sum_{l=0}^{r} \left(\sum_{j \leq i_0 \wedge j \sim i_l} \hat{a}_j + \sum_{j \leq i_0 \wedge j \sim i_l \wedge j \in \hat{\tau}_2} \hat{a}_j \right) \\
&= \frac{1}{P} \left(\sum_{j=1}^{i_0} \hat{a}_j + \sum_{j=1 \wedge j \in \hat{\tau}_2}^{i_0} \hat{a}_j \right) \\
&= \frac{\alpha_{i_0} + \beta_{i_0}}{P}. \qquad \square \qquad (20)
\end{aligned}$$

Proof of Lemma 3.7. Let i_0 be any task in $\hat{\tau}_2$. Let $T_{i_0} = \{i \in \hat{\tau} : i \sim i_0\}$ be the height component containing task i_0. Let $h = \hat{h}_{i_0} = \gamma^{\lceil \log_\gamma h_{i_0} \rceil}$ be the height of each task in T_{i_0}. Let the shelves in T_{i_0} be numbered consecutively according to the order in which they were created. Then shelf 1 is in $\hat{\tau}_1$ and shelf 2 is the first shelf in $\mathcal{D}(i_0)$.

Let us define $\phi(k)$ to be the first task $i \in T_{i_0}$ assigned to shelf k. Observe that $\phi(k)$ for $k > 1$ is too wide to fit on shelf $k-1$, and so the total width of any shelf k plus the width of $\phi(k+1)$ is greater than P. Since every task in T_{i_0} has height h, it follows that the total area of any shelf k plus the area of $\phi(k+1)$ is greater than hP. It also follows that ordering the tasks by increasing width to weight ratio is the same as ordering the tasks by increasing area to weight ratio. Thus, if task i_0 is assigned to shelf k, then the total area of all tasks in T up to and including i_0 satisfies $\tilde{\alpha}_{i_0} \geq (k-1)hP - \sum_{j=2}^{k} \hat{a}_{\phi(j)} \geq (k-1)hP - \tilde{\beta}_{i_0}$. On the other hand, the completion time of task i_0 in $\mathcal{D}(i_0)$ is $\tilde{g}_{i_0} = (k-1)h$. $\square$

Putting all of these lemmas together we get

$$\begin{aligned}
s_\tau &\leq c(\hat{\tau}, S_\tau) + H_\tau - H_{\hat{\tau}} \\
&\leq (\delta+1) c(\hat{\tau}_1, S_\tau) + \left(\frac{1}{\delta} + 1\right) c(\hat{\tau}_2, S_\tau) + H_\tau - H_{\hat{\tau}} \\
&\leq \left(\frac{1}{\delta} + 1\right) A_{\hat{\tau}} + \left(\frac{1}{\delta} + 1\right) A_{\hat{\tau}_2} - \left(\frac{1}{\delta} + 1\right) A_{\hat{\tau}_1} \\
&\quad + \frac{\gamma}{\gamma - 1} (\delta + 1) H_{\hat{\tau}_1} + H_\tau - H_{\hat{\tau}} \\
&= \left(\frac{1}{\delta} + 1\right) A_{\hat{\tau}} + \left(\frac{1}{\delta} + 1\right) A_{\hat{\tau}_2} - \left(\frac{1}{\delta} + 1\right) A_{\hat{\tau}_1} \\
&\quad + \left(\frac{1}{\delta} + 1\right) H_{\hat{\tau}_1} + \left[\frac{\gamma}{\gamma - 1}(\delta + 1) - \frac{1}{\delta} - 2\right] H_{\hat{\tau}}
\end{aligned}$$

$$
\begin{aligned}
&- \left[\frac{\gamma}{\gamma-1}(\delta+1) - \frac{1}{\delta} - 1\right] H_{\hat{\tau}_2} + H_\tau \\
&\le \gamma\left(\frac{1}{\delta}+1\right) A_\tau + \gamma\left(\frac{1}{\delta}+1\right) A_{\tau_2} - \left(\frac{1}{\delta}+1\right) A_{\tau_1} \\
&\quad + \gamma\left(\frac{1}{\delta}+1\right) H_{\tau_1} + \left[\frac{\gamma^2}{\gamma-1}(\delta+1) - \frac{\gamma}{\delta} - 2\gamma + 1\right] H_\tau \\
&\quad - \left[\frac{\gamma}{\gamma-1}(\delta+1) - \frac{1}{\delta} - 1\right] H_{\tau_2} \\
&\le \gamma\left(\frac{1}{\delta}+1\right) o_\tau + \gamma\left(\frac{1}{\delta}+1\right) A_\tau \\
&\quad + \left[\frac{\gamma^2}{\gamma-1}(\delta+1) - \frac{\gamma}{\delta} - 2\gamma + 1\right] H_\tau \\
&\quad - \min\left\{\frac{1}{\delta}+1, \left[\frac{\gamma}{\gamma-1}(\delta+1) - \frac{1}{\delta} - 1\right]\right\} W_\tau. \qquad (21)
\end{aligned}
$$

The next-to-last inequality is actually only guaranteed to hold if the term $\frac{\gamma}{\gamma-1}(\delta+1) - \frac{1}{\delta} - 2$ is nonnegative. In any case, the optimal values $\gamma = 1.65$ and $\delta = 0.72$ satisfy this condition, and yield $\frac{s_\tau}{o_\tau} \le 8.53$.

Note from (21) that choosing $\gamma = 2$ and $\delta = 1$ yields $\frac{s_\tau}{o_\tau} \le 9$, and thus the original algorithm of [12] for the unweighted problem is 9-approximate.

4. Examples. We first present a family of task systems for the unweighted minimum response time problem with the property that $\text{SMART}_{\text{FFIA}}$ produces a ratio $\frac{s_\tau}{o_\tau}$ approaching 4.5 asymptotically. There are two groups of tasks, a *height* group and an *area* group.

The tasks in the height group fit onto $n-1$ shelves of heights $4, 8, 16, \ldots, 2^n$, with the number of tasks on each shelf being equal to the height of that shelf. On each such shelf k with $2 \le k \le n$ there is one task of full height 2^k. All other tasks on that shelf are of height slightly larger than 2^{k-1}. The width of each task in the height group is 1, and P is chosen large enough so that the total number of tasks in the height group is less than or equal to $\frac{P}{4} - 3$. $\text{SMART}_{\text{FFIA}}$ will produce a schedule with cost $2^{2n+1} - 7 \cdot 2^n + 6$, while in the optimal case all tasks start concurrently and produce overall costs of $\frac{1}{3}(2^{2n+1} + 3 \cdot 2^n - 14)$.

The area group consists of q tasks of width $\frac{P}{2} + 1$ and height $1 + \frac{4}{P}$ and q tasks of width $\frac{P}{4} + \frac{2}{P} + 1.5$ and height 2. Note that all tasks in this group have the same area. Therefore, we may assume that $\text{SMART}_{\text{FFIA}}$ will produce a schedule with q shelves such that each shelf contains a wide and a narrow task. Assuming $P \to \infty$ the cost of this schedule is $2q^2 + q$. On the other hand, if q is a multiple of 6 the optimal schedule is achieved by placing all q wide tasks on top of each other side by side with a stack of $\frac{q}{2}$ of the narrow tasks. The remaining narrow tasks are divided into three equal stacks of $\frac{q}{6}$ tasks each and started after the completion of the wide and narrow stacks. (Three narrow tasks fit side by side.) The cost of this schedule is $\frac{4q^2}{3} + \frac{3q}{2}$. (For values of q which are not multiples of 6 this is a close approximation.)

Note that all shelves k in both groups have $\frac{\mathcal{H}_k}{\mathcal{M}_k} = 1$, so that the ordering of them is immaterial. It can therefore be assumed that $\text{SMART}_{\text{FFIA}}$ puts all shelves of the height group on top of the shelves of the area group, thus producing an additional cost

of $(2^{n+1} - 4)2q$. However, the optimal schedule is obtained by placing the optimal schedules for both the height and the area group side by side. For $q \approx 2^{n-1}$, the ratio $\frac{s_\tau}{o_\tau}$ then approaches 4.5 asymptotically as $q \to \infty$.

For the weighted minimum response time problem, there is a similar family of task systems for which γ-SMART$_{\text{NFIW}}$ produces a ratio $\frac{s_\tau}{o_\tau}$ approaching approximately 6.75 asymptotically for the value of γ specified in section 3. Once again, there is a height group and an area group.

The tasks in the height group fit onto $n-1$ shelves of heights $\gamma^2, \gamma^3, \ldots, \gamma^n$, and there are two tasks of width 1 on each shelf. On shelf k, where $2 \le k \le n$, there is one task of height γ^k and weight ϵ, where ϵ is small. The other task is of height slightly greater than γ^{k-1} and weight $\gamma^k - \epsilon$. Note that the total number of tasks in the height group is $2n-2$. For large values of n, the γ-SMART$_{\text{NFIW}}$ produces a schedule with one shelf per height group and cost $\gamma^{2n}\frac{\gamma(2\gamma-1)}{(\gamma^2-1)(\gamma-1)}$. The optimal schedule places all tasks of the height group side by side and has cost $\gamma^{2n}\frac{\gamma}{\gamma^2-1}$.

The area group consists of q tasks of width $\frac{P-2n+2}{2}$, height slightly greater than 1, and weight $\frac{\gamma(P-2n+2)}{P+2n}$, and q tasks of width $2n-1$, height γ, and weight $\frac{2\gamma(2n-1)}{P+2n}$. Note that all tasks have the same width to weight ratio. Therefore, assuming $P \gg n$, γ-SMART$_{\text{NFIW}}$ may produce a schedule with one task of each type per shelf. For large values of q, the cost of this schedule is $q^2\frac{\gamma^2}{2}$. The optimal schedule consists of two stacks of wide tasks placed side by side, with the narrow tasks placed on top. The cost of this schedule is $q^2\frac{\gamma}{4}$.

Once again, all shelves k in both groups have $\frac{\mathcal{H}_k}{\mathcal{U}_k} = 1$, so that the ordering of these shelves is immaterial. Combining both shelf schedules thus produces a further cost of $q \cdot \gamma^n \frac{\gamma^2}{\gamma-1}$. For $P \gg qn$, the combined optimal schedule consists of the optimal schedules of both the height and area groups placed side by side. For $q \approx 1.47 \cdot \gamma^n$ the ratio $\frac{s_\tau}{o_\tau}$ approaches approximately 6.75 asymptotically.

5. Conclusions. In this paper we have

1. developed a variant SMART$_{\text{FFIA}}$ of the original SMART algorithm for the (unweighted) minimum average response time problem and shown that this new algorithm has an approximation factor of 8;
2. developed a generalization γ-SMART$_{\text{NFIW}}$ of the original SMART algorithm and shown that this new algorithm handles the weighted minimum average response time problem with an approximation factor of 8.53;
3. shown that the original SMART algorithm has an approximation factor of 9 when applied to the unweighted problem;
4. given examples that show that the bounds for the new algorithms are tight to within a factor of less than 2.

We should point out that in the unweighted case these algorithms generalize as before to the so-called *malleable* [13, 14] versions of the problems, yielding the same bounds while retaining polynomial-time complexity. (Malleability here means that a task can be run on an arbitrary number of processors with an execution time that depends on the number of processors assigned. Thus the number of processors each task will use becomes a decision variable rather than an input parameter.) Malleable response time algorithms with significantly improved bounds can be achieved under certain reasonable special conditions. See [11, 8] for details.

REFERENCES

[1] B. Baker, E. Coffman, and R. Rivest, *Orthogonal packings in two dimensions*, SIAM J. Comput., 9 (1980), pp. 846–855.

[2] J. Blazewicz, J. Lenstra, and A. R. Kan, *Scheduling subject to resource constraints: Classification and complexity*, Discrete Appl. Math., 5 (1983), pp. 11–24.

[3] E. Coffman, M. Garey, D. Johnson, and R. Tarjan, *Performance bounds for level-oriented two-dimensional packing algorithms*, SIAM J. Comput., 9 (1980), pp. 808–826.

[4] M. Garey and R. Graham, *Bounds for multiprocessor scheduling with resource constraints*, SIAM J. Comput., 4 (1975), pp. 187–200.

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

[6] D. Johnson, *Near-Optimal Bin-Packing Algorithms*, Tech. Report MAC TR-109, MIT, Cambridge, MA, June 1973.

[7] D. Johnson, *Fast algorithms for bin packing*, J. Comput. System Sci., 8 (1974), pp. 272–314.

[8] W. Ludwig and P. Tiwari, *The Power of Choice in Scheduling Parallel Tasks*, Technical Report 1190, Computer Sciences Department, University of Wisconsin–Madison, November 1993.

[9] D. Sleator, *A* 2.5 *times optimal algorithm for packing in two dimensions*, Inform. Process. Lett., 10 (1980), pp. 37–40.

[10] W. Smith, *Various optimizers for single-stage production*, Naval Research Logistics Quarterly, 3 (1956), pp. 59–66.

[11] J. Turek, W. Ludwig, J. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. Yu, *Scheduling parallelizable tasks to minimize average response time*, in Proceedings of the 6th Annual Symposium on Parallel Algorithms and Architectures, Cape May, NJ, June 1994, pp. 200–209.

[12] J. Turek, U. Schwiegelshohn, J. Wolf, and P. Yu, *Scheduling parallel tasks to minimize average response times*, in Proceedings of the SIAM Symposium on Discrete Algorithms, Arlington, VA, January 1994, pp. 112–121.

[13] J. Turek, J. Wolf, K. Pattipati, and P. Yu, *Scheduling parallelizable tasks: Putting it all on the shelf*, in Proceedings of the ACM Sigmetrics Conference, Newport, RI, June 1992, pp. 225–236.

[14] J. Turek, J. Wolf, and P. Yu, *Approximate algorithms for scheduling parallelizable tasks*, in Proceedings of the 4th Annual Symposium on Parallel Algorithms and Architectures, San Diego, CA, June 1992, pp. 323–332.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 254–262

NEW APPROXIMATION GUARANTEES FOR MINIMUM-WEIGHT k-TREES AND PRIZE-COLLECTING SALESMEN*

BARUCH AWERBUCH†, YOSSI AZAR‡, AVRIM BLUM§, AND SANTOSH VEMPALA§

Abstract. We consider a formalization of the following problem. A salesperson must sell some quota of brushes in order to win a trip to Hawaii. This salesperson has a map (a weighted graph) in which each city has an attached demand specifying the number of brushes that can be sold in that city. What is the best route to take to sell the quota while traveling the least distance possible? Notice that unlike the standard traveling salesman problem, not only do we need to figure out the order in which to visit the cities, but we must decide the more fundamental question: which cities do we want to visit?

In this paper we give the first approximation algorithm having a polylogarithmic performance guarantee for this problem, as well as for the slightly more general "prize-collecting traveling salesman problem" (PCTSP) of Balas, and a variation we call the "bank robber problem" (also called the "orienteering problem" by Golden, Levi, and Vohra). We do this by providing an $O(\log^2 k)$ approximation to the somewhat cleaner k-MST problem which is defined as follows. Given an undirected graph on n nodes with nonnegative edge weights and an integer $k \leq n$, find the tree of least weight that spans k vertices. (If desired, one may specify in the problem a "root vertex" that must be in the tree as well.) Our result improves on the previous best bound of $O(\sqrt{k})$ of Ravi et al.

Key words. approximation algorithm, prize-collecting traveling salesman problem, k-MST

AMS subject classifications. 68Q20, 68Q25, 90C27, 90B06, 05C85

PII. S009753979528826X

1. Introduction.

1.1. The problem. Consider a salesperson who must sell some quota of R brushes in order to win a trip to Hawaii. This salesperson has a map (a weighted graph) of n cities in which each city has an attached demand specifying the number of brushes that can be sold in that city. What is the best route to take to sell the quota while traveling the least distance possible? Notice that unlike the standard traveling salesman problem (TSP), not only do we need to figure out the order in which to visit the cities, but we must decide the more fundamental question: which cities do we want to visit?

R. Ravi et al. [17] considered the cleanest case of the above problem, called the minimum-weight k-tree, or k-MST problem. In this problem, one is given a graph on n vertices with nonnegative distances on the edges, and a number $k \leq n$, and the goal is to find a tree of least total cost that spans k vertices. For $k = n$ this is the (easy) minimum spanning tree problem. For general k, however, the problem is

*Received by the editors June 26, 1995; accepted for publication (in revised form) December 23, 1996; published electronically June 15, 1998.

http://www.siam.org/journals/sicomp/28-1/28826.html

†Department of Computer Science, The Johns Hopkins University, Baltimore, MD 21218 (baruch@blaze.cs.jhu.edu) and MIT Lab. for Computer Science, Cambridge, MA, 02139. The work of this author was supported by Air Force contract TNDGAFOSR-86-0078, ARPA/Army contract DABT63-93-C-0038, ARO contract DAAL03-86-K-0171, NSF contract 9114440-CCR, DARPA contract N00014-J-92-1799, and a special grant from IBM.

‡Department of Computer Science, Tel Aviv University, Tel Aviv, Israel (azar@math.tau.ac.il). The work of this author was supported in part by an Allon Fellowship and by the Israel Science Foundation administered by the Israel Academy of Sciences.

§School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213-3891 (avrim@cs.cmu.edu, svempala@cs.cmu.edu). The work of these authors was supported in part by NSF National Young Investigator grant CCR-9357793 and a Sloan Foundation Research Fellowship.

NP-complete and has the same main difficulty faced by the above salesperson: which points to include and which to ignore? In fact, the k-MST problem nicely focuses on just that issue since once the set is determined, the least weight tree on that set is easy to find.

Cheung and Kumar [8] call this problem the "quorum-cast" problem; its applications are in the domain of communication networks.

The bank robber problem is the following: given the map of a city including the amounts of money in each bank, and a car with bounded gas tank, the robber has to rob the maximum amount of money without refueling after the first robbery (thus avoiding being reported to the police). This problem is also called the "orienteering problem" by Golden, Levi, and Vohra [13].

1.2. Prior work. Ravi et al. [17] provide an algorithm that achieves an approximation ratio of $O(\sqrt{k})$ for the k-MST problem on general graphs (i.e., the tree found is at most $O(\sqrt{k})$ times heavier than the optimal tree) and ratio $O(k^{1/4})$ for the special case of points in two-dimensional Euclidean space. Garg and Hochbaum [10] improved the ratio for the latter case to $O(\log k)$, which was then improved to a constant factor by Blum, Chalasani, and Vempala [6]. Heuristics for problems described above have been given by Balas [4] and by Cheung and Kumar [8].

1.3. Results of this paper. In this paper, we describe an algorithm that achieves an approximation ratio $O(\log^2 k)$ for the k-MST problem on general graphs, improving the previous bound of $O(\sqrt{k})$ [17]. Our results hold for both the rooted and unrooted versions of the problem. (In the rooted version there is a specified root vertex that must be in the tree produced.) This result immediately implies an $O(\log^2 R)$ approximation for the quota-driven salesperson described above (R is the quota): namely, just treat a vertex with "demand" d as a cluster of d vertices, find the R-MST, and then tour the tree in the standard way. In fact, our algorithm actually achieves the somewhat better bound of $O(\log^2(\min(R, n)))$ for this problem, and does not require the demands to be polynomial in n.

Our algorithm also extends easily to an $O(\log^2(\min(R, n)))$ bound for the prize-collecting traveling salesman problem (PCTSP) due to Balas [4] on undirected graphs. The PCTSP problem is just like the quota TSP problem above, but in addition, there are nonnegative penalties attached to each city and the salesperson's cost is the sum of the distance traveled plus the penalties on cities *not* visited. Thus, the quota problem can be thought of as the special case in which penalties are 0. The $O(\log^2(\min(R, n)))$ bound for the PCTSP follows immediately by concatenating the tour found by our algorithm (which ignores the penalties) to a tour found by an algorithm of Goemans and Williamson [12] that provides a factor-of-2 approximation to a relaxed version of the PCTSP in which the quota requirement is removed. (In the original PCTSP there is also a restriction that each city not be visited more than once; if this is desired and if the graph is a metric space, then we can achieve the same bound in the usual way.)

We also derive an approximation algorithm with similar bounds for the bank robber problem.

It is worthwhile to point out that our algorithms are easily implementable in a distributed environment, since they operate on the basis of local information.

1.4. Subsequent results. Since the initial (conference) publication of this paper [2], several results have appeared that build and improve upon those here. Rajagopalan and Vazirani [16] describe an algorithm that can be viewed as a somewhat "smoothed" version of the algorithm of this paper and prove that it achieves an

$O(\log k)$ approximation to the k-MST. In a further improvement, Blum, Ravi, and Vempala [7] prove that a version of the Goemans–Williamson [12] algorithm achieves a constant-factor approximation. Most recently, Garg [9] has improved this constant factor to 3. Also very recently, for the case of points in the plane with the Euclidean distance metric, Arora [1] and Mitchell [15] (see also [14]) have independently developed a polynomial-time approximation scheme (PTAS) for the k-MST problem, that is, an algorithm that for any fixed $\epsilon > 0$ can achieve a $(1 + \epsilon)$ approximation in polynomial time. Their result applies to a variety of related problems such as the TSP and the minimum Steiner tree problem.

2. The k-MST problem. We begin by presenting an algorithm for the k-MST problem that achieves an approximation ratio of $O(\log^3 k)$. We then describe an improvement that removes one of the logarithmic factors to achieve the ratio of $O(\log^2 k)$. Before presenting the algorithm, however, let us point out that the "rooted" and "unrooted" versions of the problem are essentially equivalent from the point of view of approximation for the following reason.

Given an algorithm for the rooted problem, to solve the unrooted case one can simply try all possible start vertices and then choose the smallest tree found. Given an algorithm for the unrooted version, to solve the rooted case when the weight ℓ of the optimal tree is known, just throw out all vertices of distance greater than ℓ from the root, solve the unrooted problem, and then connect the tree to the root for an added cost of at most ℓ. Note that the approximation factor may increase by 1. If the optimal cost ℓ is not known, simply sort the distances from the root to each of the n points in increasing order, run the algorithm n times throwing out the i farthest points in the ith iteration, and pick the best result.

In the rest of this section we will use OPT to denote the optimal k-tree and ℓ to denote its total weight.

Our algorithm and analysis contain two main ideas. The first is a measure used for grouping points into components in a Kruskal's-algorithm-like manner. The second is a bucketing technique that allows one to prove this measure to be useful. The measure we use is the following: given two components C_i, C_j, we examine the ratio: $d(C_i, C_j)/\min(|C_i|, |C_j|)$, where $d(\cdot, \cdot)$ is the distance according to the shortest-path metric and $|\cdot|$ is the size in terms of number of points. The general step of the algorithm will be joining together (using the shortest path) the two components for which this ratio is smallest.

The bulk of the argument will be for proving correctness of an algorithm for the following slight relaxation of our goal, which is similar to the "maximal dense" tree concept in [3]. Given k, we will find a tree on at least $k/4$ points whose weight is at most $O(\log^2 k)$ times the weight of the minimum k-tree. With this algorithm in place, it will be easy to remove the relaxation and solve our original problem. The Kruskal-like algorithm for this relaxed problem is as follows.

Algorithm Merge-Cluster.

1. Begin with n components, one for each point.
2. Join (using the shortest path) the two components such that the ratio of the distance between the components to the number of points in the smaller one is least. That is, join the pair C_i, C_j that minimize $d(C_i, C_j)/\min(|C_i|, |C_j|)$.
3. Repeat step 2 until some component has size at least $k/4$.

Theorem 1. *The weight of the largest component produced by Algorithm Merge-Cluster is at most* $4(\log_2 k)^2$ *times the weight of the optimal k-tree.*

The proof of Theorem 1 follows immediately from Lemmas 1 and 2 below.

LEMMA 1. *If at any time the largest ratio used by Algorithm Merge-Cluster so far is r, then any component of p points will have total weight at most $rp \log_2 p$.*

LEMMA 2. *Algorithm Merge-Cluster never uses a ratio larger than $\frac{8\ell \log_2 k}{k}$ where ℓ is the weight of the optimal k-tree.*

To prove Theorem 1 from these lemmas, just note that the only way in which the largest component produced could have size greater than $k/2$ is for the additional vertices to be included "for free" in the shortest path that makes up the final connection. Thus combining the bounds of the two lemmas yields the theorem.

We begin with a proof of the simpler lemma.

Proof of Lemma 1. Consider a joining of two components. Since the length of the connection used is at most r times the number of points in the smaller component, we can "pay for" the connection by charging a cost of at most r to each of the points in the smaller component. Any time a point is charged, the size of the component it belongs to at least doubles. So, any point in a component of p points has been charged a total cost at most $r \log_2 p$. Since the weight of a component is at most the total charge to points inside it, this proves the lemma. □

Proof of Lemma 2. In contradiction, suppose that at some time all components produced by the algorithm have size less than $k/4$ and the distance between any two is greater than $r = (8\ell \log_2 k)/k$ times the number of points in the smaller. Group the components into buckets based on size, where the ith bucket contains those components with between $k/2^i$ and $k/2^{i+1}$ points ($i = 2, 3, \ldots$). Now, throw out all components that do not intersect the optimal k-tree. Clearly, the optimal k-tree can have at most $k/4 + k/8 + \cdots < k/2$ points inside buckets that contain only one component. So, there is some bucket containing at least two components such that OPT has at least $k/(2 \log_2 k)$ points inside that bucket. Say all components in this bucket have size between s and $2s$. This means that the balls of radius $rs/2$ about each component do not touch each other, and OPT must intersect at least $k/(4s \log_2 k)$ components. Therefore OPT must have a connection cost greater than $rk/(8 \log_2 k) = \ell$, a contradiction. □

Algorithm Merge-Cluster immediately gives us a simple $O(\log^3 k)$ approximation algorithm for the k-MST problem as follows. For simplicity, we consider the rooted version. Also, for the moment, suppose that we know the weight ℓ of the optimal k-tree. In the procedure below, we view Algorithm Merge-Cluster as taking "k" as an argument.

ALGORITHM CONNECT-CLUSTERS.

1. Mark as "to be ignored" all vertices of distance greater than ℓ from the root.
2. While $k > 0$ do the following:
   (a) Run Algorithm Merge-Cluster on the unmarked vertices. (By this we mean that the distance between two components is still the shortest-path distance in the original graph, but only unmarked vertices are considered in computing a component's size.)
   (b) Let s be the size of the component that was found and mark its vertices as "to be ignored."
   (c) Set $k = k - s$ (number of vertices we still need).
3. Connect together all the components found in Step 2.

THEOREM 2. *Algorithm Connect-Clusters finds a tree of at least k points whose weight is at most $O(\log^3 k)$ times the optimal.*

Proof. Suppose that in the invocations of Algorithm Merge-Cluster so far we have found components with k' points total. Then, the optimal k-tree contains at least

$k - k'$ points in the graph remaining, and all these are within distance ℓ from the root. Thus, the next invocation of the algorithm will find a tree on at least $(k-k')/4$ points, at cost at most $O(\ell \log^2 k)$. So, the algorithm will be run at most $O(\log k)$ times and the sum total cost of all components found is at most $O(\ell \log^3 k)$. The cost to connect them together is a low-order $O(\ell \log k)$. □

We can remove the knowledge of the optimal cost ℓ from the above algorithm in the same manner as was done for converting the rooted version of the k-MST problem to the unrooted version. For improved efficiency, note that the true ℓ satisfies $\lambda \leq \ell \leq k\lambda$, where λ is the distance of the kth farthest vertex from the root. So we can begin with a guess of $\ell = \lambda$ and then double our guess if the numbers and sizes of the components found do not satisfy the guaranteed bounds, for a total of $O(\log k)$ iterations maximum.

We now show how to modify Algorithm Connect-Clusters to achieve an $O(\log^2 k)$ approximation. To do this, we use the following corollary to a result by Goemans and Williamson [12]. In [5] this is called a $(3, 6)$-TSP approximator. (Goemans and Kleinberg [11] have recently improved this to a $(2, 3)$-TSP approximator.)

FACT 1. *Given a weighted graph on n points and an $\epsilon > 0$, let L_ϵ be the length of the shortest tour that visits at least $(1-\epsilon)n$ points. One can find in polynomial time a path of length at most $6L_\epsilon$ that visits at least $(1-3\epsilon)n$ points.*

For simplicity, we describe the modified algorithm as either finding a tree of k points with cost at most $O(\ell \log^2 k)$ or else finding a tree on at least $k/4$ points with cost $O(\ell)$. It is not hard to see that this suffices because the latter case removes an $O(\log^2 k)$ factor from the bounds of Theorem 1 (which is even better). Let us also assume for simplicity that the algorithm is given the value of ℓ; this assumption can be removed, as was done for Algorithm Connect-Clusters above. The new algorithm works as follows.

ALGORITHM IMPROVED-CONNECT. Run Algorithm Connect-Clusters until components totaling at least $\frac{15}{16}k$ points have been found. This requires only a constant number of applications of Algorithm Merge-Cluster. For simplicity, if the number of points found exceeds $\frac{15}{16}k$, then discard points until we have only $\frac{15}{16}k$ left. Call this set of points S.

Now, apply the algorithm of Fact 1 with $\epsilon = \frac{3}{15}$ to the graph induced by the set S. Notice that if the optimal k-tree intersects at least a $(1 - \frac{3}{15})$ fraction of S, then this algorithm is guaranteed to find a path of length at most 6ℓ that visits at least $(1 - \frac{9}{15})\frac{15}{16}k = \frac{3}{8}k$ points. If the algorithm produces such a path, then halt: the MST on these points is a tree of cost $O(\ell)$ on more than $k/4$ points as desired.

On the other hand, if the algorithm of Fact 1 returns a path that either is longer than 6ℓ or else visits insufficiently many points, then we know that the optimal k-tree intersects less than a $(1 - \frac{3}{15})$ fraction of S (and so contains at least $k/4$ new points). We now apply Merge-Cluster, with argument $k/4$, on the remaining graph to find a new component with at least $k/16$ points. Connecting this component to the components in S results in a tree of k points of cost $O(\ell \log^2 k)$, and we are done.

We thus have the following theorem.

THEOREM 3. *Algorithm Improved-Connect provides an $O(\log^2 k)$ approximation for the k-MST problem and runs in polynomial time.*

3. Extensions of the basic k-MST algorithm. We now describe how the algorithms of the previous section can be used to give guaranteed approximations to the other problems mentioned in the introduction, such as

- the quota TSP problem,

- the PCTSP, and
- the bank robber (orienteering) problem.

3.1. Algorithms for quota-driven salesmen. In the quota TSP problem each vertex in the graph has some attached integral value $w_i \geq 0$ and the salesman has a target quota R. The goal is to find a route as short as possible that visits vertices whose sum total value is at least R. The salesman may visit a city several times, but if he does so he only receives its value once.

First, it is immediate that we can approximate the quota TSP to a factor of $O(\log^2 R)$. Simply replace each vertex of value w by w vertices all at the same location, find the approximate R-MST, and then traverse it at most twice. Notice that this bound might not be so good if R is much larger than n. Also, this approach naively requires R to be only polynomially large; however, since the first step of the k-MST approximation algorithm is to reconnect vertices at the same location into a cluster, we can view the replacement described above as just a thought experiment. We show now that the algorithm in fact achieves the better bound of $O(\log^2(\min(R, n)))$.

It will be simplest to view Algorithm Merge-Cluster as acting directly on the weighted vertices, merging the two components C_i, C_j that minimize

$$d(C_i, C_j)/\min(wt(C_i), wt(C_j))$$

where $wt(C)$ is the sum of the values of the vertices contained in C. Let us call this algorithm Merge-Weighted-Cluster even though it is really exactly the same algorithm, except for running time, as the thought experiment described above. For the analysis corresponding to Lemma 1, however, when two components are merged we will "pay for" the cost by charging to the smaller one in *number*, not in weight. This still means that for a connection of ratio r a vertex of weight w will be charged at most rw, if we charge vertices proportionally to their weight. But, now it is clear that a vertex will be charged at most $\log(p)$ times if it is in a component of p vertices, as opposed to a component having *weight* p. Thus we have the following lemma. (We also give a more formal proof below.)

LEMMA 3. *If at any time, the largest ratio used by the algorithm Merge-Weighted-Cluster so far is r, then any component of p points and total vertex weight w will have total edge weight (cost) at most $rw \log_2 p$.*

Proof. We prove the lemma by induction. It is true initially since the cost begins at 0. When merging two clusters C_i and C_j into C we note that

$$\begin{aligned} cost(C) &= cost(C_i) + cost(C_j) + d(C_i, C_j) \\ &\leq r \cdot wt(C_i) \cdot \log(|C_i|) + r \cdot wt(C_j) \\ &\quad \cdot \log(|C_j|) + r \cdot \min\{wt(C_i), wt(C_j)\} \\ &\leq r \cdot (wt(C_i) + wt(C_j)) \cdot \log(|C_i| + |C_j|) \\ &\leq r \cdot wt(C) \cdot \log(|C|). \qquad \square \end{aligned}$$

We can similarly improve Lemma 2 as follows.

LEMMA 4. *Algorithm Merge-Weighted-Cluster never uses a ratio larger than $O(\ell(\log_2 n)/R)$ where ℓ is the (edge) weight of the optimal tree having vertex weight R.*

Proof. Following the proof of Lemma 2 we have buckets containing the components of vertex weight $R/4$ to $R/8$, $R/8$ to $R/16$, etc. We stop, however, at weight $R/(10n)$ and put all components of that weight or less into one single bucket. Now

there are only $O(\log n)$ buckets instead of $O(\log R)$ and the final small bucket intersects the optimal tree in at most $R/10$ total weight, and so can be "thrown out" in the analysis. The rest of the proof of Lemma 2 then can be followed directly. □

The above two lemmas imply that Algorithm Improved-Connect of Theorem 3 in fact achieves a ratio of $O(\log^2 n)$ as well as $O(\log^2 R)$, which gives us our desired bound.

One technical point: the algorithm of Fact 1, which is just the PCTSP algorithm of Goemans and Williamson [12] with an appropriate setting of the prize values, works also for the vertex-weighted case. Therefore, Algorithm Improved-Connect runs in polynomial time even if R is large.

3.2. Algorithms for prize-collecting salesmen. As mentioned in the introduction, an approximation algorithm to the quota TSP problem can be transformed into an approximation algorithm to the PCTSP problem of [4]. The PCTSP is the following. You are given an undirected edge-weighted graph in which each vertex has a prize value and a penalty value. You are also given a quota R. The goal is to find a tour of minimum "cost" such that the sum of the prizes on the vertices visited is at least R, where *cost* is defined to be the length of the tour plus the sum of the penalties on vertices not visited. In other words, the PCTSP is the same as the quota TSP problem, but with the additional complication of vertices having penalties for *not* being on the tour.

The Goemans–Williamson algorithm [12] provides a 2-approximation to a version of the PCTSP in which the quota requirement R is removed. That is, the goal is simply to minimize the cost of the tour as defined above. To approximate the PCTSP problem of Balas, simply concatenate the tour found by the quota TSP approximator to a (rooted) tour found by the Goemans–Williamson algorithm. The tour found by the quota TSP approximator is guaranteed to meet the quota requirement and have length at most $O(\log^2(\min(R, n)))$ times the cost of the optimum solution. The second tour guarantees that the final result incurs a penalty totaling at most twice the cost of the optimum solution, while introducing only a small increase in the length. (Removing the quota restriction only decreases the cost of the optimum solution.) Thus we have the following theorem.

THEOREM 4. *There is a polynomial-time algorithm that approximates the PCTSP problem of Balas* [4] *on n-vertex undirected weighted graphs with a ratio $O(\log^2(\min(R, n)))$, where R is the required vertex weight to be visited.*

3.3. The bank robber (orienteering) algorithm. The bank robber (orienteering) problem [13] is much like the problem faced by our quota-driven salesperson, except that the distance d that may be traveled is fixed and the goal is to maximize the total value R of points visited. If we do not require a specified starting point, then we can approximate this problem to the same ratio as the quota-TSP problem as follows. We "guess" the value R, we run the quota TSP-approximator to find a path of length $O(d \log^2(\min(n, R)))$ visiting vertex weight R, we break the path found into segments of length $d/2$, and then we choose the segment that contains the most vertex value inside. Notice, however, that this does not approximate the orienteering problem with a specified start vertex (root) since there is no guarantee the "good" segment found will intersect the root.

4. A hard example for our algorithm. Our basic algorithm, Merge-Cluster, finds a tree on at least $k/4$ points with cost at most $O(\ell \log^2 k)$, where ℓ is the cost of the optimal k-tree. In fact, there exist examples that force the algorithm to pay

$\Omega(\ell \log^2 k)$, and we describe one such example here.

Define a 1-block to be a single point, a 2-block to be two points separated by distance 1, a 4-block to be two 2-blocks separated by distance 2, and more generally a 2^t-block to be two 2^{t-1}-blocks separated by distance 2^{t-1}. Note that a 2^t-block has 2^t points. Also, notice that all the points in such a block would be connected together by Merge-Cluster using connections of ratio 1.

Suppose we have a cluster of A points (by a cluster, we just mean that all the points are at the same location) separated by some distance from a cluster of B points. Let us define a "filling in" of the region between the two clusters as follows. First, if $A = 1$ or $B = 1$ then we do nothing. Otherwise, let $C = \lfloor \min(A/2, B/2) \rfloor$ and place a cluster of C points halfway between the A-point cluster and the B-point cluster. Then recursively fill in the region between the A-point and C-point clusters and the region between the C-point and B-point clusters.

A hard example for the algorithm can now be described as follows. The graph consists of two sets of points separated by a large distance. The first set of points is a k-block. The second set of points is constructed by placing two clusters of $4k/\log k$ points at a distance of $8k/\log k$ from each other, and then "filling in" the region between the clusters as described above. Given such a graph, there exists a tree on at least k points with total length $8k/\log k$. In particular, just connect all the points in the second set. However, the algorithm will instead connect together points in the first set, since the ratios are better there (1 versus 2), and pay a total cost of $\Omega(k \log k)$.

5. Open questions. The obvious open question is whether there are polynomial-time algorithms with better approximation ratios, i.e. logarithmic, or even constant, for the problems considered in this paper. As noted above, this question has been answered in the affirmative by subsequent work.

Another interesting open question is finding a polynomial-time polylogarithmic approximation for the rooted version of the bank robber (orienteering) problem. This problem appears to be much more difficult than the unrooted version. Intuitively, the difficulty with approximating the rooted problem is that many of the points on the optimal tour might be in a clump at distance just about $d/2$ from the root. Thus, a strategy that opportunistically visits nearer vertices on its way to that clump may find that it cannot reach those vertices and return in the given distance d.

Acknowledgments. We thank Noga Alon and Prasad Chalasani for helpful discussions, and the anonymous referees for their suggestions for improving the presentation of this paper.

REFERENCES

[1] S. Arora, *Polynomial time approximation schemes for Euclidean TSP and other geometric problems*, in Proc. 37th Annual IEEE Symposium on Foundations of Computer Science, 1996, pp. 2–11.

[2] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala, *Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen*, in Proc. 27th Annual ACM Symposium on Theory of Computing, 1995, pp. 277–283.

[3] B. Awerbuch, Y. Azar, and R. Gawlick, *Dense trees and competitive selective multicast*, unpublished manuscript, December 1993.

[4] E. Balas, *The prize collecting traveling salesman problem*, Networks, 19 (1989), pp. 621–636.

[5] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, *The minimum latency problem*, in Proc. 26th Annual ACM Symposium on Theory of Computing, 1994, pp. 163–171.

[6] A. Blum, P. Chalasani, and S. Vempala, *A constant-factor approximation for the k-MST problem in the plane*, in Proc. 27th Annual ACM Symposium on Theory of Computing, 1995, pp. 294–302.

[7] A. Blum, R. Ravi, and S. Vempala, *A constant-factor approximation algorithm for the k-MST problem*, in Proc. 28th Annual ACM Symposium on Theory of Computing, 1996, pp. 442–448.

[8] S. Cheung and A. Kumar, *Efficient quorumcast routing algorithms*, in Proc. INFOCOM '94, Vol. 2, Toronto, 1994, pp. 840–855.

[9] N. Garg, *A 3-approximation for the minimum tree spanning k vertices*, in Proc. 37th Annual IEEE Symposium on Foundations of Computer Science, 1996, pp. 302–309.

[10] N. Garg and D. Hochbaum, *An $O(\log k)$ approximation algorithm for the k minimum spanning tree problem in the plane*, in Proc. 26th Annual ACM Symposium on Theory of Computing, 1994, pp. 432–438.

[11] M. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, in Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1996, pp. 152–158.

[12] M. Goemans and D. Williamson, *A general approximation technique for constrained forest problems*, in Proc. 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1992, pp. 307–315.

[13] B. Golden, L. Levy, and R. Vohra, *The orienteering problem*, Naval Research Logistics, 34 (1987), pp. 307–318.

[14] J. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-MST problem*, in Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1996, pp. 402–408.

[15] J. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: Part* II - *a simple polynomial-time approximation scheme for geometric k-MST, TSP, and related problems*, SIAM J. Comput., to appear.

[16] S. Rajagopalan and V. Vazirani, *Logarithmic approximation of minimum weight k trees*, unpublished manuscript, 1995.

[17] R. Ravi, R. Sundaram, M. Marathe, D. Rosenkrantz, and S. Ravi, *Spanning trees short and small*, in Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1994, pp. 546–555.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 263–277

NEAR-LINEAR TIME CONSTRUCTION OF SPARSE NEIGHBORHOOD COVERS*

BARUCH AWERBUCH†, BONNIE BERGER‡, LENORE COWEN§, AND DAVID PELEG¶

Abstract. This paper introduces a near-linear time sequential algorithm for constructing a sparse neighborhood cover. This implies analogous improvements (from quadratic to near-linear time) for any problem whose solution relies on network decompositions, including small edge cuts in planar graphs, approximate shortest paths, and weight- and distance-preserving graph spanners. In particular, an $O(\log n)$ approximation to the k-shortest paths problem on an n-vertex, E-edge graph is obtained that runs in $\tilde{O}(n + E + k)$ time.

Key words. neighborhood covers, network decompositions, approximate shortest paths, spanners

AMS subject classifications. 05C85, 68R10, 05C65, 05C70

PII. S0097539794271898

1. Introduction.

1.1. Background. An *r-neighborhood* of a vertex in an undirected weighted graph is the set of vertices within distance r away from it in the graph. An *r-neighborhood cover* is a set of overlapping sets of vertices in the graph (called *clusters*) with the property that every vertex has its entire r-neighborhood contained in one of these clusters. An r-neighborhood cover efficiently represents the local neighborhoods in a graph when all clusters in the cover have low diameter and the overlap among clusters (measured by the maximum number of clusters that intersect at a vertex) is minimized.

There is an inherent tradeoff between achieving low diameter and low overlap. If the clusters are chosen to be all r-neighborhoods (namely, the balls of radius r around every vertex), then all clusters have diameter r, but each vertex can be in as many as n clusters, where n is the number of vertices in the network. On the other hand, the entire network can be considered a single cluster, in which case each vertex is in just one cluster, but the diameter of this cluster can be as large as n.

Sparse neighborhood covers were first introduced in [9] in the context of distributed computing. A general r-neighborhood cover construction with $O(r \log n)$ cluster diameter and $O(\log n)$ cluster overlap is presented in [9]. It is also shown

*Received by the editors July 15, 1994; accepted for publication (in revised form) March 2, 1997; published electronically June 15, 1998.

http://www.siam.org/journals/sicomp/28-1/27189.html

†Laboratory for Computer Science, MIT, Cambridge, MA 02139. Current address: The Johns Hopkins University, Baltimore, MD 21218 (baruch@blaze.cs.jhu.edu). This work was supported by Air Force contract AFOSR F49620-92-J-0125, NSF contract 9114440-CCR, DARPA contracts N00014-91-J-1698 and N00014-J-92-1799, and a special grant from IBM.

‡Department of Mathematics and Laboratory for Computer Science, MIT, Cambridge, MA 02139 (bab@theory.lcs.mit.edu). This research was supported in part by an NSF Postdoctoral Research Fellowship and an ONR grant provided to the Radcliffe Bunting Institute.

§Department of Mathematical Sciences and Department of Computer Science, The Johns Hopkins University, Baltimore, MD 21218. This research was supported in part by an NSF postdoctoral fellowship.

¶Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel (peleg@wisdom.weizmann.ac.il). This research was supported in part by an Allon Fellowship, a Bantrell Fellowship, a Minerva Fellowship, and a Walter and Elise Haas Career Development Award.

therein how to achieve an essentially optimal tradeoff between the diameter and overlap parameters. However, the construction of [9] takes $O(nE)$ time and space, where E is the number of edges of the graph.

1.2. Summary of results. In this paper, we present a new efficient construction method that achieves the optimal tradeoff in near-linear time. More specifically, the main result of this paper is a deterministic sequential algorithm for the construction of a sparse r-neighborhood cover with $O(r \log n)$ cluster diameter and $O(\log n)$ cluster overlap with running time and space $O(E \log n + n \log^2 n)$. The algorithm can also be applied for constructing an r-neighborhood cover with $O(\beta r)$ diameter and $O(\beta n^{1/\beta})$ overlap, in time and space $O((E\beta + n\beta^2)n^{2/\beta})$, where β is given an input to the algorithm, representing the desired tradeoff between diameter and overlap.

To achieve this, we refine the "set coarsening" method of [9] and show how to construct the cover by only producing breadth-first search (BFS) trees from carefully selected vertices. The novel aspect of our approach is that a BFS tree is carved out in such a way that one can bound the overlap with other BFS trees. It turns out that for $r > 1$, bounding the overlap between BFS trees poses significant difficulties. In order to derive good complexity bounds, we need to charge the work performed in overlapping regions against the work performed in disjoint regions. Also, we need to lower-bound the fraction of r-neighborhoods that have "escaped carving" in a given iteration.

While most previous applications of neighborhood covers were in the distributed domain, the new algorithm makes the r-neighborhood cover a viable data structure for speeding up certain *sequential* algorithms. In particular, the results in this paper automatically imply analogous improvements (i.e., from quadratic to near-linear time) to many of the results in the literature that rely on neighborhood covers as a "black box." These include finding small edge cuts in planar graphs in $\tilde{O}(E)$ time and space[1] (as a consequence of using the algorithm of this paper in conjunction with the algorithm in [24], thus improving the $O(n^2)$ time and space complexity obtained by using the construction method of [9] in [24]), and weight- and distance-preserving graph spanners with $\tilde{O}(E)$ running time and space (see [5, 12], down from $O(nE)$ time using [11] or a combination of [5] and [9]).

Another application that is derived explicitly in the sequel is a sequential algorithm for approximating k-pairs shortest paths. We describe this application next.

1.3. Approximating k-pairs shortest paths. The *k-pairs shortest paths* problem, as discussed in the sequel, is defined as follows. Given an undirected graph $G(V, \mathcal{E})$ (where $|V| = n$ and $|\mathcal{E}| = E$), nonnegative edge weights, and k pairs of vertices, find the length of the shortest path between each of these pairs.

Another version, henceforth referred to as the *explicit shortest paths (ESP)* problem, requires the algorithm to actually produce the paths. Clearly, the time complexity of the ESP version is lower-bounded by the length of the output, which is proportional to the total length of the requested paths.

The best known bounds for exact solution of the k-pairs shortest paths problem are based on one of two approaches: implementations of Dijkstra's algorithm and solutions based on matrix multiplication.

The first approach involves running Dijkstra's algorithm with a binary heap implementation of the priority queue, yielding time $O(kE \log n)$ [13], or with a Fibonacci heap implementation, yielding time $O(kn \log n + kE)$ [14]. For the all-pairs shortest

[1] We use the $\tilde{O}(R)$ notation to denote $O(R \log^{O(1)} R)$, for cleaner statement of bounds.

paths problem, the time bounds become $O(n^2 \log n + nE)$ and $O(nE \log n)$, respectively [13, 14].

There have been other improvements on the running time for the all-pairs shortest paths problem for some special case graphs [2, 10, 15, 16, 17, 18, 19, 22, 26], but they all have worst-case time $O(nE)$. Karger, Koller, and Phillips [19] show that $\Omega(nE)$ is a lower bound for directed graphs on the running time of any algorithm which is "path-comparison based," which is evidence that it is hard to improve exact algorithms with this approach.

The second approach is based on computing (exact) all-pairs shortest paths relying on matrix multiplication. Alon, Galil, and Margalit [3] describe an $O((nW)^{2.688})$ time algorithm for the case of integer edge weights whose absolute value is less than W. This result has been improved to give an algorithm for the unweighted case which runs in time $O(M(n) \log n)$, where $M(n)$ is the time for matrix multiplication (currently known to be $o(n^{2.376})$) [25]. Subsequently, this result has been further improved in [4], solving exact all-pairs shortest paths in the case of integer edge weights between 0 and W in $O(W^2 M(n) \log n)$ time. Notice that the algorithm presented in this paper runs faster than these recent algorithms in both the unweighted and weighted cases.

Hence the bottlenecks in the running time of the exact algorithms stem either from repeated application of Dijkstra's single-source shortest paths algorithm or from matrix multiplication. We bypass these bottlenecks at the cost of producing an approximate solution.

There has also been recent work on approximation algorithms for shortest paths in the domain of parallel computation. Klein [20] extended the work of Ullman and Yannakakis [27] to the weighted case, obtaining a randomized PRAM algorithm that can $(1+\epsilon)$-approximate shortest paths between k pairs of vertices in $O(\sqrt{n}\epsilon^{-2} \log n \cdot \log^* n)$ time using $(kE \log n)\epsilon^2$ processors (where ϵ is constant). When this algorithm is run sequentially, the time is not as good as known sequential algorithms for the exact problem. Hence, Klein's approximation algorithm is strictly of interest for parallel computation.

As a corollary of our near-linear cost construction of sparse neighborhood covers, we obtain an $O(\log n)$-approximation algorithm for this problem that runs in $\tilde{O}(E + k)$ time. Our improvement in running time is achieved through our ability to construct a data structure of $\delta = \lceil \log \text{Diam}(G) \rceil$ sparse neighborhood covers (where $\text{Diam}(G)$ is the diameter of the graph) in time $O((E \log n + n \log^2 n)\delta)$, thereby allowing us to then quickly retrieve short paths. (The previous construction of [9] has been insufficient to produce any improvement over existing algorithms for the shortest paths application.) Once the data structure is in place, we show how to query it for the approximate distance between two vertices in $O(\log n \log \delta)$ time. (For ESP, the explicit version of the problem, the cost increases (inherently) by the length of the path.) Thus we approximate k-pairs shortest paths in time $O((E \log n + n \log^2 n)\delta + k(\log n \log \delta))$ (plus the total length of the k paths, for the ESP version).

More generally, a tradeoff can be achieved between the quality of the approximation and the running time; for example, a 32β approximation takes $O((E\beta + n\beta^2)n^{2/\beta}\delta + k\beta n^{1/\beta} \log \delta)$ time, for any $\beta \geq 1$. Note that we can achieve a better running time than known exact algorithms even for paths that are only a constant factor times the length of the shortest path. Further work in this direction is reported by Cohen [12].

We remark that there are no algorithms known for the single-pair shortest path problem that run asymptotically faster than the best single-source algorithms in the worst case [13]. Consequently, classical algorithms for k-pairs shortest paths have

done just as well using single-source shortest paths algorithms repeatedly.

1.4. Structure of this paper. The remainder of the paper is organized as follows. In section 2, we provide the graph-theoretic definition of neighborhood covers [9]. Section 3 presents an overview of the sparse neighborhood covers construction algorithm, followed by a formal presentation of the algorithm. The analysis of the algorithm is given in section 4. Finally, section 5 presents the application of the new algorithm to near-shortest path computation.

2. Network covers. Let $\mathrm{dist}(u,v)$ denote the minimum distance between u and v in the graph G. For a cluster of vertices $S_i \subseteq V$, let $\mathrm{dist}_{S_i}(u,v)$ denote the minimum distance from u to v in the subgraph induced by S_i. The *diameter* of the cluster is defined as $\mathrm{Diam}(S_i) = \max_{u,v \in S_i}(\mathrm{dist}_{S_i}(u,v))$.

The *r-neighborhood* of a vertex v is defined as $N_r(v) = \{u \mid \mathrm{dist}\ (u,v) \leq r\}$.

A *sparse neighborhood cover* is defined as follows. A *(β, r)-neighborhood cover* is a collection of sets (also called *clusters*) of vertices $S_1, \ldots, S_l$, with the following properties.

1. For every vertex v there exists some $1 \leq i \leq l$ s.t. $N_r(v) \subseteq S_i$.
2. For every i, $\mathrm{Diam}(S_i) \leq O(\beta r)$.

The *overlap* of the cover is the maximum number of clusters a vertex belongs to. A (β, r)-neighborhood cover is said to be *sparse* if its overlap is at most $\beta n^{1/\beta}$.

For the applications, β can be used to control the tradeoffs between complexity and the quality of the approximation, and we typically set $\beta = \log n$. Then a sparse $(\log n, r)$-neighborhood cover is a collection of sets that contain all r-neighborhoods such that the diameter of the sets is bounded by $O(r \log n)$, and each vertex is contained in at most $O(\log n)$ sets.

We remark that the diameter/sparsity tradeoff in this definition is tight to within a constant factor. In particular, for $\beta = \log n$, there exist graphs for which any $(\log n, r)$-neighborhood cover places some vertex in at least $\Omega(\log n)$ sets [21].

A *sparse neighborhood covers data structure* is a family of sparse neighborhood covers for a fixed β and different values of r. Applications typically require the construction of sparse r-neighborhood covers for $O(\log \mathrm{Diam}(G))$ successively doubled values of r (namely, $r = 1, 2, 4, 8, \ldots$).

3. The cover construction algorithm.

3.1. Overview. Let us start with an overview of the algorithm. The algorithm builds the cover $\mathcal{X}$ in phases. Each phase produces a new subcollection $\mathcal{Y}$ of sets to be added to the cover. All sets added to the cover have low diameter, and each vertex will appear in at most one set per phase. In addition, an $\Omega(n^{-1/\beta})$ fraction of the vertices whose r-neighborhoods have so far remained uncovered by any set already in the cover, will have their r-neighborhoods placed in some set constructed in the current phase. Hence, in $\beta n^{1/\beta}$ phases the algorithm finishes covering the r-neighborhood of all vertices in the graph. Since each vertex appears in at most one set per phase, each vertex will appear in at most $\beta n^{1/\beta}$ sets in the output cover.

Fix a phase of the algorithm, and let U be the collection of vertices whose r-neighborhood remains uncovered. Let $\mathcal{Y}$ denote the subcollection of sets we will place in the cover in the current phase.

We now sketch how to construct the sets $\mathcal{Y}$, in near-linear time. Low diameter sets Y are constructed one at a time, in a breadth-first manner, and added to $\mathcal{Y}$. The key element is a new "guarded" BFS technique, which carefully controls how new sets are grown around existing sets in $\mathcal{Y}$. In particular, a two-layer "shield" is grown

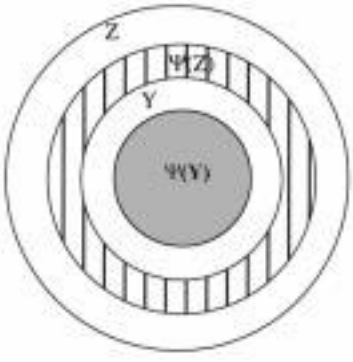


FIG. 1. *The layers of a grown cluster. The sets Y, $\Psi(Z)$, and Z all grow in successive bands of width r around $\Psi(Y)$.*

around each set Y. The shield serves a dual purpose: it keeps different sets in $\mathcal{Y}$ from overlapping, and it also insures that several sets do not duplicate work by performing BFS forays again and again into the same area of the graph.

Section 3.2 describes how the algorithm constructs a single set Y in the cover, and section 3.3 shows how a new nonoverlapping set Y grows around existing sets Y in the collection $\mathcal{Y}$. The details of the algorithm, including the code for the procedures, are presented in section 3.4. A formal correctness proof and complexity analysis of the algorithm appear in section 4.

3.2. Growth of a single set. We show how to grow a single set Y for the cover. Procedure `Cluster` grows Y iteratively, in layers, and outer layers form a shield around Y. We keep track of four layers around an "internal kernel" called $\Psi(Y)$. The set $\Psi(Y)$ consists of those vertices whose r-neighborhood is fully subsumed in the set Y; Z is the $2r$-neighborhood of Y; and $\Psi(Z)$ consists of those vertices whose r-neighborhood is subsumed by Z (see Figure 1). Note that

$$\Psi(Y) \subset Y \subset Z \quad \text{and} \quad \Psi(Y) \subset \Psi(Z) \subset Z.$$

We will sometimes refer to the set Z as the set Y's associated *cluster*.

The set Y starts growing as follows. Initially $\Psi(Y)$ is just the single vertex v, and Y is its r-neighborhood. The sets Y, $\Psi(Z)$, and Z all grow in successive bands of width r around $\Psi(Y)$. Specifically, if the stopping condition of Procedure `Cluster` is not met, then at the next growth iteration, Y grows to include the whole of the old Z, $\Psi(Y)$ grows to the old $\Psi(Z)$, the algorithm grows the BFS to construct a new Z which contains the entire $2r$-neighborhood of the new Y, and the new $\Psi(Z)$ consists of those vertices whose r-neighborhood lies in the new Z. Note that Y, $\Psi(Z)$, and Z in this picture are entire balls, not just rings; i.e., they contain all vertices within their borders. However, also note that it is not necessarily the case that $Y \subset \Psi(Z)$ (see section 3.3).

The stopping condition of Procedure `Cluster` is actually the conjunction of three separate conditions. The first stopping condition says that the number of still uncovered vertices in Y must be an appropriately large (specifically, $\Omega(n^{-1/\beta})$) fraction of those in Z. The second condition says that the number of still uncovered vertices in $\Psi(Y)$, i.e., the interior of the set Y, must be a sufficiently large fraction of those in

$\Psi(Z)$. These two conditions help guarantee that the cover is sparse, as will be argued below. A third stopping condition makes sure that the BFS exploration of the outer Z layer does not involve too much computation. Define $\lambda(v)$ to be the degree of vertex v in the graph G. For a set of vertices W, let $\lambda(W) = \sum_{v \in W} \lambda(v)$. The final stopping condition says that $\lambda(Y)$ is a sufficiently large fraction times $\lambda(Z)$. We argue below that all three stopping conditions are guaranteed to be met simultaneously within $O(\beta)$ iterations of Procedure `Cluster`. Thus, the diameter of a Y will be at most $O(r\beta)$.

3.3. A subcollection of sets. In this section, we show how a set Y and its cluster grow around previously built nonoverlapping sets Y from the same subcollection $\mathcal{Y}$ in the cover $\mathcal{X}$. A new cluster is always grown starting from a vertex v which lies outside the shielding ball $\Psi(Z)$ of any previous Y. Notice that this insures that the entire r-neighborhood of v lies outside any previous Y.

When a new cluster then grows into territory already occupied by a previous cluster, its BFS growth is curtailed by the previous cluster's layers. Each of the successive shields around an existing cluster permit a different degree of penetration.

- No cluster is allowed to grow into a previous cluster's Y. This ensures that the sets put in $\mathcal{Y}$ are disjoint. (See Figure 2.)
- A cluster's layer $\Psi(Z)$ does not grow into previous sets $\Psi(Z)$.
- The Z level is entirely permeable. (We control the cost of repeated BFS forays into previous Z instead by a stopping condition on the growth of a single cluster.)

We point out, to aid comprehension, that the relation $Y \subseteq \Psi(Z)$ may not hold for clusters grown after the first cluster (see Figure 3), since $\Psi(Z)$ is restricted to include new vertices if they have not already been placed in some previous cluster's associated set $\Psi(Z)$.

Figures 2–4 give snapshots of how new clusters grow. Figure 2 depicts how a new cluster begins growing, with $\Psi(Y)$ selected to be any single vertex outside the previous sets $\Psi(Z)$. Notice that this ensures that Y, the r-neighborhood of $\Psi(Y)$, will not overlap with previous Y's. Hence previous Y's form an impenetrable barrier (bold line) that nothing else can enter. In addition, the new cluster's $\Psi(Z) \setminus Y$ (striped region) does not enter previous sets $\Psi(Z)$.

Figure 3 illustrates the formation of the second layer of the new cluster (when the stopping condition was not met in the first iteration). Notice that the new $\Psi(Y)$ does not contain all of Y from the previous iteration, since $\Psi(Y)$ is the old $\Psi(Z)$, which does not extend through previous clusters' $\Psi(Z)$ sets. Even though Y and Z do not extend into the territory of previous clusters' Y sets, Y contains the r-neighborhood of $\Psi(Y)$, and Z contains the r-neighborhood of $\Psi(Z)$.

Finally, Figure 4 illustrates how a third cluster begins to grow. As before, the kernels $\Psi(Y)$ are shaded, the sets Y are marked with a bold line, the $\Psi(Z)$ is striped, and Z is within the outside ring. Excepting the participation of vertices in the outer Z ring, every vertex participates in the BFSs at most twice: a vertex can lie in at most one set Y, and prior to being placed in Y, it could lie in at most one $\Psi(Z)$. The BFS performed in the Z layer can be charged to the Y cluster for which it was grown. When subcollection $\mathcal{Y}$ is complete, every vertex appears in a $\Psi(Z)$, for some cluster Z, and the stopping conditions ensure that a sufficiently large fraction of the vertices appear in $\Psi(Y)$ for some Y. Thus every vertex appears in at most one Y, and the same fraction of the vertices have their r-neighborhoods contained in Y as well.

We stop growing clusters in the subcollection $\mathcal{Y}$ when every vertex is in $\Psi(Z)$ for

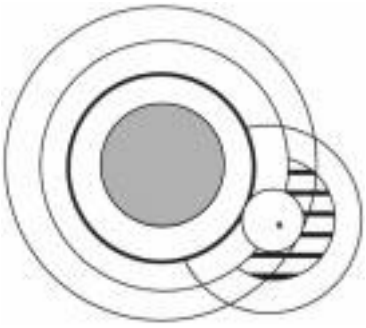


FIG. 2. *A new cluster begins growing around a single-vertex set* $\Psi(Y)$ *outside the previous sets* $\Psi(Z)$.

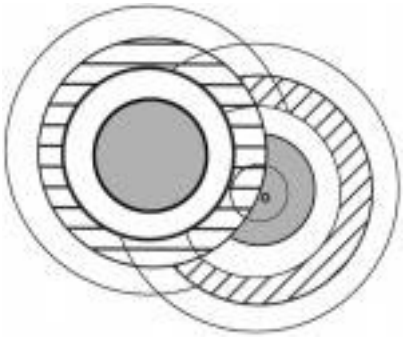


FIG. 3. *Formation of the second layer of the new cluster.*

some cluster Z.

3.4. Details of the algorithm. This subsection introduces Algorithm `All_Cover` that, given β and r, constructs a sparse (β, r)-neighborhood cover. The algorithm is built from two intermediate subprocedures: Procedure `Cluster` (see Figure 5), which grows a single set to be placed in the cover, and Procedure `Cover`, which calls `Cluster` to produce a subcollection $\mathcal{Y}$ of the sets in the cover $\mathcal{X}$ (see Figure 6). Procedure `Cover` produces what was earlier called a single phase, in the overview section.

The input to the algorithm is a graph $G = (V, \mathcal{E})$ (where $|V| = n$ and $|\mathcal{E}| = E$) and integers $r, \beta \geq 1$. The output collection of cover clusters, $\mathcal{X}$, is initially empty. The algorithm maintains the set of "remaining" vertices R. These are the vertices whose neighborhoods are not yet subsumed by the constructed cover. Initially, $R = V$, and the algorithm terminates once $R = \emptyset$. The code for Algorithm `All_Cover` appears in Figure 7.

The algorithm operates in (at most $\beta n^{1/\beta}$) phases. Each phase consists of the activation of Procedure `Cover`(R), which adds a subcollection of output clusters $\mathcal{Y}$ to $\mathcal{X}$ and removes the set of vertices $\Psi(R)$ whose r-neighborhood appears in some $Y \in \mathcal{Y}$ from R.

The collection of clusters $\mathcal{Y}$ returned by Procedure `Cover` are placed in the sparse neighborhood cover built by Algorithm `All_Cover`. The collection of clusters $\mathcal{Z}$ returned by Procedure `Cover` themselves form an *average degree cover* (see section 6).

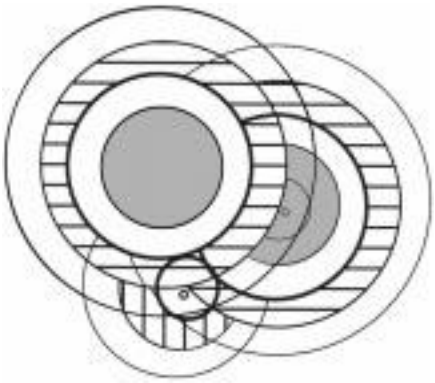


FIG. 4. *The growth of a third cluster. As before, the kernels $\Psi(Y)$ are shaded, the sets Y are marked with a bold line, $\Psi(Z)$ is striped, and Z is the outside ring.*

$\Psi(Z) \leftarrow \{v\}$
$Z \leftarrow \{N_r(v)\}$
repeat
  $\Psi(Y) \leftarrow \Psi(Z)$
  $Y \leftarrow Z$
  Perform a multiorigin BFS w.r.t. Y
    to depth $2r$ in $G(V \setminus \bigcup \mathcal{Y})$
  Add all vertices encountered to Z
  $\Psi(Z) \leftarrow \{v \mid v \in U \cap Z, \ \ \mathrm{dist}(v, Y) \leq r\}$
until $|Z| \leq n^{1/\beta} \cdot |Y|$
  and $|\Psi(Z)| \leq |R|^{1/\beta} \cdot |\Psi(Y)|$
  and $\lambda(Z) \leq n^{1/\beta} \cdot \lambda(Y)$
return $(\Psi(Y), Y, \Psi(Z), Z)$

FIG. 5. *Procedure* `Cluster`$(R, U, v, \mathcal{Y})$.

4. Analysis. In this section, we first analyze the properties of the procedures in section 3.4 and prove that Algorithm `All_Cover` constructs a sparse neighborhood cover. The complexity analysis then follows in section 4.2.

4.1. Correctness of the algorithm.

LEMMA 4.1. *A set Y produced by Procedure* `Cluster` *has diameter at most $O(\beta r)$.*

Proof. Recall the three stopping conditions on the growth of a cluster. If the first stopping condition fails to be met, this means that $|Z| \geq n^{1/\beta} \cdot |Y|$. Since each time the stopping condition fails we increase Y by a factor of $n^{1/\beta}$, this stopping condition can fail at most β times.

If the second stopping condition fails to be met, this means that $|\Psi(Z)| \geq |R|^{1/\beta} \cdot |\Psi(Y)|$. Note that whenever Procedure `Cluster` is invoked within Procedure `Cover`, we have $U \subseteq R$. Since each time the second stopping condition fails we increase $\Psi(Y)$ by a factor of $|R|^{1/\beta}$, and $\Psi(Y)$ is restricted to U, this stopping condition can again fail at most β times.

The third stopping condition says that $\lambda(Z)$ is less than or equal to $n^{1/\beta}$ times

$U \leftarrow R$
$\Psi(R) \leftarrow \emptyset$.
$\mathcal{Y}, \mathcal{Z} \leftarrow \emptyset$.
while $U \neq \emptyset$ **do**
  Select an arbitrary vertex $v \in U$
  $(\Psi(Y), Y, \Psi(Z), Z) \leftarrow$ `Cluster`$(R, U, v, \mathcal{Y})$
  $\Psi(R) \leftarrow \Psi(R) \cup \Psi(Y)$
  $U \leftarrow U \setminus \Psi(Z)$
  $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{Y\}$
  $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{Z\}$
end-while
return $(\Psi(R), \mathcal{Y}, \mathcal{Z})$

FIG. 6. *Procedure* `Cover`(R).

$R \leftarrow V$
$\mathcal{X} \leftarrow \emptyset$
repeat
  $(\Psi(R), \mathcal{Y}, \mathcal{Z}) \leftarrow$ `Cover`(R)
  $\mathcal{X} \leftarrow \mathcal{X} \cup \mathcal{Y}$
  $R \leftarrow R \setminus \Psi(R)$
until $R = \emptyset$
return $\mathcal{X}$

FIG. 7. *Algorithm* `All_Cover`.

$\lambda(Y)$, and since the sum of the degrees of the entire graph is at most $E \leq n^2$, the third stopping condition can successively fail at most 2β times.

Therefore, the number of iterations of the growth process for which *any* of the stopping conditions fail is at worst 4β, and so, for one of the first $4\beta + 1$ iterations, all three stopping conditions will hold, and the cluster will stop growing. Since each time we grow Y, we add an additional distance r to the radius, the diameter of the sets is bounded by $O(\beta r)$. □

LEMMA 4.2. *The r-neighborhood of any vertex in a cluster's set Y is contained in $\Psi(Z)$ for that cluster or some previous cluster's set $\Psi(Z)$.*

Proof. Let v be a vertex in the r-neighborhood of Y. Then v is certainly in the $2r$-neighborhood of Y, so Procedure `Cluster` places $v \in Z$ for that cluster. Procedure `Cluster` then makes $\Psi(Z) = \{v \mid v \in U \cap Z, \ \ \mathrm{dist}(v, Y) \leq r\}$, so if v is not placed in $\Psi(Z)$, it means v is not in U. Now we examine the set U as it is modified over each invocation of Procedure `Cover`. The set U initially contains the entire graph G, and Procedure `Cover` only deletes those vertices from U which lie in some previous cluster's set $\Psi(Z)$. □

LEMMA 4.3. *For every $w \in G$, each invocation of Procedure* `Cover` *places w into at most one set Y.*

Proof. The proof is by contradiction. Suppose w was placed in Y_1 and Y_2, and without loss of generality, let Y_1 be constructed by `Cover` before Y_2. Note that since $w \in Y_1$, then $w \in \Psi(Z)$ for Y_1. When `Cover` constructs Y_2, it picks some center vertex v and invokes Procedure `Cluster`. We consider three cases: either w is v, or

w is in the r-neighborhood of v, or w is not in the r-neighborhood of v. First, w cannot be v, since w is in $\Psi(Y)$ for Y_1, $\Psi(Y) \subseteq \Psi(Z)$, and thus Procedure `Cover` has removed w from the set U of vertices from which it chooses centers v. In fact, since v is outside $\Psi(Z)$ for all previous Y, its entire r-neighborhood cannot contain anything in a previous Y, by Lemma 4.2. Thus w is not in the r-neighborhood of v, and so it will not be placed into Y when `Cluster` is initialized. As `Cluster` grows cluster Y_2, by construction, it will only place vertices w into Y_2 that it first places in the Z layer. However, members of previous Y are barred from entering into any subsequent Z layer, by construction. □

LEMMA 4.4. *An $\Omega(|R|^{-1/\beta})$ fraction of the vertices whose r-neighborhood still remains to be covered (i.e., vertices in R) have their r-neighborhoods covered each time Algorithm* `All_Cover` *calls Procedure* `Cover`. *Thus Procedure* `Cover` *is invoked $O(\beta n^{1/\beta})$ times.*

Proof. The vertices that lie in $\Psi(Y)$ for some cluster Y are the vertices that have their r-neighborhoods covered. The stopping condition guarantees that for each cluster, an $\Omega(|R|^{-1/\beta})$ fraction of the vertices in $\Psi(Z)$ (in R) lie in $\Psi(Y)$. Since every vertex in R is in some $\Psi(Z)$ (by the termination condition for Procedure `Cover`), and since the Y sets, and hence the $\Psi(Y)$ sets, are all disjoint (by Lemma 4.3), summing over all clusters produced in one invocation of `Cover` gives the result (as $|R| \leq n$). Hence, after $O(\beta n^{1/\beta})$ iterations of Procedure `Cover`, all vertices have their r-neighborhoods covered, and Algorithm `All_Cover` returns the desired cover. □

We can now complete the correctness proof.

THEOREM 4.5. *Given a graph $G = (V, \mathcal{E})$ (where $|V| = n, |\mathcal{E}| = E$) and integer parameters $\beta, r \geq 1$, Algorithm* `All_Cover` *constructs a sparse neighborhood cover.*

Proof. By Lemma 4.4, Algorithm `All_Cover` succeeds in covering all r-neighborhoods after $O(\beta n^{1/\beta})$ iterations of Procedure `Cover`. All clusters in the cover are constructed by Procedure `Cluster`, and hence have low diameter, by Lemma 4.1. Finally, by Lemma 4.3, each vertex appears in at most one set for each invocation of Procedure `Cover`, and by Lemma 4.4, Procedure `Cover` is invoked at most $O(\beta n^{1/\beta})$ times. Thus each vertex is in at most $O(\beta n^{1/\beta})$ sets, and hence the cover is sparse. □

4.2. Complexity analysis. We compute the cost of checking the stopping conditions and then performing the appropriate BFS explorations to construct $\mathcal{Y}$. The stopping conditions need to be checked at most $O(\beta)$ times for each cluster, since the conditions will be met within $O(\beta)$ iterations. To check the stopping conditions, we count vertices in Z, for all $Z \in \mathcal{Z}$. Recall that one of the stopping conditions insures that for all Z the number of vertices in Z is less than $O(n^{1/\beta})$ times the number of vertices in Y. Therefore, $O(\sum_{Z \in \mathcal{Z}} |Z|) = O(n^{1/\beta} \sum_{Y \in \mathcal{Y}} |Y|) = O(n^{1+1/\beta})$, since the sets Y are disjoint. Thus, the cost of checking the stopping conditions to construct $\mathcal{Y}$ is $O(\beta n^{1+1/\beta})$.

Now we turn to estimating the cost of performing the BFSs. Ignore for the moment the cost of the BFS exploration of the final layer $Z \setminus \Psi(Z)$ for each cluster. Then for the construction of $\mathcal{Y}$, each vertex participates at most twice in BFSs: each edge is placed in at most one set Y, and prior to being placed in Y, it could lie in at most one $\Psi(Z)$. Thus, without the Z layer, each edge has been examined at most twice in the construction of $\mathcal{Y}$.

Now consider the complexity of examining edges in Z. Notice that when $r = 1$, this is trivial, since every edge in $Z \setminus \Psi(Z)$ has an endpoint in $\Psi(Z)$, and every vertex is in a unique $\Psi(Z)$. For $r > 1$, bounding the amount of work done in Z is controlled

by means of the third stopping condition on the growth of individual clusters. In particular, if we have done too much work on exploring the layer $Z \setminus Y$, we are then obligated to grow Y to contain Z.

Recall that the final stopping condition on clusters ensures $\lambda(Z)$ is less than or equal to $O(n^{1/\beta})$ times $\lambda(Y)$. Let Z_Y denote the set Z associated with cluster Y. Then this gives

$$\sum_{Y \in \mathcal{Y}} |Z_Y| \leq \sum_{Y \in \mathcal{Y}} O(n^{1/\beta})\lambda(Y) \leq O(n^{1/\beta})\lambda(G) \leq O(n^{1/\beta}E).$$

Constructing $\beta n^{1/\beta}$ subcollections $\mathcal{Y}$ to complete the cover $\mathcal{X}$, we thus obtain the following theorem.

THEOREM 4.6. *Algorithm* `All_Cover` *produces a sparse* (β, r)*-neighborhood cover in time* $O((E\beta + n\beta^2)n^{2/\beta})$.

Setting $\beta = \log n$, we obtain the following corollary.

COROLLARY 4.7. *Algorithm* `All_Cover` *produces a sparse* $(\log n, r)$*-neighborhood cover in time* $O(E \log n + n \log^2 n)$.

5. Application: Approximating *k*-pairs shortest paths. In this section we present an approximation algorithm for the k-pairs shortest paths problem on undirected graphs with nonnegative edge weights. We exhibit a tradeoff between the quality of the approximation and running time. Specifically, we obtain a 32β approximation to the k-pairs shortest paths problem in time $O((E\beta + n\beta^2)n^{2/\beta}\delta + k\beta n^{1/\beta} \log\log n)$, for integer $\beta \geq 1$. This means that we can achieve a better running time than known exact algorithms even for paths that are only a constant factor times the length of the shortest path. Further work in this direction is reported in [12].

For $\beta = \log n$ we get an $O(\log n)$ times optimal approximation algorithm for the problem that runs in $\tilde{O}(E + k)$ time. Finding approximate shortest paths for *all* pairs will take $\tilde{O}\left(E + n^2\right) = \tilde{O}\left(n^2\right)$ time.

5.1. Sparse tree covers. The explanation of the algorithm is made simpler through the notion of *sparse tree covers*. We give these definitions here for unit edge weights; the extension to nonnegative edge weights is straightforward, as explained later.

DEFINITION 5.1. *For an undirected graph* $G(V, \mathcal{E})$ *and integers* $\beta, r \geq 1$*, a* (β, r)-tree cover *is a collection* $\mathcal{F}_{\beta,r}$ *of trees in* G*, that satisfies the following properties.*

1. *Every tree* $F \in \mathcal{F}_{\beta,r}$ *has depth* $8\beta r$ *or less.*
2. *For every two vertices* $u, v \in V$ *whose distance in* G *is* r *or less, there exists a common tree* $F \in \mathcal{F}_{\beta,r}$*, containing both.*

A (β, r)*-tree cover is said to be* sparse*, if each vertex is in at most* $\beta n^{1/\beta}$ *sets.*

Note that it is easy to modify Algorithm `All_Cover` so that it produces a sparse tree cover, not just a sparse neighborhood cover. In fact, Procedure `Cluster` actually constructs a BFS spanning tree for each cluster it builds, so the main change necessary is to include these trees in the output. The resulting (β, r)-tree cover will have the property that each tree in it is a BFS spanning tree of a set in the corresponding (β, r)-neighborhood cover. The time complexity remains $O((E\beta + n\beta^2)n^{2/\beta})$, which is $O(E \log n + n \log^2 n)$ when we set $\beta = \log n$.

5.2. The algorithm. We first give our algorithm for approximating k-pairs shortest paths with unit edge weights, and later explain how to extend this to the case when the edge weights are nonnegative.

Preprocessing. Let $\delta = \lceil \log \mathrm{Diam}(G) \rceil$. For every level $1 \leq i \leq \delta$, construct a sparse tree cover $\mathcal{F}_{\beta,2^i}$ for G, and number the trees in the cover. For every vertex v, and for every level i, store (in order of increasing tree ID) a list of all trees $F \in \mathcal{F}_{\beta,2^i}$ that contain v.

Query response. For any given pair of vertices u, v, for which a shortest path is sought, do the following.

Perform a binary search over the levels $1 \leq i \leq \delta$: for a given level i, if there exists a tree $F \in \mathcal{F}_{\beta,2^i}$ that contains both u and v, then restrict the search to lower values of i; otherwise, restrict the search to higher values of i.

The binary search will produce a level J such that $2^{J-1} \leq d(u,v) \leq 2^J 16\beta$, and a tree $F \in \mathcal{F}_{\beta,2^J}$ that contains both u and v.

Return the value $16\beta 2^J$ as the approximate distance between u and v. (For the ESP version of the problem, also return the unique path connecting a pair u, v in the common tree F as the approximate shortest path between u and v.)

The weighted case. The algorithms for sparse neighborhood covers presented in section 3 can easily be modified to produce a *weighted* tree cover by forming shortest path trees rather than BFS trees. The algorithm presented in this section can in turn be trivially modified to handle weights. As we will see below, the running time of our algorithms remains asymptotically unchanged.

5.3. Analysis. First we wish to prove that our algorithm is an $O(\beta)$-approximation algorithm for the shortest paths problem.

LEMMA 5.2. *For each of the k pairs of vertices given, our algorithm returns a path length between them within 32β times the length of the shortest path.*

Proof. We will argue this for a given pair u, v. Note that by Definition 5.1, if $\mathrm{dist}(u,v) \leq 2^i$, then the tree cover of level i has a tree containing both u and v. Since J is the minimum level i for which this is so, it must be that $2^{J-1} < \mathrm{dist}(u,v) \leq 2^J$. Also by Definition 5.1, the common tree has depth at most $8\beta \cdot 2^J < 16\beta \cdot \mathrm{dist}(u,v)$. That is, our algorithm returns the maximum length of the unique path connecting u and v in the common tree: $16\beta \cdot 2^J < 32\beta \cdot \mathrm{dist}(u,v)$. ∎

Now we address the running time of our algorithm.

LEMMA 5.3. *Our algorithm for approximating k-pairs shortest paths takes*

$$O((E\beta + n\beta^2)n^{2/\beta}\delta + k(\beta n^{1/\beta} \cdot \log \delta))$$

time. For the ESP version, the complexity increases (inherently) by the total length of the paths.

Proof. We have already noted that the algorithm for sparse neighborhood covers in this paper can produce the tree cover as it goes along at no additional payment in time. As we actually set up a data structure consisting of δ different tree covers, one for each level, the time bound for setting up the data structure is $O((E\beta + n\beta^2)n^{2/\beta}\delta)$ by Theorem 4.6.

As for the bound on a single query, the binary search for finding the right level requires checking $O(\log \delta)$ levels, and for a given level i, a common tree F can be found in $\beta n^{1/\beta}$ time by searching the ordered lists of tree IDs to which the two vertices belong. (Recall that by the sparsity property, every vertex belongs to at most $\beta n^{1/\beta}$ different trees in the ith-level tree cover $\mathcal{F}_{\beta,2^i}$.)

For the *ESP* version, retrieving the unique path connecting a pair u, v in the common tree F takes time proportional to the length of that path. ∎

Lemmas 5.2 and 5.3 yield the following theorem.

THEOREM 5.4. *Our algorithm for approximating k-pairs shortest paths takes* $O((E\beta + n\beta^2)n^{2/\beta}\delta + k(\beta n^{1/\beta} \cdot \log \delta))$ *time, and returns a solution that is within* $O(\beta)$ *of the exact one.*

Setting $\beta = \log n$, we get the following corollary.

COROLLARY 5.5. *Our algorithm provides an* $O(\log n)$ *approximation to the k-pairs shortest paths problem in* $O((E \log n + n \log^2 n)\delta + k(\log n \cdot \log \delta))$ *time.*

For the *all*-pairs shortest paths problem we get the following corollary.

COROLLARY 5.6. *Our algorithm for approximating* all*-pairs shortest paths takes* $O((E\beta + n\beta^2)n^{2/\beta}\delta + n^2(\beta n^{1/\beta} \cdot \log \delta))$ *time, and returns a solution that is within* $O(\beta)$ *of the exact one.*

And again, setting $\beta = \log n$, we have the following corollary.

COROLLARY 5.7. *Our algorithm provides an* $O(\log n)$ *approximation to the all-pairs shortest paths problem in* $O((E \log n + n \log^2 n)\delta + n^2(\log n \cdot \log \delta))$ *time.*

6. Discussion. We close the paper by discussing two final points. First, the construction algorithm described in this paper can be adapted to work in a (synchronous or asynchronous) distributed network. The asynchronous implementation requires additional techniques in order to guarantee maintaining a bound of $\tilde{O}(1)$ messages per edge on the message complexity. Roughly speaking, a synchronous distributed variant of the cover construction algorithm needs to be bootstrapped concurrently with the synchronizer protocol of [8] in a mutually recursive fashion. This distributed implementation will be described in full elsewhere; concise description of the essential details can be found in [8, 7, 6]. Let us comment that this distributed algorithm implies similar improvements in message complexity to a variety of distributed applications that make use of sparse covers, including adaptive routing schemes and a distributed *from-scratch* BFS and network synchronizer construction.

Second, it is worth mentioning that there exists a somewhat weaker notion of neighborhood covers: a small *average degree cover* [23, 9], where the *average* cluster overlaps over all vertices is small. In contrast, our neighborhood cover algorithm produces a cover with small *maximum* cluster that overlaps over all vertices. The average degree cover problem is known to be significantly easier [23, 9]; in fact, the maximum degree cover algorithm in this paper can be thought of as constructed through a logarithmic number of iterations, each producing an average degree cover in the "remaining graph."

Average degree covers are not sufficient for an efficient solution to the k-pairs shortest paths problem, because the "imbalance" (of high and low overlaps) cannot be amortized over many queries. (Unlike the algorithms in this paper, other constructions of average degree covers [23, 12] cannot be run for multiple iterations to produce a small maximum degree cover.)

However, as pointed out in [1, 12], in the special case of the *all*-pairs shortest paths application, an average degree cover suffices and leads to a better tradeoff between running time and approximation; namely, the constant of approximation can be improved. Specifically, the approximation constant of 32β derived earlier can be improved in this case to 4β by running our sparse neighborhood cover algorithm and taking the $2r$-neighborhoods of the sets constructed in one iteration, thereby producing a sparse average degree cover, and by eliminating two of the stopping conditions of the algorithm. Cohen [12] observed that the extra stopping condition can be removed in this case, and she also introduced a randomized parallel algorithm for approximate all-pairs shortest paths. The randomized algorithm constructs *pairwise covers* along the lines of [21]. Pairwise covers satisfy the weaker condition that any

pair of points at distance $\leq r$ lie together in the same cluster (but 3 points in the same r-neighborhood may not all appear together in any one cluster).

Acknowledgments. We thank Yehuda Afek, Edith Cohen, Tom Leighton, and Moti Ricklin for stimulating discussions.

REFERENCES

[1] Y. Afek and M. Ricklin, *Sparser: A paradigm for running distributed algorithms*, in Proc. 6th Workshop on Distributed Algorithms, Springer-Verlag, New York, 1992, pp. 1–10.

[2] R. K. Ahuja, K. Melhourn, J. B. Orlin, and R. E. Tarjan, *Faster Algorithms for the Shortest Path Problem*, Technical Report 193, MIT Operations Research Center, Cambridge, MA, 1988.

[3] N. Alon, Z. Galil, and O. Margalit, *On the exponent of the all-pairs shortest path problem*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1991.

[4] N. Alon, Z. Galil, O. Margalit, and M. Naor, *Witnesses for boolean matrix multiplication and for shortest paths*, In Proc. 33rd IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1992.

[5] B. Awerbuch, A. Baratz, and D. Peleg, *Efficient Broadcast and Light-Weight Spanners*, Unpublished manuscript, 1991.

[6] B. Awerbuch, B. Patt-Shamir, D. Peleg, and M. Saks, *Adapting to asynchronous dynamic networks with polylogarithmic overhead*, in Proc. 24th ACM Symp. on Theory of Computing, ACM, New York, 1992, pp. 557–570.

[7] B. Awerbuch and D. Peleg, *Efficient Distributed Construction of Sparse Covers*, Technical Report CS90-17, The Weizmann Institute, Rehovot, Israel, 1990.

[8] B. Awerbuch and D. Peleg, *Network synchronization with polylogarithmic overhead*, in 31st IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1990, pp. 514–522.

[9] B. Awerbuch and D. Peleg, *Sparse partitions*, in 31st IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1990, pp. 503–513.

[10] P. A. Bloniarz, *A Shortest Path Algorithm with Expected Time* $o(n^2 \log n \log^* n)$, Technical Report 80-3, Dept. of Computer Science, State University of Albany, New York, 1980.

[11] B. Chandra, G. Das, G. Narasimhan, and J. Soares, *New sparseness results on graph spanners*, in Proc. 8th ACM Symp. on Computational Geometry, ACM, New York, 1992.

[12] E. Cohen, *Fast algorithms for constructing* t*-spanners and paths with stretch* t, in Proc. 34th IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1993, pp. 648–658.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press/McGraw-Hill, New York, 1990.

[14] M. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. Assoc. Comput. Mach., 34 (1987), pp. 596–615.

[15] A. M. Frieze and G. R. Grimmet, *The shortest-path problem for graphs with random arc-lengths*, Disc. Appl. Math., 10 (1985), pp. 57–77.

[16] H. Gabow, *Scaling algorithms for network problems*, J. Comput. System Sci., 31 (1985), pp. 148–168.

[17] H. Jakobsson, *Mixed-approach algorithms for transitive closure*, in Proc. 10th ACM Symp. on Principles of Database Systems, ACM, New York, 1991.

[18] D. Johnson, *Efficient algorithms for shortest paths in sparse networks*, J. Assoc. Comput. Mach., 24 (1977), pp. 1–13.

[19] D. R. Karger, D. Koller, and S. J. Phillips, *Finding the hidden path: Time bounds for all-pairs shortest paths*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, IEEE, Piscataway, NJ, 1991.

[20] P. N. Klein, *A Parallel Randomized Approximation Scheme for Shortest Paths*, Technical Report CS-91-56, Brown University, Providence, RI, 1991.

[21] N. Linial and M. Saks, *Decomposing graphs into regions of small diameter*, in Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms, SIAM, Philadelphia, 1991, pp. 320–330.

[22] C. C. McGeoch, *A New All-Pairs Shortest-Path Algorithm*, Technical Report 91-30, DIMACS, New Brunswick, NJ, 1991.

[23] D. Peleg, *Sparse Graph Partitions*, Technical Report CS89-01, The Weizmann Institute, Rehovot, Israel, 1989.

[24] S. Rao, *Finding small edge cuts in planar graphs*, in Proc. 24th ACM Symp. on Theory of Computing, ACM, New York, 1992, pp. 229–240.
[25] R. Seidel, *On the all-pairs-shortest-path problem*, in Proc. 24th ACM Symp. on Theory of Computing, ACM, New York, 1992, pp. 745–749.
[26] P. M. Spira, *A new algorithm for finding all shortest paths in a graph of positive arcs in average time* $O(n^2 \log^2 n)$, SIAM J. Comput., 2 (1973), pp. 28–32.
[27] J. D. Ullman and M. Yannakakis, *High-probability parallel transitive closure algorithms*, SIAM J. Comput., 20 (1991), pp. 100–125.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 278–296

UNORIENTED Θ-MAXIMA IN THE PLANE: COMPLEXITY AND ALGORITHMS*

DAVID AVIS†, BRYAN BERESFORD-SMITH‡, LUC DEVROYE†, HOSSAM ELGINDY‡, ERIC GUÉVREMONT§, FERRAN HURTADO¶, AND BINHAI ZHU‖

Abstract. We introduce the unoriented Θ-maximum as a new criterion for describing the shape of a set of planar points. We present efficient algorithms for computing the unoriented Θ-maximum of a set of planar points. We also propose a simple linear expected time algorithm for computing the unoriented Θ-maximum of a set of planar points when $\Theta = \pi/2$.

Key words. maxima, plane sweep, lower bound, probabilistic analysis, expected complexity

AMS subject classifications. 68Q25, 60D05, 60C05

PII. S0097539794277871

1. Introduction. The development of image processing has motivated the investigation of properties of point sets for the purpose of image classification and/or understanding. Connectivity graphs and various enclosing boundary sets have been used to characterize the shape of point sets. The minimum spanning tree, the Gabriel graph, and the Delaunay triangulation are important connectivity graphs. Convex, maximal, and α-hulls [KLP75, EKS83] are instances of boundary sets. The kth iterated hull [Ch85] and the related concept of k-hull [CSY87] have also been proposed.

In this paper we introduce unoriented Θ-maxima as a generalization of extreme and maximal vectors. These are useful as boundary descriptors, and remain invariant under rotation.

Let S be a set of n planar points. A ray from a point $p \in S$ is the collection of all points $\{p + \lambda(v - p) : \lambda > 0\}$, where v is a fixed point in the plane not equal to p. A ray from a point $p \in S$ is called a *maximal ray* if it passes through another point $q \in S$. A cone is defined by a point p and two rays A and B emanating from it: it is the convex set $\{\lambda u + (1-\lambda)v : u \in A, v \in B, \lambda \in [0,1]\}$. A point $p \in S$ is said to be a *maximum* (or *maximal*) with respect to S if there exist two rays, A and B, emanating from p such that A and B are parallel to the $+x$- and $+y$-axes, respectively (thus, $v = p + (1,0)$ and $v = p + (0,1)$ in the definition of A and B), and the points of S lie outside the ($\pi/2$-angle) cone defined by p, A, and B. A point $p \in S$ is an unoriented Θ-maximum with respect to S if and only if there exist two *maximal* rays, A and B, emanating from p with an angle at least Θ between them so that the points of S lie outside the (Θ-angle) cone defined by p, A, and B (see Figure 1). We let S_Θ denote the subset of S whose elements are unoriented Θ-maxima. For the remainder of this

*Received by the editors December 1, 1994; accepted for publication (in revised form) December 30, 1996; published electronically June 15, 1998.
http://www.siam.org/journals/sicomp/28-1/27787.html

†School of Computer Science, McGill University, Montreal, QC H3A 2A7, Canada (avis@cs.mcgill.ca, luc@cs.mcgill.ca).

‡Dept. of Computer Science, The University of Newcastle, NSW, Australia (bbs@cs.newcastle.edu.au, hossam@cs.newcastle.edu.au).

§School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada.

¶Dept. de Mathemàtica Aplicada II, Universitat Politècnica de Catalunya, Barcelona, Spain (hurtado@ma2.upc.es).

‖Los Alamos National Laboratory, Group C-3, Los Alamos, NM 87545 and Dept. of Computer Science, City University of Hong Kong, Kowloon, Hong Kong (bhz@cs.cityu.edu.hk).

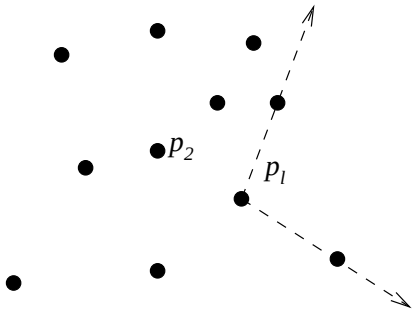


FIG. 1. *Point p_1 is an unoriented $\pi/2$-maximum whereas p_2 is not.*

paper, we only consider the problem of computing $S_{\pi/2}$. The algorithms apply to other values of $\Theta > \pi/2$. The additional technique for handling values of $\Theta < \pi/2$ is discussed in the appendix.

For each $p \in S_{\pi/2}$, our algorithms report two witnesses in the form of two maximal rays with an included angle, denoted by $\alpha_p \geq \pi/2$. Each of these maximal rays intersects the same edge of the convex hull of S and contains a ray parallel to either the x- or the y-axis in the cone between the two maximal rays. The above properties lead to two different approaches for computing $S_{\pi/2}$, which we outline in the following paragraphs.

1.1. Convex hull approach. The following geometric properties of $S_{\pi/2}$ (to be proven later) form the pillars of this approach and allow for a reduction of the problem into simpler tasks equal in number to the convex hull of S.

1. For each point $p \in S_{\pi/2}$, there exist two maximal rays emanating from p which intersect the same edge of the convex hull of S and such that the points of S lie outside the $\pi/2$-cone between the two rays.
2. For each point $p \in S_{\pi/2}$ there exist no more than *three* pairs of maximal rays which satisfy the previous property.
3. A pair of maximal rays which satisfies the first property includes the perpendicular from p to the corresponding convex hull edge in the $\pi/2$-cone between them.

The first task involves reporting unoriented maxima whose corresponding maximal rays intersect the same convex hull edge of S, and the other two properties facilitate the use of efficient computational geometry tools to develop an optimal running time algorithm. A detailed description of this approach is given in section 2.

1.2. Restricted unoriented maximum approach. This approach is based on the following simple property: for each point $p \in S_{\pi/2}$ there exist two maximal rays emanating from p which contain the $+x$-, the $-x$-, the $+y$-, or the $-y$-axis in the $\pi/2$-angle cone between them.

The problem is thus reduced to reporting for each of the four (directed) axes the unoriented maxima whose corresponding maximal rays contain it. For each axis, e.g., the $+y$-axis, we first sort points of the set in the direction perpendicular to the selected axis. We then perform two more linear passes. In the first pass, we scan the points of S from left to right constructing the convex hull of the visited points. Before $p \in S$ is processed, we compute the empty angle between the tangent from p to the convex hull and the selected axis, and call it θ. Perform a similar pass from right to left, storing the angle at p in ξ. A simple geometric argument shows that with respect

to the selected axis a point $p \in S_{\pi/2}$ if and only if $\theta + \xi \geq \pi/2$.

It is natural to observe the similarity between the two approaches. However, the restricted unoriented maxima (RUM) approach is more suitable for handling the discrete versions of the problem, namely, answering unoriented maximum queries, and identifying unoriented maxima of a set in parallel models of computation. This follows from the fact that focusing on a particular direction allows for the use of the divide-and-conquer technique with an efficient merging process. Moreover, the RUM approach is more suitable for probabilistic analysis. A detailed description and the probabilistic analysis of this approach is given in section 4.

The rest of the paper is organized as follows. Section 2 is dedicated to the details of computing unoriented $\pi/2$-maxima for a given set of planar points. In section 3 a lower bound for the algebraic computation tree model is developed, which implies that our algorithm is optimal. Finally, in section 4 the expected number of unoriented $\pi/2$-maxima is analyzed (and used) to obtain a linear expected running time algorithm. In conclusion, we discuss an approach for handling arbitrary values of Θ and some related results and unsolved problems.

2. Computing unoriented $\pi/2$-maxima. Let $S = \{X_1, X_2, \ldots, X_n\}$ denote a set of n planar points in general position (no three points are collinear). Its convex hull $CH(S)$ is the pair $(V(S), E(S))$, where $V(S)$ is the set of vertices and $E(S)$ is the set of edges. We denote the size of the convex hull by h ($h = |V(S)| = |E(S)|$). A point $p \in S - V(S)$ is called a *candidate* for an edge $e \in E(S)$ if there exist two rays emanating from p with a $\pi/2$-angle cone between them which intersect the edge e. Clearly, a point which is an unoriented maximum must be a candidate for some edge of the convex hull, and all convex hull points are candidates. From now on, we pay attention to candidate points that are not on the convex hull. To report the elements of the set $S_{\pi/2}$, based on the convex hull approach, we first identify the *candidates* for each edge of $E(S)$; then we consider each subset separately and check whether a candidate is a true unoriented $\pi/2$-maximum (i.e., whether the $\pi/2$-cone defined by the candidate is empty or not). The following geometric properties of candidates are critical to the efficiency of our algorithm.

LEMMA 1. *Each point $p \in S - V(S)$ may be a candidate for at most three edges of $E(S)$.*

Proof. The circular angle around p is 2π and, moreover, the points are in general position. Therefore, if p is the candidate for more than three edges, then one of the cones must have angle less than $\pi/2$. □

LEMMA 2. *If point p is a candidate for the edge $e \in E(S)$, then p lies in the semicircle of diameter e which has a nonempty intersection with the interior of the polygon defined by $E(S)$.*

Proof. The proof is elementary and omitted. □

Therefore, we have h semicircles with the constraint that no point in $S - V(S)$ belongs to more than three semicircles. In the following subsection, we establish a linear bound on the number of intersections of such curves. Algorithms for identifying candidates for each edge and for reporting unoriented maxima are then presented in sections 2.2 and 2.3, respectively.

2.1. A combinatorial property of constrained circles. Let $C^{(h)} = \{C_1, \ldots, C_h\}$ be a set of h planar circles with the constraint that no point in the plane belongs to more than k circles ($k \leq h$). Let x_i and r_i denote, respectively, the center and the radius of the ith circle C_i.

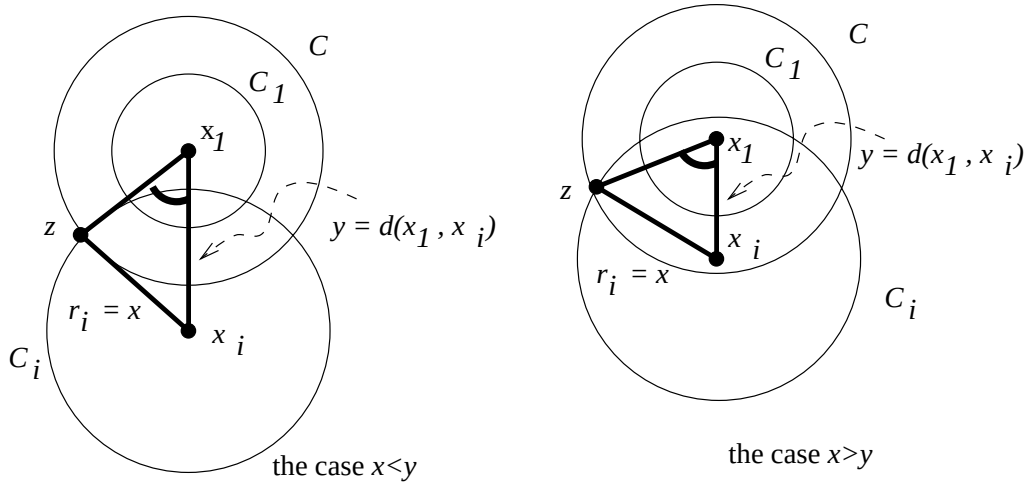


FIG. 2. *Finding a bound on θ.*

Without loss of generality, we assume that C_1 is the circle of $C^{(h)}$ with the smallest radius and that $r_1 = \min_{1 \le i \le h}\{r_i\} = 1$.

LEMMA 3. *At most $k-1$ circles can have their centers inside C_1.*

Proof. C_1 is the circle with smallest radius. Therefore, any circle C_i having x_i inside C_1 must contain x_1. Since x_1 cannot belong to more than k circles, the lemma follows. □

Let C be the circle concentric to C_1 with radius $\sqrt{3}$, let D_i be the disc consisting of circle C_i with its interior, and let the arc A_i be the intersection of D_i and the boundary of C if it exists. It is easy to see that any circle in $C^{(h)} - C_1$ that intersects C_1 and has a center outside C_1 must intersect C. The following lemma is based on Avis and Horton [AH81].

LEMMA 4. *If C_i intersects C_1 and x_i lies outside of C_1, then A_i subtends an angle of at least $\pi/3$ radians.*

Proof. Refer to Figure 2 for illustration. Let x, y and $\theta = \angle x_i x_1 z$ be as shown in Figure 2. Since C_i intersects C_1 and x_i lies outside of C_1, $y = d(x_1, x_i) \le r_1 + r_i = 1+x$. Therefore, we have $1 \le x, y \le 1+x$ and $\cos\theta = (3+y^2-x^2)/(2\sqrt{3}y)$; elementary geometry shows that $\cos\theta \le \sqrt{3}/2$. Therefore, $\theta \ge \pi/6$ radians, and thus the lemma follows. □

THEOREM 1. *At most $7k$ circles can intersect C_1.*

Proof. No point of the plane can belong to more than k circles. Therefore, Lemma 4 implies that no more than $6k$ circles can intersect C_1 and have their center outside C_1. Also, Lemma 3 implies that no more than k circles can intersect C_1 and have their center inside C_1. □

COROLLARY 1. *$C^{(h)}$ induces at most $14kh$ intersection points.*

Proof. Theorem 1 implies that C_1 can have at most $14k$ intersection points. By an inductive argument (removing C_1 from $C^{(h)}$ to obtain $C^{(h-1)}$), we can conclude that $C^{(h)}$ induces at most $14kh$ intersection points. □

It is clear that Corollary 1 holds for semicircles too.

COROLLARY 2. *In the arrangement of semicircles that was introduced in Lemma 2, there are at most $42h$ intersection points.*

Proof. The proof follows from Corollary 1 and Lemma 1. □

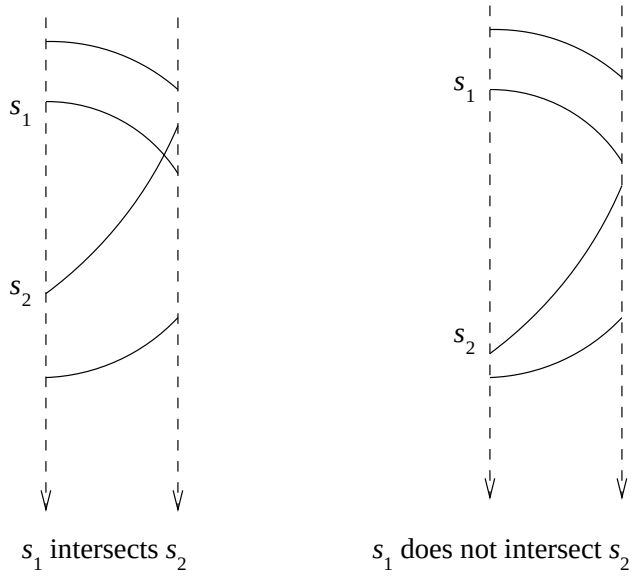


FIG. 3. *An example illustrating the idea of plane sweep.*

In this subsection, we showed that there are only a linear number of intersections among all the semicircles. We show in the next subsection how to apply this result to report candidates with respect to each hull edge.

2.2. Reporting candidates for each hull edge. In this section we will describe a procedure for reporting the candidates for $E(S)$. The procedure is based on the plane sweep technique of Bentley and Ottmann [BO79]. The idea of plane sweep can be described with the following simple example. Assume that we have two segments s_1, s_2, and without loss of generality, assume that the x-coordinates of the left (right) endpoints of s_1, s_2 are the same. The problem is to decide whether s_1 intersects s_2. We can see that s_1 intersects s_2 if and only if the order from top to bottom of the y-coordinates of the right endpoints of s_1, s_2 differs from the top-to-bottom order of the y-coordinates of the left endpoints of s_1, s_2. In general, the plane sweep method maintains a total order of some geometric objects (e.g., $O(n)$ segments) at a given stage. To check certain properties of two valid objects (e.g., whether s_1 intersects s_2), it simply checks whether the top-to-bottom order of these two objects switches at a later stage. Usually a dynamic balanced binary search tree is sufficient for the plane sweep method (to maintain the total order) [BO79]. In Figure 3 we illustrate an example for plane sweep for some xy-monotone (i.e., monotone in both the x- and y-directions) circular segments.

First we give a description of the procedure and then explain the essential details and analyze its correctness and performance.

PROCEDURE CANDIDATES

INPUT: A set S of n planar points.

OUTPUT: The list of edges of $E(S)$ together with a list of candidate points for each edge.

METHOD:

1. Compute the convex hull of S and store the edges of $CH(S)$, $E(S)$ in a doubly linked list.
2. Compute the semicircles having as diameters the edges of $E(S)$.
3. Partition each semicircle into at most three parts such that every (circular) segment produced is xy-monotone. Let H be the set of segments obtained (note that $|H| \in O(n)$).
4. Apply the Bentley and Ottmann [BO79] plane sweep algorithm on $H \cup (S - V(S))$ to report the intersection points of the monotone segments in H. When a point $p \in S - V(S)$ is met by the sweep line, an $O(\log n)$ search in a balanced search data structure T may be used to identify those edges of $CH(S)$ for which p is a candidate. At the end of this step, all candidates of $S - V(S)$ are known.
5. Produce the list of candidates for each edge of $E(S)$ using the output of step 4.

End of Procedure

Correctness of Procedure Candidates in computing the intersection points of the elements in $H \cup (S - V(S))$ follows directly from correctness of the sweep line algorithm in [BO79]. Computing such intersections is essential to maintaining a total vertical ordering of the segments in a search structure T where the following four operations can be implemented in $O(\log n)$ time.

1. `INSERT`(s, T) inserts the segment s into the total order maintained by T.
2. `DELETE`(s, T) deletes segment s from T.
3. `ABOVE`(s, T) returns the name of the segment immediately above s in T.
4. `BELOW`(s, T) returns the name of the segment immediately below s in T.

These operations are listed in [SH76] and referred to by [BO79]. They can be implemented using a balanced binary search tree.

For a given vertical sweep line L, T contains the total ordering of the monotone segments (of semicircles) intersecting L. They define vertical intervals on L, each of which corresponds to a unique intersection region. We modify the balanced search tree by keeping for each vertical interval (uniquely determined by two adjacent elements of H) the list of semicircles containing that segment. By Lemma 1, at most three such semicircles may exist. Therefore, the space complexity of the data structure is still linear. When a new semicircle is encountered (and two monotone segments are to be inserted), we use the information in its neighbor vertical intervals to establish its linked list. A deletion of a semicircle can be handled similarly. Finally, when an intersection point of two segments of semicircles is encountered, the appropriate linked list can be updated in constant time. Handling point $p \in S - V(S)$ requires performing a search of the structure T which returns the vertical interval that contains p. We can then determine the semicircles that contain p in constant time, and update the list of candidates for each of the corresponding convex hull edges in $E(S)$.

Step 1 can be done in $O(n \log n)$ time, and steps 2, 3, and 5 can be accomplished in $O(n)$ time. The Bentley and Ottmann [BO79] algorithm has an $O(n \log n + k \log n)$ running time, where k is the number of intersection points to be reported. Since we have $O(n)$ intersection points by Corollary 2, the execution time of step 4 is $O(n \log n)$. Therefore, Procedure Candidates reports the set of candidates for the convex hull edges in $O(n \log n)$ time and $O(n)$ space.

2.3. Computing unoriented maxima among candidates. Given the output of Procedure Candidates (i.e., a set of candidates for each edge of $E(S)$), we now

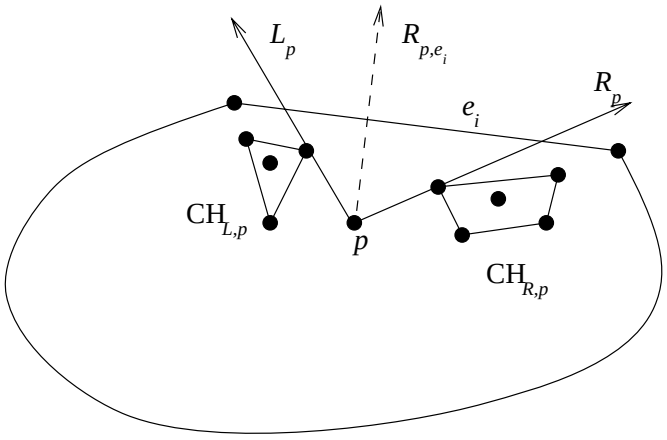


FIG. 4. *Computing unoriented maxima from candidates.*

develop a procedure to identify for each convex hull edge the unoriented maxima among its list of candidates.

Let $\mathcal{C}_i$ denote the set of candidates for the ith edge e_i of $E(S)$, and let k_i denote its size. If $p \in \mathcal{C}_i$, then the ray emanating from p and perpendicular to the edge e_i, denoted by R_{p,e_i}, properly intersects e_i since p lies inside the semicircle of diameter e_i. Let *wedge*(p, e_i) be the largest angle at p which does not contain points of S and is bounded by two maximal rays, L_p and R_p, that emanate from p and intersect e_i (Figure 4).

LEMMA 5. *If point p is an unoriented maximum with respect to edge e_i, then R_{p,e_i} must belong to the cone defined by p and the maximal rays L_p and R_p.*

Proof. If R_{p,e_i} does not lie between the maximal rays L_p and R_p, then *wedge*$(p, e_i) < \pi/2$, a contradiction. □

LEMMA 6. *If the convex hulls of points in $\mathcal{C}_i - \{p\}$ to the left and to the right of R_{p,e_i}, denoted by $CH_{L,p}$ and $CH_{R,p}$ respectively, are known, then we can compute wedge(p, e_i) in $O(\log n)$ time.*

Proof. Refer to Figure 4. Our problem is to compute the rightmost and leftmost (maximal) rays, R_p and L_p, emanating from p, intersecting e_i, and containing R_{p,e_i} in the cone (p, L_p, R_p). R_p (L_p) can be computed by finding the ray from p tangent to the convex hull to the right (left) of R_{p,e_i}, which can be done in $O(\log n)$ time [PS85]. If the angle between R_p and L_p (defined by the cone containing R_{p,e_i}) is $\geq \pi/2$, then p is an unoriented maximum. □

PROCEDURE UNORIENTED MAXIMA

INPUT: A list of candidates for the ith edge of *E(S)*.

OUTPUT: The unoriented maximal points and the rays defining their widest angles.

METHOD:

1. Sort the k_i points of $\mathcal{C}_i$ along e_i. Note that the sorted points define a simple polygonal chain.
2. Compute L_p for all points $p \in \mathcal{C}_i$ as follows:
   - $CH_L \leftarrow$ endpoint of e_i
   - Going from left to right using the order of the points of step 1:
     - Compute L_p using CH_L (as explained in Lemma 6).

– Insert p in CH_L using the rules of the convex hull algorithm of Avis, ElGindy, and Seidel [AES85].

3. Compute R_p for all points $p \in \mathcal{C}_i$ in a similar fashion to step 2, by scanning them from right to left.
4. For each $p \in \mathcal{C}_i$, compute angle *wedge*(p, e_i) between L_p and R_p, and if $\alpha_p \geq \pi/2$, output (p,L_p,R_p).
5. Output $V(S)$.

End of Procedure

Correctness of the above procedure follows from the correctness of the on-line convex hull algorithm in [AES85] and from Lemma 6.

Step 1 is performed in $O(k_i \log k_i)$ time. Since the algorithm in [AES85] updates the convex hull of k_i points by insertion in $O(\log k_i)$ time, and since searching for L_p and R_p requires $O(\log k_i)$ time at most as explained in Lemma 6, then steps 2 and 3 require $O(k_i \log k_i)$ time. Step 4 is clearly performed in $O(k_i)$ time, hence $O(k_i \log k_i)$ total time is spent for edge e_i. Lemma 1 implies that $\sum_{i=1}^{h} k_i \log k_i \leq \log n \sum_{i=1}^{h} k_i \leq 3n \log n \in O(n \log n)$. Therefore, we can state the final result of this section as follows.

THEOREM 2. *All unoriented maximal points of S can be computed in $O(n \log n)$ time and $O(n)$ space.*

In the next section, we establish an $\Omega(n \log n)$ lower bound for computing unoriented maxima in the plane, thus proving that our algorithm is optimal.

3. Lower bound for the algebraic computation tree model. In this section we establish an $\Omega(n \log n)$ lower bound for computing unoriented Θ-maxima in the plane. This $\Omega(n \log n)$ lower bound for computing the unoriented maxima $S_\Theta \subseteq S$ in the plane, for $\pi/2 \leq \Theta \leq \pi$, is achieved by a reduction from the integer element uniqueness problem. Note that when $\Theta \geq \pi$, the unoriented maxima $S_\Theta \subseteq S$ are exactly the convex hull (extreme) points, and it is well known that computing the extreme points of a set of n points has a lower bound of $\Omega(n \log n)$ under the algebraic computation tree model [PS85]. Our result is as follows.

THEOREM 3. *The problem of computing $S_\Theta \subseteq S$ for $\pi/2 \leq \Theta \leq \pi$ is $\Omega(n \log n)$ under the algebraic computation tree model, where $|S| = n$.*

Proof. We use a reduction from integer element uniqueness. In Yao [Ya89] this problem is shown to have a lower bound of $\Omega(n \log n)$ under the algebraic computation tree model.

We are given a set of integers $M = \{x_1, \ldots, x_n\}$, input to the integer element uniqueness problem. For each x_i, produce the following six points: $(i + \epsilon, (nx_i)^2)$, $(i+\epsilon, (nx_i)^2+\epsilon)$, $(i+\epsilon, (nx_i)^2-\epsilon)$, $(i-\epsilon, (nx_i)^2)$, $(i-\epsilon, (nx_i)^2+\epsilon)$, and $(i-\epsilon, (nx_i)^2-\epsilon)$. The value of $\epsilon = 1/4$ is used for our proof. Let S be the set containing all of these points.

If $x_i = x_j$ then at least two out of the twelve induced points cannot be unoriented maxima (Figure 5). On the other hand, if x_i is unique in M, then the six points created for x_i are all unoriented maxima. Hence all x_i's in M are distinct if and only if there are exactly $6n$ unoriented maxima in S. We have thus reduced integer element uniqueness to computing the unoriented maxima in linear time. Since the integer element uniqueness problem has a lower bound of $\Omega(n \log n)$ under the algebraic computation tree model, the theorem follows. □

We have thus obtained an optimal algorithm for computing unoriented Θ-maxima in the plane. In the next section we present the RUM algorithm which will beat the $\Omega(n \log n)$ lower bound when the points are drawn from a common distribution. This is obtained via a careful probabilistic analysis of the expected number of unoriented

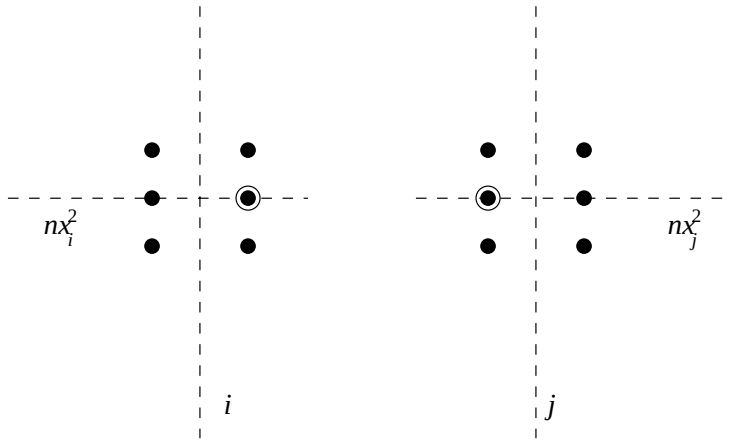


FIG. 5. *Lower bound proof: the case when $x_i = x_j$. The marked points cannot be unoriented maxima.*

maxima, together with a simple divide-and-conquer algorithm.

4. Expected number of unoriented maxima. In this section, we analyze the expected number of unoriented maxima when elements of the set S are independently drawn from a common distribution. Since n points on the perimeter of a convex set are all unoriented maxima, it is only natural to exclude such pathological cases. This is done by assuming that the distribution of the prototype data point is absolutely continuous; i.e., it has a density f. This has the added benefit that with probability one, no two points have the same coordinates. We also assume that f has compact support. Without loss of generality, we can then assume that f vanishes off $[0,1]^2$. We will show that under a mild condition on f, which is satisfied for most distributions that appear in probabilistic models, the expected number of unoriented maxima is $O(\sqrt{n})$. In section 4.4, we describe a divide-and-conquer algorithm that runs in linear expected time for this class of distributions.

The notion of unoriented maximum generalizes that of a maximal vector, for which algorithms can be found in [BS78, BKST78, De80, De85, GBT84, BCL90, Go94, KS85, KLP75]. The expected time was considered in all but the last two of these papers. For additional analysis, see [Dw90, Bu89]. All linear expected time algorithms described in these papers have conditions on the distribution that are more restrictive than the ones used in this paper.

4.1. Preliminaries. We define a cone $C_\theta(x,\eta)$ for $x \in \mathbb{R}^2$, $\theta \in [0,2\pi)$, and $\eta \in [0,2\pi)$ as the collection of all points $y \in \mathbb{R}^2$ with polar coordinate representation $y = x + re^{i\phi}$ for some $r > 0$ and $\phi \in (\theta - \eta/2, \theta + \eta/2)$. Thus, x is the top of the cone, and θ is the direction of the bisector, while η is the opening angle. Given a set of vectors $\mathcal{X}_n = \{X_1, \ldots, X_n\}$ in $\mathbb{R}^2$, we say that X_j is an UNORIENTED MAXIMUM if there exists a θ such that $C_\theta(X_j, \pi/2) \cap \mathcal{X}_n = \emptyset$. Thus, every maximal vector and every point on the convex hull of $\mathcal{X}_n$ is an unoriented maximum of $\mathcal{X}_n$.

It is helpful to cut the problem into manageable subproblems. To do so, we introduce the notion of a restricted unoriented maximum or RUM. Fix a direction $\zeta \in [0,2\pi)$. Call X_j a RUM of $\mathcal{X}_n$ if there exists a direction θ such that

$$C_\theta(X_j, \pi/2) \cap \mathcal{X}_n = \emptyset$$

and

$$C_\theta(X_j, \pi/2) \supseteq C_\zeta(X_j, \pi/3) \ .$$

Call this collection of directional unoriented maxima S_ζ. Obviously, if S is the collection of all unoriented maxima, we have

$$S = \cup_{\zeta \in [0,2\pi)} S_\zeta = \cup_{j=0}^{11} S_{j\pi/6} \ .$$

This property allows us to focus on RUMs. In what follows, we fix $\zeta = \pi/2$ and abbreviate the restricted unoriented maxima with respect to this ζ to RUMs. The set of all RUMs among $X_1, \ldots, X_n$ is denoted by $\mathcal{S}_n$. We list three structural properties of $\mathcal{S}_n$.

1. The Lipschitz property. If $X_i \in \mathcal{S}_n$, $X_j \in \mathcal{S}_n$, then the line segment joining X_i and X_j has an angle with the x-axis within $\pi/3$ of 0 or π. Suppose that the segment forms an angle of ξ degrees, with $\pi/2 \geq \xi > \pi/3$. Then either

$$X_j \in C_\theta(X_i, \pi/2) \supseteq C_\zeta(X_i, \pi/3)$$

for some θ, or vice versa,

$$X_i \in C_\theta(X_j, \pi/2) \supseteq C_\zeta(X_j, \pi/3) \ .$$

In the former case, X_i is not a RUM, and in the latter case, X_j is not a RUM. If we sort all the RUMs from left to right and join them by straight line segments, we obtain a piecewise linear curve that is Lipschitz of constant not more than $\pi/3$. (A function f is Lipschitz of constant C if $|f(x) - f(y)| \leq C|x - y|$.)

2. The monotonicity property.

$$\text{RUM}(X_1, \ldots, X_{n+1}) \subseteq \text{RUM}(X_1, \ldots, X_n) \cup \{X_{n+1}\} \ .$$

3. The transitive property.

$$\text{RUM}(X_1, \ldots, X_{n+m}) = \text{RUM}(\text{RUM}(X_1, \ldots, X_n), \text{RUM}(X_{n+1}, \ldots, X_{n+m})) \ .$$

We will need the following elementary lemma.

LEMMA 7. *If N is a binomial (n, p) random variable, then $\mathbf{P}\{N > enp\} \leq e^{-np}$.*

Proof. By Chernoff's bounding method [Ch52], for $t > 0$ and $\lambda > 0$,

$$\begin{aligned}
\mathbf{P}\{N > t\} &\leq \mathbf{E}\left\{e^{\lambda N - t}\right\} \\
&\leq \left(e^\lambda p + 1 - p\right)^n e^{-\lambda t} \\
&\leq \exp\left((e^\lambda - 1)np - \lambda t\right) \\
&= \exp\left(t - np - t\log\left(\frac{t}{np}\right)\right) \quad (\text{take } \ \lambda = \log(t/(np)) \ )
\end{aligned}$$

so that

$$\mathbf{P}\left\{N > enp\right\} \leq e^{-np} \ . \qquad \square$$

Theorem 4 deals with distributions having a bounded density: for such distributions, there is limited dependence between the components of the random vector X. In a later section, we will obtain analogous results for unbounded densities. In the

bounds presented in this paper, the dependence upon f is measured through $\|f\|_\infty$ or $\int f^\alpha$.

THEOREM 4. *Let X be a random vector on $[0,1]^2$ whose density is bounded by $\|f\|_\infty$. For an i.i.d. sample $X_1, \ldots, X_n$ drawn from X, let $\mathcal{S}_n$ be the collection of* RUM*s. Then*

$$\lim_{n\to\infty} \mathbf{P}\{|\mathcal{S}_n| > C\sqrt{n}\} = 0,$$

where $C = e\sqrt{2(1+2/\sqrt{3})\|f\|_\infty \log 4}$. *Also,*

$$\limsup_{n\to\infty} \frac{\mathbf{E}\{|\mathcal{S}_n|\}}{C\sqrt{n}} \le 1 \ .$$

Proof. As described in the caption of Figure 6, the unit square is covered by a circumscribed rhombus with angles 120, 60, 120, and 60 degrees. From top to bottom, it measures $2a = 1+\sqrt{3}$, and from left to right $2b = 1+1/\sqrt{3}$. The area of the rhombus is $1+2\sqrt{3}$. Partition the rhombus into $m \times m$ equal rhombi as shown in the figure. This is achieved by taking m slabs A_i and m slabs B_j, and defining rhombi by the intersections $A_i \cap B_j$. There are m^2 small rhombi that can be addressed by index pairs (i,j), $1 \le i,j \le m$. A chain of cells is an ordered collection of such pairs, beginning with $(1,1)$ and ending with (m,m), satisfying the successor rule: (i,j) must be followed by either $(i,j+1)$ or $(i+1,j)$. See the lightly shaded collection in Figure 6. Thus, the chain contains precisely $2m-1$ cells, and by a simple counting argument, it is easy to see that there are exactly

$$\binom{2m-2}{m-1}$$

possible chains. Let us mark each cell that contains at least one RUM (dark in Figure 6). We claim that the marked cells are contained in a chain. This, of course, follows from the Lipschitz curve property we established above and our choice of angles when defining the partition. We let $N(\mathcal{C})$ denote the number of data points in the chain $\mathcal{C}$. Thus,

$$|\mathcal{S}_n| \le \max_{\text{all chains } \mathcal{C}} N(\mathcal{C}).$$

By the inclusion–exclusion inequality, we have

$$\begin{aligned}
\mathbf{P}\{|\mathcal{S}_n| > t\} &\le \mathbf{P}\left\{\max_{\text{all chains } \mathcal{C}} N(\mathcal{C}) > t\right\} \\
&\le \sum_{\text{all chains } \mathcal{C}} \mathbf{P}\{N(\mathcal{C}) > t\} \\
&\le \binom{2m-2}{m-1} \sup_{\text{all chains } \mathcal{C}} \mathbf{P}\{N(\mathcal{C}) > t\} \ .
\end{aligned}$$

Next, observe that the probability of a cell is given by

$$\int_{A_i \cap B_j} f(x,y)\,dx\,dy \le \|f\|_\infty \int_{A_i \cap B_j \cap [0,1]^2} dx\,dy \le \frac{\|f\|_\infty (1+2/\sqrt{3})}{m^2} \ .$$

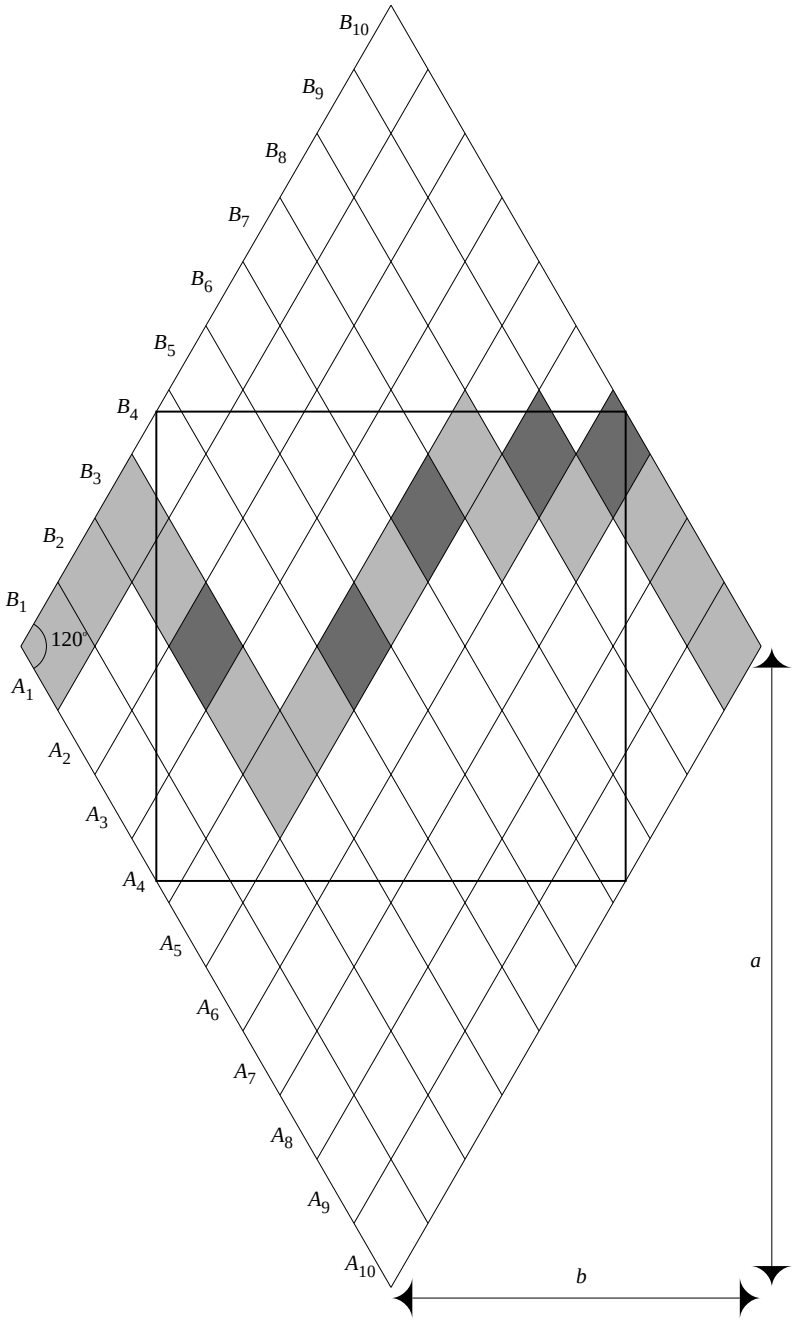


FIG. 6. *The unit square* $[0,1]^2$ *is shown in dark lines. Consider the rhombus of angle* 120 *degrees that circumscribes the square. Partition the rhombus into a grid of* $m \times m$ *similar small rhombi. A chain (in lightest shading) is any collection of small rhombi where the first and last rhombus are the leftmost and rightmost rhombi, respectively, and intermediate rhombi share one side. The rhombi in a chain must have increasing* x*-coordinate values of their centers. The dark shaded rhombi are those that contain at least one restricted unoriented maximum. Observe that these cells always belong to a chain.*

Thus, for a chain of $2m-1$ cells $\mathcal{C}$, $N(\mathcal{C})$ is binomial with parameters n and

$$\int_{\mathcal{C}} f \le \frac{\|f\|_\infty (1+2/\sqrt{3})(2m-1)}{m^2} \le \frac{2\|f\|_\infty (1+2/\sqrt{3})}{m} \stackrel{\text{def}}{=} q \ .$$

Hence, $\mathbf{P}\{N(\mathcal{C}) > t\} \le \mathbf{P}\{\text{Binomial}(n,q) > t\}$. Therefore, by Lemma 7, if $m \to \infty$ as $n \to \infty$,

$$\begin{aligned}
\mathbf{P}\left\{|\mathcal{S}_n| > enq\right\} &\le \binom{2m-2}{m-1} \sup_{\text{all chains } \mathcal{C}} \mathbf{P}\left\{N(\mathcal{C}) > enq\right\} \\
&\sim \frac{2^{2m-2}}{\sqrt{\pi m}} \sup_{\text{all chains } \mathcal{C}} \mathbf{P}\left\{N(\mathcal{C}) > t\right\} \\
&\le \frac{2^{2m-2}}{\sqrt{\pi m}} \mathbf{P}\left\{\text{Binomial}(n,q) > enq\right\} \\
&\le \frac{2^{2m-2} e^{-nq}}{\sqrt{\pi m}} \ .
\end{aligned}$$

Define $q' = mq$. First, take

$$m = \left\lceil \sqrt{\frac{nq'}{\log 4}} \right\rceil \ .$$

Then

$$\mathbf{P}\left\{|\mathcal{S}_n| > enq\right\} \le (1+o(1)) \frac{\exp(m \log 4 - nq'/m)}{4\sqrt{\pi m}} \le \frac{4+o(1)}{\sqrt{\pi m}} \to 0 \ .$$

This proves the first part of Theorem 4. For the second part, choose $\epsilon > 0$ very small and set

$$m = \left\lfloor (1-\epsilon) \sqrt{\frac{nq'}{\log 4}} \right\rfloor \ .$$

We verify quickly that

$$\mathbf{P}\left\{|\mathcal{S}_n| > enq\right\} \le e^{-c\sqrt{n}}$$

for some constant $c > 0$ depending upon ϵ. Then,

$$\mathbf{E}|\mathcal{S}_n| \le enq + n\mathbf{P}\{|\mathcal{S}_n| > enq\} = enq + o(1) \sim e\sqrt{nq' \log 4}/(1-\epsilon) \ .$$

Theorem 4 now follows since ϵ was arbitrary. □

4.2. Lower bounds. The number of unoriented maxima is larger than the number of maximal vectors, i.e., the number of data points X_i for which one of $C_{\pi/4}(X_i, \pi/2)$, $C_{3\pi/4}(X_i, \pi/2)$, $C_{5\pi/4}(X_i, \pi/2)$, and $C_{7\pi/4}(X_i, \pi/2)$ has a nonempty intersection with $X_1, \ldots, X_n$. We denote the set of maximal vectors for $X_1, \ldots, X_n$ by $\mathcal{M}_n$. Thus, $|\mathcal{S}_n| \ge |\mathcal{M}_n|$. This can be used to show that the bound of Theorem 4 cannot be improved for many simple distributions. To clarify this, just take the uniform distribution on the trapezoid T formed by intersecting $[0,1]^2$ with $\{(x,y) : y <$

$x < y + c\}$, with $0 < c \leq 1$. The area of the trapezoid is $1/2(1-(1-c)^2) = c - c^2/2$. Hence,

$$f(x,y) = \frac{1}{c - c^2/2} I_T(x,y) \ .$$

Thus, $\|f\|_\infty = 1/(c - c^2/2)$. We take an integer m large enough such that $1/m < c$. Then partition the unit square into a rectangular grid of m by m with sides equal to $1/m$. Mark the m grid cells that straddle the diagonal of the square. Let $E_1, \ldots, E_m$ be the indicators of the events that the marked grid cells contain at least one data point, with E_1 referring to the cell with the largest y-values, and so on down. A simple geometric argument shows that

$$|\mathcal{M}_n| \geq \sum_{i=1}^{m} E_i \ .$$

Hence, if the marked grid cells intersected with our trapezoid T yield the triangles $S_1, \ldots, S_m$,

$$\begin{aligned}
\mathbf{E}|\mathcal{M}_n| &\geq m\mathbf{E}E_1 \\
&= m\left(1 - (1 - \|f\|_\infty \text{ area}(S_1))^n\right) \\
&= m\left(1 - (1 - \|f\|_\infty/(2m^2))^n\right) \\
&\geq m\left(1 - \exp(-\|f\|_\infty n/(2m^2))\right) \\
&\geq m/2 \\
&\geq \sqrt{\|f\|_\infty n/4\log 4} - 1
\end{aligned}$$

if we choose $m = \lfloor \sqrt{\|f\|_\infty n/\log 4} \rfloor$. Recall that n has to be large enough to insure that $1/m < c$. Thus, we have

$$\mathbf{E}|\mathcal{M}_n| \geq \sqrt{\|f\|_\infty n/4\log 4} - 1 \ .$$

The upper bound in Theorem 4 cannot be improved upon in terms of $\|f\|_\infty$ and n unless the class of distributions is further restricted.

4.3. Random vectors with very dependent coordinates. If f is unbounded, Theorem 4 becomes useless. It is possible, however, that $\int f^\alpha < \infty$ for some $\alpha > 1$. This fact can be used to obtain a different collection of upper bounds.

THEOREM 5. *Let X be a random vector on $[0,1]^2$ whose density satisfies $\int f^\alpha < \infty$ for some $\alpha > 1$. For an i.i.d. sample $X_1, \ldots, X_n$ drawn from X, let $\mathcal{S}_n$ be the collection of* RUMs. *Then*

$$\lim_{n\to\infty} \mathbf{P}\{|\mathcal{S}_n| > Cn^{\alpha/(2\alpha-1)}\} = 0,$$

where

$$C = e\left(\left(\int f^\alpha\right)^{1/\alpha} \left(2(1+2/\sqrt{3})\right)^{1-1/\alpha}\right)^{\alpha/(2\alpha-1)} (\log 4)^{(2\alpha-1)/(\alpha-1)} \ .$$

Also,

$$\limsup_{n\to\infty} \frac{\mathbf{E}\{|\mathcal{S}_n|\}}{Cn^{\alpha/(2\alpha-1)}} \leq 1 \ .$$

Proof. We follow the proof of Theorem 4. Note that $N(\mathcal{C})$ is binomial with parameters n and p, with p given by

$$\begin{aligned}\int_{\mathcal{C}} f(x,y)\,dx\,dy &\le \left(\int f^\alpha\right)^{1/\alpha} \left(\int_{\mathcal{C}\cap[0,1]^2} dx\,dy\right)^{1-1/\alpha} \\ &\le \left(\int f^\alpha\right)^{1/\alpha} \left(\frac{2(1+2/\sqrt{3})}{m}\right)^{1-1/\alpha} \\ &\stackrel{\text{def}}{=} q \stackrel{\text{def}}{=} q'/m^{1-1/\alpha}\ .\end{aligned}$$

Here we used Hölder's inequality and an inequality from the proof of Theorem 4. Therefore, $N(\mathcal{C})$ is binomial with parameters n and p where $p \le q$, and $\mathbf{P}\{N(\mathcal{C}) > t\} \le \mathbf{P}\{\text{ Binomial}(n,q) > t\}$. As in the proof of Theorem 4, when $m \to \infty$ as $n \to \infty$,

$$\mathbf{P}\left\{|\mathcal{S}_n| > enq\right\} \le \frac{(1+o(1))2^{2m-2}e^{-nq}}{\sqrt{\pi m}}\ .$$

With

$$m = \left\lceil \left(\frac{nq'}{\log 4}\right)^{\alpha/(2\alpha-1)} \right\rceil\ ,$$

we obtain

$$\begin{aligned}\mathbf{P}\left\{|\mathcal{S}_n| > enq\right\} &\le (1+o(1))\frac{\exp(m\log 4 - nq'/m^{1-1/\alpha})}{4\sqrt{\pi m}} \\ &\le \frac{4+o(1)}{\sqrt{\pi m}} \\ &\to 0\ .\end{aligned}$$

This proves the first part of Theorem 5. The second part follows from the first part by using arguments analogous to those of Theorem 4. □

Remark 1. We note that the condition $\int f^\alpha < \infty$ imposes a condition on the peakedness of the density f. For bounded densities, we clearly have $\int f^\alpha < \infty$. Theorem 4 is obtained as a limit of Theorem 5 when we let $\alpha \to \infty$. □

Remark 2. If ψ is a positive convex strictly increasing function, then for the chain $\mathcal{C}$ in the proof, we have by Jensen's inequality,

$$\int_{\mathcal{C}} f \le \int_{\mathcal{C}\cap[0,1]^2} dx\,dy \times \psi^{\text{inv}}\left(A/\int_{\mathcal{C}\cap[0,1]^2} dx\,dy\right)\ ,$$

where $A = \int \psi(f)$. Using this instead of Hölder's inequality, with $\psi(u) = u\log^a(1+u)$ for $a > 0$, we see that

$$\mathbf{E}|\mathcal{S}_n| = O\left(\frac{n}{\log^a n}\right)$$

whenever $\int f\log^a(1+f) < \infty$. Observe also that this condition is satisfied whenever $\int f^b < \infty$ for some $b > 1$. □

Remark 3. Theorems 4 and 5 remain valid with different constants for cones $C_\theta(x,\eta)$, $\eta \in (0,\pi]$. □

4.4. Divide-and-conquer algorithms for unoriented maxima. At least five strategically different algorithms can be used for finding the outer layer $\mathcal{M}_n$ of $X_1, \ldots, X_n$ in the plane. Let $L_n = |\mathcal{M}_n|$ denote the number of points on the outer layer.

1. **The naive algorithm.** For each X_i, determine in linear time whether a point is a maximal vector. The time taken by this algorithm is $\Theta(n^2)$, while the space is $\Theta(n)$.
2. **One sort and one elimination pass.** Sort the data points according to their y-coordinates, and eliminate unwanted points in a second stage by passing through the sorted array and keeping partial extrema in the x-direction. This may be implemented in $O(n \log n)$ worst-case time.
3. **Divide-and-conquer** [BS78]. Start with n singleton outer layers, marry (merge) all outer layers pairwise, and repeat these pairwise marriages until one outer layer is left. Noting that outer layers of sizes k and m can be married in $O(k + m)$ time, and that about $\log_2 n$ rounds of merging are needed, it is easy to see that the time taken by this algorithm is $O(n \log n)$. However, since many points are thrown away at early stages, there is reasonable hope of obtaining linear expected time $\mathbf{E}T$. The following is known: $\mathbf{E}T = O(n)$ when the components of X_1 are independent [BS78, De83]. In the general case, $\mathbf{E}T = O(n)$ if and only if $\sum_n \mathbf{E}L_n/n^2 < \infty$ by a general theorem on the expected time analysis of divide-and-conquer algorithms [De83]. An important class of problems is that in which f is bounded, in which case we see that $\mathbf{E}L_n = O(\sqrt{n})$ and thus $\mathbf{E}T = O(n)$ [De85].
4. **Bucketing methods.** Partition $[0, 1]$ into a grid of size about $\sqrt{n} \times \sqrt{n}$, assign all points to grid locations, and mark in all columns (rows) the topmost (leftmost) and bottommost (rightmost) occupied grid cells, together with their inner neighbors. Finally, use the naive algorithm (1) to obtain the outer layer among the points in all the marked cells [De86]. This too yields linear expected time for bounded densities, but it uses a different computational model because truncation is assumed to be available at unit time cost. [Ma84] use another grid in which in each cell, the outer layer is found, and the overall outer layer is found in a second step.
5. **Output-sensitive algorithms based on lazy sorting.** In [KS85], algorithms are presented that take worst-case time bounded by $O(n \log L_n)$. The expected time therefore is bounded by a constant times $\mathbf{E}n \log L_n \leq n \log \mathbf{E}L_n$.

In this section, using the results of the previous sections, we offer a linear expected time divide-and-conquer algorithm for finding the set $\mathcal{S}_n$ of all RUMs that runs under conditions weaker than any condition mentioned above for linear expected time for outer layers. A similarly adapted divide-and-conquer algorithm for outer layers would yield linear expected time under the same general conditions.

PROCEDURE RESTRICTED UNORIENTED MAXIMA

INPUT: A set of n planar points $X_1, \ldots, X_n$.

OUTPUT: The set $\mathcal{S}_n$ of all RUMs of $X_1, \ldots, X_n$.

METHOD:

1. Put all data points X_i in singleton sets S_i.
2. Put all sets S_i in a queue Q.

3. WHILE $|Q| > 1$ DO
   - Dequeue sets S and T from Q.
   - Compute $V = \text{RUM}(S \cup T)$.
   - Enqueue Q with V.

End of Procedure

THEOREM 6. *If the divide-and-conquer algorithm is used on data that are i.i.d. and have a density of compact support such that* $\int f \log^a(1+f) < \infty$ *for some* $a > 1$, *and if the merging of two sets of* RUMs *is supported in linear time, then the overall expected time is* $O(n)$. *The running time is* $O(n \log n)$ *in the worst case.*

Proof. The expected time analysis of general divide-and-conquer algorithms given in [De83] shows that linear expected time is obtained if the data constitute an i.i.d. sequence, the MERGE step takes linear time in the size $|S| + |T|$, and

$$\sum_{n=1}^{\infty} \frac{\mathbf{E}|\mathcal{S}_n|}{n^2} < \infty \ .$$

By Remark 2,

$$\mathbf{E}|\mathcal{S}_n| = O\left(\frac{n}{\log^a n}\right)$$

when f has compact support and $\int f \log^a(1+f) < \infty$. Theorem 6 follows when $a > 1$. □

Remark 4. The condition mentioned in the proof above was rediscovered later in the context of randomized incremental algorithms by Clarkson and Shor [CS88, CS89]. For a slightly different approach with conditions deduced from recursions, see [BS78].

Remark 5. One can push things further and get linear expected time if f has compact support and if for some $a > 1$, $\int f \log(1+f) \log^a \log(1+f) < \infty$. □

Remark 6*: On merging sets of* RUMS. Performing the MERGE step in linear time requires keeping track of the sets of RUMs according to increasing x-coordinates. First, we merge the sorted sets S and T into a set W, sorted by x-coordinate. We then perform two more linear passes. In the first pass, we construct the convex hull in clockwise fashion from left to right as we visit points of W (in fact, this only gives the upper part of the convex hull; the lower part is not needed). This is done by Graham's incremental algorithm [Gr72]. As X_i is processed, we note the angle between the convex hull edge leading to X_i, and the y-axis, and call it θ_i. Repeat a similar pass in counterclockwise manner from right to left, storing the angles in ξ_i. A simple geometric argument shows that $X_i \in \text{RUM}(W)$ if and only if $\theta_i + \xi_i \le \pi/2$. The entire procedure takes linear time. □

Remark 7*: Lazy merging of rums.* If we find $\text{RUM}(W)$ in time $O(|W| \log |W|)$, results from [De83] guarantee overall linear expected time if

$$\sum_{n=1}^{\infty} \frac{\mathbf{E}|\mathcal{S}_n| \log |\mathcal{S}_n|}{n^2} < \infty \ .$$

Since $\log |\mathcal{S}_n| \le \log n$, it suffices to verify that

$$\sum_{n=1}^{\infty} \frac{\mathbf{E}|\mathcal{S}_n| \log n}{n^2} < \infty \ .$$

By Theorem 5, this is satisfied if for some $\alpha > 1$, $\int f^\alpha < \infty$. By Remark 2, it also suffices that $\int f \log^a(1+f) < \infty$ for some $a > 2$. □

5. Concluding remarks. We introduced unoriented Θ-maximal points and described an optimal $O(n \log n)$ algorithm for identifying them when $\Theta \geq \pi/2$. The case $\Theta < \pi/2$ is handled in the Appendix. We also showed that if the points are random and have a common density (satisfying mild regularity conditions), then we can compute the unoriented $\pi/2$-maxima in $O(n)$ expected time.

6. Appendix. For values of $\Theta < \pi/2$, the geometric properties of Lemmas 2 and 5 become useless. However, we are able to modify them slightly as shown below to obtain efficient algorithms for this case.

LEMMA 8. *If point p is a candidate for the edge $e \in E(S)$, then p lies in the part of the circle which has e as a chord, and p makes an angle Θ with e which has a nonempty intersection with the interior of $CH(S)$.*

LEMMA 9. *If point p is an unoriented Θ-maximum with respect to edge e_i, then the angle between the rays L_p and R_p must contain either R_{p,e_i} or one of $\pi/\Theta - 2$ directions which are separated from R_{p,e_i} by integer multiples of Θ.*

A point $p \in S - V(S)$ may be a candidate for at most $2\pi/\Theta$ edges of $CH(S)$. Therefore, Corollary 1 implies that the circles defined in Lemma 8 cannot have more than $14(2\pi/\Theta)h$ $(\in O(n/\Theta))$ intersections, which changes the running time of Procedure Candidates to $O((n/\Theta)\log n)$. In addition, the procedure of section 2.3 for computing unoriented Θ-maxima among candidates needs to be executed $(\pi/\Theta) - 1$ times for each convex hull edge. As a result, we can compute the set S_Θ, for $\Theta < \pi/2$, in $O((n/\Theta)\log n)$ running time. The algorithm is clearly optimal for fixed values of Θ. However, no matching lower bound is known when Θ is part of the input.

REFERENCES

[AES85] D. AVIS, H. ELGINDY, AND R. SEIDEL, *Simple on-line algorithms for convex polygons*, in Computational Geometry, G.T. Toussaint, ed., North-Holland, Amsterdam, 1985, pp. 23–42.

[AH81] D. AVIS AND J. HORTON, *Remarks on the sphere of influence graph*, Ann. New York Acad. Sci., 1981, pp. 323–327.

[BCL90] J. L. BENTLEY, K. L. CLARKSON, AND D. B. LEVINE, *Fast linear expected-time algorithms for computing maxima and convex hulls*, in Proc. 1st Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1990, pp. 179–187.

[BKST78] J. L. BENTLEY, H. T. KUNG, M. SCHKOLNICK, AND C. D. THOMPSON, *On the average number of maxima in a set of vectors*, J. Assoc. Comput. Mach., 25 (1978), pp. 536–543.

[BO79] J. L. BENTLEY AND T. A. OTTMANN, *Algorithms for reporting and counting geometric intersections*, IEEE Trans. Comput., C-28 (1979), pp. 643–647.

[BS78] J. L. BENTLEY AND M. I. SHAMOS, *Divide and conquer for linear expected time*, Inform. Process. Lett., 7 (1978), pp. 87–91.

[Bu89] C. BUCHTA, *On the average number of maxima in a set of vectors*, Inform. Process. Lett., 33 (1989), pp. 63–65.

[Ch52] H. CHERNOFF, *A measure of asymptotic efficiency of tests of a hypothesis based on the sum of observations*, Ann. Math. Stat., 23 (1952), pp. 493–507.

[Ch85] B. CHAZELLE, *On the convex layers of a convex set*, IEEE Trans. Inform. Theory, IT-31 (1985), pp. 509–517.

[CS88] K. L. CLARKSON AND P. W. SHOR, *Algorithms for diametrical pairs and convex hulls that are optimal, randomized, and incremental*, in Proc. 4th Symposium on Computational Geometry, ACM, New York, 1988, pp. 12–17.

[CS89] K. L. CLARKSON AND P. W. SHOR, *Applications of random sampling in computational geometry* II, Disc. Comput. Geom., 4 (1989), pp. 387–422.

[CSY87] R. COLE, M. SHARIR, AND C. K. YAP, *On K-hulls and related problems*, SIAM J. Comput., 16 (1987), pp. 61–77.

[De80] L. DEVROYE, *A note on finding convex hulls via maximal vectors*, Inform. Process. Lett., 11 (1980), pp. 53–56.

[De83] L. Devroye, *Moment inequalities for random variables in computational geometry*, Computing, 30 (1983), pp. 111–119.

[De85] L. Devroye, *On the expected time required to construct the outer layer*, Inform. Process. Lett., 20 (1985), pp. 255–257.

[De86] L. Devroye, *Lecture Notes on Bucket Algorithms*, Birkhäuser-Verlag, Boston, 1986.

[Dw90] R. A. Dwyer, *Kinder, gentler average-case analysis for convex hulls and maximal vectors*, SIGACT News, 21 (1990), pp. 64–71.

[EKS83] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel, *On the shape of a set of points in the plane*, IEEE Trans. Inform. Theory, IT-29 (1983), pp. 551–559.

[GBT84] H. N. Gabow, J. L. Bentley, and R. E. Tarjan, *Scaling and related techniques for geometry problems*, in Proc. 16th Annual ACM Symposium on the Theory of Computing, 1984, pp. 135–143.

[Go94] M. J. Golin, *A provably fast linear-expected-time maxima-finding algorithm finite planar set*, Algorithmica, 11 (1994), pp. 501–524.

[Gr72] R. Graham, *An efficient algorithm for determining the convex hull of a finite planar set*, Inform. Process. Lett., 1 (1972), pp. 132–133.

[KLP75] H. T. Kung, F. Luccio, and F. P. Preparata, *On finding the maxima of a set of vectors*, J. Assoc. Comput. Mach., 22 (1975), pp. 469–476.

[KS85] D. G. Kirkpatrick and R. Seidel, *Output-sensitive algorithms for finding maximal vectors*, in Proc. 2nd Annual Symposium on Computational Geometry, ACM, New York, 1985, pp. 89–96.

[Ma84] M. Machii and Y. Igarashi, *A Hashing Method of Finding the Maxima of a Set of Vectors*, Technical Report CS-84-2, Department of Computer Science, Gunma University, Gunma, Japan, 1984.

[PS85] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[SH76] M. I. Shamos and D. Hoey, *Geometric intersection problems*, in Proc. 17th Annual IEEE Symposium on Foundations of Computer Science, 1976, pp. 208–215.

[Ya89] A. C. C. Yao, *Lower bounds for algebraic computation trees with integer inputs*, in Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 308–313.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 297–310

A SPECTRAL ALGORITHM FOR SERIATION AND THE CONSECUTIVE ONES PROBLEM*

JONATHAN E. ATKINS†, ERIK G. BOMAN‡, AND BRUCE HENDRICKSON§

Abstract. In applications ranging from DNA sequencing through archeological dating to sparse matrix reordering, a recurrent problem is the sequencing of elements in such a way that highly correlated pairs of elements are near each other. That is, given a correlation function f reflecting the desire for each pair of elements to be near each other, find all permutations π with the property that if $\pi(i) < \pi(j) < \pi(k)$ then $f(i,j) \geq f(i,k)$ and $f(j,k) \geq f(i,k)$. This *seriation problem* is a generalization of the well-studied consecutive ones problem. We present a spectral algorithm for this problem that has a number of interesting features. Whereas most previous applications of spectral techniques provide only bounds or heuristics, our result is an algorithm that correctly solves a nontrivial combinatorial problem. In addition, spectral methods are being successfully applied as heuristics to a variety of sequencing problems, and our result helps explain and justify these applications.

Key words. seriation, consecutive ones property, eigenvector, Fiedler vector, analysis of algorithms

AMS subject classifications. 05C15, 15A18, 15A48, 68E15

PII. S0097539795285771

1. Introduction.

Many applied computational problems involve ordering a set so that closely coupled elements are placed near each other. This is the underlying problem in such diverse applications as genomic sequencing, sparse matrix envelope reduction, and graph linear arrangement as well as less familiar settings such as archeological dating. In this paper we present a *spectral algorithm* for this class of problems. Unlike traditional combinatorial methods, our approach uses an eigenvector of a matrix to order the elements. Our main result is that this approach correctly solves an important ordering problem we call the *seriation problem* which includes the well-known consecutive ones problem (C1P) [5] as a special case.

More formally, we are given a set of n elements to sequence; that is, we wish to bijectively map the elements to the integers $1, \ldots, n$. We also have a symmetric, real valued *correlation function* (sometimes called a *similarity function*) that reflects the desire for elements i and j to be near each other in the sequence. We now wish to find all ways to sequence the elements so that the correlations are *consistent*; that is, if π is our permutation of elements and $\pi(i) < \pi(j) < \pi(k)$ then $f(i,j) \geq f(i,k)$ and $f(j,k) \geq f(i,k)$. Although there may be an exponential number of such orderings, they can all be described in a compact data structure known as a PQ-tree [5], which we review in the next section. Not all correlation functions allow for a consistent sequencing. If a consistent ordering is possible we will say the problem is *well posed*.

* Received by the editors May 8, 1995; accepted for publication (in revised form) January 8, 1997; published electronically June 15, 1998. This work was supported by the Mathematical, Information, and Computational Sciences Division of the U.S. DOE, Office of Energy Research, and was performed at Sandia National Laboratories, operated for the U.S. DOE under contract DE-AL04-94AL8500.
http://www.siam.org/journals/sicomp/28-1/28577.html

† Infinity Financial Technology, Mountain View, CA 94043 (atkins@infinity.com).

‡ Scientific Computing & Computational Mathematics, Gates Bldg. 2B, Stanford University, Stanford, CA 94305-9025 (boman@sccm.stanford.edu).

§ Applied & Numerical Mathematics Department, Sandia National Laboratories, Albuquerque, NM 87185-1110 (bah@cs.sandia.gov).

Determining an ordering from a correlation function is what we will call the *seriation problem*, reflecting its origins in archeology [29, 33].

C1P is a closely related ordering problem. A $(0,1)$-matrix C has the *consecutive ones property* if there exists a permutation matrix Π such that for each column in ΠC, all the ones form a consecutive sequence. If a matrix has the consecutive ones property, then the C1P is to find all such permutations. As shown by Kendall [19] and reviewed in section 6, C1P is a special case of the seriation problem.

Our algorithm orders elements using their value in an eigenvector of a *Laplacian matrix* which we formally define in section 2. Eigenvectors related to graphs have been studied since the 1950s (see, for example, the survey books by Cvetković et al. [8, 7]). Most of the early work involved eigenvectors of adjacency matrices. Laplacian eigenvectors were first studied by Fiedler [10, 11] and independently by Donath and Hoffman [9]. More recently, there have been a number of attempts to apply spectral graph theory to problems in combinatorial optimization. For example, spectral algorithms have been developed for graph coloring [3], graph partitioning [9, 28], and envelope reduction [4], and more examples can be found in the survey papers of Mohar [23, 24]. However, in most previous applications, these techniques have been used to provide bounds, heuristics, or in a few cases, approximation algorithms [2, 6, 14] for NP-hard problems. There are only a small number of previous results in which eigenvector techniques have been used to exactly solve combinatorial problems including finding the number of connected components of a graph [10], coloring k-partite graphs [3], and finding stable sets (independent sets) in perfect graphs [16]. This paper describes another such application.

Spectral methods are closely related to the more general method of *semidefinite programming*, which has been applied successfully to many combinatorial problems (e.g., MAX-CUT and MAX-2SAT [14] and graph coloring [18]). See Alizadeh [1] for a survey of semidefinite programming with applications to combinatorial optimization.

Our result is important for several reasons. First, it provides new insight into the well-studied C1P. Second, some important practical problems like envelope reduction for matrices and genomic reconstruction can be thought of as variations on seriation. For example, if biological experiments were error-free, the genomic reconstruction problem would be precisely C1P. Unfortunately, real experimental data always contain errors, and attempts to generalize the consecutive ones concept to data with errors seems to invariably lead to NP-complete problems [31, 15]. A spectral heuristic based upon our approach has recently been applied to such problems and found to be highly successful in practice [15]. Our result helps explain this empirical success by revealing that in the error-free case the technique will correctly solve the problem. This places the spectral method on a stronger theoretical footing as a cross between a heuristic and an exact algorithm. Similar comments apply to envelope reduction. Matrices with dense envelopes are closely related to matrices with the consecutive ones property. Recent work has shown spectral techniques to be better in practice than any existing combinatorial approaches at reducing envelopes [4]. Our result sheds some light on this success.

Another way to interpret our result is that we provide an algorithm for C1P that generalizes to become an attractive heuristic in the presence of errors. Designed as decision algorithms for the consecutive ones property, existing combinatorial approaches for C1P break down if there are errors and fail to provide useful approximate orderings. However, our goal here is not to analyze the approach as an approximation algorithm, but rather to prove that it correctly solves error-free problem instances.

This paper is organized in the following way. In the next section we introduce the mathematical notation and the results from matrix theory that we will need later. We also describe a spectral heuristic for ordering problems which motivates the remainder of the paper. The theorem that underpins our algorithm is proved in section 3, the proof of which requires the use of a classical theorem from matrix analysis. Several additional results in section 4 lead us to an algorithm and its analysis in section 5. We review the connection to C1P in section 6.

2. Mathematical background.

2.1. Notation and definitions. Matrix concepts are useful because the correlation function defined above can be considered as a real, symmetric matrix. A permutation of the elements corresponds to a symmetric permutation of this matrix, a permutation of the matrix elements formed by permuting the rows and the columns in the same fashion. The question of whether or not the ordering problem is well posed can also be asked as a property of this matrix. Specifically, suppose the matrix has been permuted to reflect a consistent solution to the ordering problem. The off-diagonal matrix entries must now be nonincreasing as we move away from the diagonal. More formally, we will say a matrix A is an *R-matrix*[1] if and only if A is symmetric and

$$\begin{aligned} a_{i,j} &\leq a_{i,k} \qquad \text{for } j < k < i, \\ a_{i,j} &\geq a_{i,k} \qquad \text{for } i < j < k. \end{aligned}$$

The diagonal entries of an R-matrix are unspecified. If A can be symmetrically permuted to become an R-matrix, then we say that A is *pre-R*. Note that pre-R matrices correspond precisely to well-posed ordering problems. Also, the R-matrix property is preserved if we add a constant to all off-diagonal entries, so we can assume without loss of generality that all off-diagonal values are nonnegative.

When π is a permutation of the natural numbers $\{1, \ldots, n\}$ and x is a column vector, i.e. $x = [x_1, \ldots, x_n]^T$, we will denote by x^π the permutation of x by π, i.e., $x_i^\pi = x_{\pi(i)}$. Similarly, A^π is the symmetric permutation of A by π, i.e., $a_{i,j}^\pi = a_{\pi(i),\pi(j)}$. We denote by e the vector whose entries are all 1, by e_i the vector consisting of zeros except for a 1 in position i, and by I the identity matrix. A symmetric matrix A is *reducible* if there exists a permutation π such that

$$A^\pi = \begin{bmatrix} B & 0 \\ 0 & C \end{bmatrix},$$

where B and C are nonempty square matrices. If no such permutation exists then A is *irreducible*. If B and C are themselves irreducible, then we refer to them as the *irreducible blocks* of A.

We say that λ is an *eigenvalue* of A if $Ax = \lambda x$ for some vector $x \neq 0$. A corresponding vector x is an *eigenvector*. An $n \times n$ real, symmetric matrix has n eigenvectors that can be constructed to be pairwise orthogonal, and its eigenvalues are all real. We will assume that the eigenvalues are sorted by increasing value, and refer to them as λ_i, $i = 1, \ldots, n$. The *(algebraic) multiplicity* of an eigenvalue λ is defined as the number of times λ occurs as a root in the characteristic polynomial

[1] This class of matrices is named after W. S. Robinson who first defined this property in his work on seriation methods in archaeology [29].

$p(z) = \det(A - zI)$. A value that occurs only once is called *simple*; the eigenvector of a simple eigenvalue is unique (up to normalization). We write $A \geq 0$ and say A is nonnegative if all its elements $a_{i,j}$ are nonnegative. A real vector x is *monotone* if $x_i \leq x_{i+1}$ for all $1 \leq i < n$ or if $x_i \geq x_{i+1}$ for all $1 \leq i < n$.

We define the *Laplacian* of a symmetric matrix A to be $L_A = D_A - A$, where D_A is a diagonal matrix with $d_{i,i} = \sum_{j=1}^{n} a_{i,j}$. The minimum eigenvalue with an eigenvector orthogonal to e (the vector of all ones) is called the *Fiedler value*, and a corresponding eigenvector is called a *Fiedler vector*.[2] Alternatively, the Fiedler value is given by

$$\min_{x^T e=0, x^T x=1} x^T L_A x,$$

and a Fiedler vector is any vector x that achieves this minimum while satisfying these constraints. When $A \geq 0$ and irreducible, it is not hard to show that the Fiedler value is the smallest nonzero eigenvalue and a Fiedler vector is any corresponding eigenvector. We will be notationally cavalier and refer to the Fiedler value and vector of A when we really mean those of L_A.

2.2. PQ-trees. A *PQ-tree* is a data structure introduced by Booth and Lueker to efficiently encode a set of related permutations [5]. A PQ-tree over a set $U = \{u_1, u_2, \ldots, u_n\}$ is a rooted, ordered tree whose leaves are elements of U and whose internal nodes are distinguished as either P-nodes or Q-nodes. A PQ-tree is *proper* when the following three conditions hold.

1. Every element $u_i \in U$ appears precisely once as a leaf.
2. Every P-node has at least two children.
3. Every Q-node has at least three children.

Two PQ-trees are said to be equivalent if one can be transformed into the other by applying a sequence of the following two equivalence transformations.

1. Arbitrarily permute the children of a P-node.
2. Reverse the children of a Q-node.

Conveniently, the equivalence class represented by a PQ-tree corresponds precisely to the set of permutations consistent with an instance of a seriation problem. In section 5 we describe an algorithm which uses Laplacian eigenvectors to construct a PQ-tree for an instance of the seriation problem.

2.3. Motivation for spectral methods. With the above definitions we can describe a simple heuristic for the seriation problem that will motivate the remainder of the paper. This heuristic is at the heart of the more complex algorithms we will devise, and underlies many previous applications of spectral algorithms [17]. We begin by constructing a simple penalty function g whose value will be small when closely correlated elements are close to each other. We define $g(\pi) = \sum_{(i,j)} f(i,j)(\pi_i - \pi_j)^2$. Unfortunately, minimizing g is NP-hard due to the discrete nature of the permutation [13]. Instead we approximate it by a function h of continuous variables x_i that we can minimize and that maintains much of the structure of g. We define $h(x) = \sum_{(i,j)} f(i,j)(x_i - x_j)^2$. Note that h does not have a unique minimizer, since its value does not change if we add a constant to each x component. To avoid this

[2] This is in recognition of the work of Miroslav Fiedler [10, 11].

ambiguity, we need to add a constraint like $\sum_i x_i = 0$. We still have a trivial solution when all the x_i's are zero, so we need a second constraint like $\sum_i x_i^2 = 1$. The resulting minimization problem is now well defined.

$$\text{Minimize } h(x) = \sum_{(i,j)} f(i,j)(x_i - x_j)^2 \tag{1}$$
$$\text{subject to } \sum_i x_i = 0, \text{ and } \sum_i x_i^2 = 1.$$

The solution to this continuous problem can be used as a heuristic for sequencing. Merely construct the solution vector x, sort the elements x_i, and sequence based upon their sorted order. One reason this heuristic is attractive is that the minimization problem has an elegant solution. We can rewrite $h(x)$ as $x^T L_F x$ where $F = \{f_{ij}\}$ is the correlation matrix. The constraints require that x be a unit vector orthogonal to e, and since L_A is symmetric, all other eigenvectors satisfy the constraints. Consequently, a solution to the constrained minimization problem is just a Fiedler vector.

Even if the problem is not well posed, sorting the entries of the Fiedler vector generates an ordering that tries to keep highly correlated elements near each other. As mentioned above, this technique is being used for a variety of sequencing problems [4, 15, 17]. The algorithm we describe in the remainder of the paper is based upon this idea. However, when we encounter ties in entries of the Fiedler vector, we need to recurse on the subproblem encompassing the tied values. In this way, we are able to find all permutations which make a pre-R-matrix into an R-matrix.

3. The key theorem. Our main result is that a modification of the simple heuristic presented in section 2.3 is actually an algorithm for well-posed instances of the seriation problem. Completely proving this will require us to deal with the special cases of multiple Fiedler vectors and ties within the Fiedler vector. The cornerstone of our analysis is a classical result in matrix theory due to Perron and Frobenius [27]. The particular formulation below can be found on p. 46 of [30].

THEOREM 3.1 (Perron–Frobenius). *Let M be a real, nonnegative matrix. If we define $\rho(M) = \max_i |\lambda_i(M)|$, then*

1. *$\rho(M)$ is an eigenvalue of M, and*
2. *there is a vector $x \geq 0$ such that $Mx = \rho(M)x$.*

We are now ready to state and prove our main theorem.

THEOREM 3.2. *If A is an R-matrix then it has a monotone Fiedler vector.*

Proof. Our proof uses the Perron–Frobenius Theorem 3.1. The nonnegative vector in that theorem will consist of differences between neighboring entries in the Fiedler vector of the Laplacian of A.

First define the matrix $S \in \mathbb{R}^{(n-1)\times n}$ as

$$S = \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -1 & 1 \end{bmatrix}.$$

Note that for any vector x, $Sx = (x_2 - x_1, \ldots, x_n - x_{n-1})^T$. Define $T \in \mathbb{R}^{n\times(n-1)}$ by

$$T = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & & 0 \\ 1 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 1 & 1 & \cdots & 1 \end{bmatrix}.$$

It is easy to verify that $ST = I_{n-1}$, and that $TS = I_n - ee_1^T$. We define $M_A = SL_AT = \{m_{i,j}\}$ and let $L_A = \{l_{i,j}\}$. We now show that Sx is an eigenvector of M_A if and only if x is an eigenvector of L_A and $x \neq \alpha e$.

$$\begin{aligned} L_Ax = \lambda x, \;\; x \neq \alpha e \iff \\ SL_Ax = \lambda Sx, \; x \neq \alpha e \iff \\ SL_A(I - ee_1^T)x = \lambda Sx, \; x \neq \alpha e \iff \\ SL_ATSx = \lambda Sx, \; x \neq \alpha e \iff \\ M_Ay = \lambda y, \text{ where } y = Sx \neq 0. \end{aligned}$$

The transformation from the second to the third lines follows from $L_Ae = 0$. Equivalence holds between all the above equations, so λ is an eigenvalue for both L_A and M_A for eigenvectors of L_A other than e. Hence the eigenvalues of M_A are the same as the eigenvalues of L_A with the zero eigenvalue removed, and the eigenvectors of M_A are differences between neighboring entries of the corresponding eigenvectors of L_A.

It is easily seen that $(SL_A)_{i,k} = -l_{i,k} + l_{i+1,k}$ for all i, k, so

$$m_{i,j} = \sum_{k=1}^{n} (SL_A)_{i,k} T_{k,j} = \sum_{k=j+1}^{n} (-l_{i,k} + l_{i+1,k}) = \sum_{k=j+1}^{n} (a_{i,k} - a_{i+1,k}).$$

Since, by assumption, A is an R-matrix, $a_{i,k} \leq a_{i+1,k}$ for $i < k+1$, and therefore $m_{i,j} \leq 0$ for $i < j$. For $i > j$ we can use the fact that $\sum_{k=1}^{n} l_{i,k} = 0$ to obtain

$$m_{i,j} = \sum_{k=j+1}^{n} (-l_{i,k} + l_{i+1,k}) = \sum_{k=1}^{j} (l_{i,k} - l_{i+1,k}) = \sum_{k=1}^{j} (-a_{i,k} + a_{i+1,k}).$$

Again, from the R-matrix property we conclude that $m_{ij} \leq 0$ for $i > j$. Consequently, all the off-diagonal elements in M_A are nonpositive.

Now let β be a value greater than $\max_i\{\lambda_i, m_{ii}\}$, where λ_i are the eigenvalues of M_A. Then $\tilde{M}_A = \beta I - M_A$ is nonnegative with eigenvalues $\tilde{\lambda}_i = \beta - \lambda_i$. Also, $\tilde{M}_A$ and M_A share the same set of eigenvectors. By Theorem 3.1, there exists a nonnegative eigenvector y of $\tilde{M}_A$ corresponding to the largest eigenvalue of $\tilde{M}_A$. But y is also an eigenvector of M_A corresponding to M_A's smallest eigenvalue. And this is just Sx, where x is a Fiedler vector of L_A. Since $y = Sx$ is nonnegative, the corresponding Fiedler vector of L_A is nondecreasing and the theorem follows. (Note that since the sign of an eigenvector is unspecified, the Fiedler vector could also be nonincreasing.) □

THEOREM 3.3. *Let A be a pre-R-matrix with a simple Fiedler value and a Fiedler vector with no repeated values. Let π_1 (respectively, π_2) be the permutation induced by sorting the values in the Fiedler vector in increasing (decreasing) order. Then A^{π_1} and A^{π_2} are R-matrices, and no other permutations of A produce R-matrices.*

Proof. First note that since the Fiedler value is simple, the Fiedler vector is unique up to a multiplicative constant. Next observe that if x is the Fiedler vector of A, then x^π is the Fiedler vector of A^π. So applying a permutation to A merely changes the order of the entries in the Fiedler vector. Now let π_* be a permutation such that A^{π_*} is an R-matrix. By Theorem 3.2, x^{π_*} is monotone since x is the only Fiedler vector. Since x has no repeated values, π_* must be either π_1 or π_2. □

Theorem 3.3 provides the essence of our algorithm for the seriation problem, but it is too restrictive, as the Fiedler value must be simple and contain no repeated values. We will show how to remove these limitations in the next section.

4. Removing the restrictions. Several observations about the seriation problem will simplify our analysis. First note that if we add a constant to all the correlation values the set of solutions is unchanged. Consequently, we can assume without loss of generality that the smallest value of the correlation function is zero. Note that subtracting the smallest value from all correlation values does not change whether or not the matrix is pre-R. In our algebraic formulation this translates into the following.

LEMMA 4.1. *Let A be a symmetric matrix and let $\bar{A} = A - \alpha ee^T$ for some real α. A vector x is a Fiedler vector of A if and only if x is a Fiedler vector of $\bar{A}$. So without loss of generality we can assume that the smallest off-diagonal entry of A is zero.*

Proof. By the definition of a Laplacian it follows that $L_{\bar{A}} = L_A + \alpha ee^T - \alpha nI$, where n is the dimension of A. Then $L_{\bar{A}}e = 0$, but for any other eigenvector x of L_A, $L_{\bar{A}}x = L_A x + 0 - \alpha nx$. That is, the eigenvalues are simply shifted down by αn while the eigenvectors are preserved. □

This will justify the first step of our algorithm, which subtracts the value of the smallest correlation from every correlation. Accordingly, we now make the assumption that our pre-R-matrix has smallest off-diagonal entry of zero. Next observe that if A is reducible, then the seriation problem can be decoupled. The irreducible blocks of the matrix correspond to connected components in the graph of the nonzero values of the correlation function. We can solve the subproblems induced by each of these connected components and link the pieces together in an arbitrary order. More formally, we have the following lemma.

LEMMA 4.2. *Let A_i, $i = 1, \ldots, k$, be the irreducible blocks of a pre-R-matrix A, and let π_i be a permutation of block A_i such that the submatrix $A_i^{\pi_i}$ is an R-matrix. Then any permutation formed by concatenating the π_i's will make A become an R-matrix. In terms of a PQ-tree, the π_i permutations are children of a single P-node.*

Proof. By Lemma 4.1, we can assume all entries in the irreducible blocks are nonnegative. Consequently, the correlation between elements within a block will always be at least as strong as the correlation between elements in different blocks. Also, by the definition of irreducibility, each element within a block must have some positive correlation with another element in that block. Hence, any ordering that makes A_i an R-matrix must not interleave elements between different irreducible blocks. As long as the blocks themselves are ordered to be R-matrices, any ordering of blocks will make A an R-matrix since correlations across blocks are all identical. □

With these preliminaries, we will now assume that the smallest off-diagonal value is zero and that the matrix is irreducible. As the following three lemmas and theorem show, this is sufficient to ensure that the Fiedler vector is unique up to a multiplicative constant.

LEMMA 4.3. *Let A be an $n \times n$ R-matrix with a monotone Fiedler vector x. If $\mathcal{J} = [r, s]$ is a maximal interval such that $x_r = x_s$, then for any $k \notin \mathcal{J}$, $a_{r,k} =$*

$a_{r+1,k} = \cdots = a_{s,k}$.

Proof. We can without loss of generality assume x is nondecreasing since $-x$ is also a Fiedler vector. We will show that $a_{r,k} = a_{s,k}$ for all $k \notin \mathcal{J}$, and since A is an R-matrix then all elements between $a_{r,k}$ and $a_{s,k}$ must also be equal. Consider rows r and s in the equation $L_A x = \lambda x$:

$$\sum_{k=1}^{n} (l_{s,k} - l_{r,k}) x_k = \lambda (x_s - x_r) = 0.$$

Since L_A is a Laplacian, we know that $\sum_{k=1}^{n} l_{i,k} = 0$ for all i. We get

$$\begin{aligned} 0 &= \sum_{k=1}^{n} (l_{sk} - l_{rk})(x_r - x_k) \\ &= \sum_{k=1}^{r-1} \underbrace{(l_{s,k} - l_{r,k})}_{\geq 0} \underbrace{(x_r - x_k)}_{>0} + \sum_{k=s+1}^{n} \underbrace{(l_{s,k} - l_{r,k})}_{\leq 0} \underbrace{(x_r - x_k)}_{<0} \end{aligned}$$

where we have used the fact that x is nondecreasing. Because all terms in the sum are nonnegative, all terms must be exactly zero. By assumption, $x_k \neq x_r$ for $k \notin \mathcal{J}$ and consequently $l_{r,k} = l_{s,k}$ for $k \notin \mathcal{J}$ and the result follows. ∎

The following lemma is essentially a converse of this. Its proof requires detailed algebra, but it is not fundamental to what follows. Consequently, the proof is relegated to the end of this section.

LEMMA 4.4. *Let A be an irreducible $n \times n$ R-matrix with $a_{n,1} = 0$. If $\mathcal{J} = [r, s] \neq [1, n]$ is an interval such that $a_{r,k} = a_{s,k}$ for all $k \notin \mathcal{J}$, then $x_r = x_{r+1} = \cdots = x_s$ for any Fiedler vector x.*

LEMMA 4.5. *Let A be an irreducible R-matrix with $a_{n,1} = 0$, and x, a monotone Fiedler vector of A. If $\mathcal{J} = [r, s]$ is an interval such that $x_r = x_{r+1} = \cdots = x_s$, then for any Fiedler vector y, $y_r = y_{r+1} = \cdots = y_s$.*

Proof. First apply Lemma 4.3 to conclude that for any $k \notin \mathcal{J}$, $a_{r,k} = a_{r+1,k} = \cdots = a_{s,k}$. Since $x^T e = 0$, it follows that $\mathcal{J} \neq [1, n]$. Now use this in conjunction with Lemma 4.4 to obtain the result. ∎

THEOREM 4.6. *If A is an irreducible R-matrix with $a_{n,1} = 0$, then the Fiedler value λ_2 is a simple eigenvalue.*

Proof. We will assume that λ_2 is a repeated eigenvalue and produce a contradiction. Let x and y be two linearly independent Fiedler vectors with x nondecreasing. Define $z(\theta) = \cos(\theta)x + \sin(\theta)y$, with $0 \leq \theta \leq \pi$. Let θ^* be the smallest value of θ that makes $z_k = z_{k+1}$ for some k where $x_k \neq x_{k+1}$. Such a θ^* must exist since x and y are linearly independent.

By Lemma 4.5 the indices of any repeated values in x are indices of repeated values in y and $z(\theta)$. Coupled with the monotonicity of x, this implies that $z(\theta^*)$ is monotone. By Lemma 4.5 the indices of any repeated values in $z(\theta^*)$ must be repeated in x, which gives the desired contradiction. ∎

All that remains is to handle the situation where the Fiedler vector has repeated values. As the following theorem shows, repeated values decouple the problem into pieces that can be solved recursively.

THEOREM 4.7. *Let A be a pre-R-matrix with a simple Fiedler value and Fiedler vector x. Suppose there is some repeated value β in x and define $\mathcal{I}$, $\mathcal{J}$, and $\mathcal{K}$ to be the indices for which*

1. $x_i < \beta$ *for all* $i \in \mathcal{I}$,
2. $x_i = \beta$ *for all* $i \in \mathcal{J}$,
3. $x_i > \beta$ *for all* $i \in \mathcal{K}$.

Then π *is an R-matrix ordering for* A *if and only if* π *or its reversal can be expressed as* (π_i, π_j, π_k), *where* π_j *is an R-matrix ordering for the submatrix* $A(\mathcal{J}, \mathcal{J})$ *of* A *induced by* $\mathcal{J}$, *and* π_i *and* π_k *are the restrictions of some R-matrix ordering for* A *to* $\mathcal{I}$ *and* $\mathcal{K}$, *respectively.*

Proof. From Theorem 3.2 we know that for any R-matrix ordering A^π, x^π is monotone, so elements in $\mathcal{I}$ must appear before (after) elements from $\mathcal{J}$ and elements from $\mathcal{K}$ must appear after (before) elements from $\mathcal{J}$. By Lemma 4.3, we have $a_{ik} = a_{jk}$ for all $i, j \in \mathcal{J}$ and $k \notin \mathcal{J}$. Hence the orderings of elements inside $\mathcal{J}$ must be indifferent to the ordering outside of $\mathcal{J}$ and vice versa. Consequently, the R-matrix ordering of elements in $\mathcal{J}$ depends only on $A(\mathcal{J}, \mathcal{J})$. ∎

Algorithmically, this theorem means that we can break ties in the Fiedler vector by recursing on the submatrix $A(\mathcal{J}, \mathcal{J})$ where $\mathcal{J}$ corresponds to the set of repeated values. The distinct values in the Fiedler vector of A constrain R-matrix orderings, but repeated values need to be handled recursively. In the language of PQ-trees, the distinct values are combined via a Q-node, and the components (subtrees) of the Q-node must then be expanded recursively.

Proof of Lemma 4.4. First we recall that the Fiedler value is the value obtained by

$$\min_{x^T e = 0, x^T x = 1} x^T L_A x = \min_{x^T e = 0, x^T x = 1} \sum_{i>j} a_{i,j}(x_i - x_j)^2, \tag{2}$$

and a Fiedler vector is a vector that achieves this minimum. We note that if we replace A by a matrix that is at least as large on an elementwise comparison, then $x^T L_A x$ cannot decrease for any vector x.

We consider $A(\mathcal{J}, \mathcal{J})$, the diagonal block of A indexed by $\mathcal{J}$. By the definition of an R-matrix, all values in $A(\mathcal{J}, \mathcal{J})$ must be at least as large as $a_{r,s}$. However, $a_{r,s}$ must be greater than zero. Otherwise, by the R-matrix property, $a_{i,j} = 0$ for all $i \geq r$ and $j < s$ and for all $j \geq r$ and $i < s$. But then, by the statement of the theorem, $a_{i,j} = 0$ for all $i \geq s$ and $j < s$ and all $j \geq r$ and $j < s$, which would make the matrix reducible.

The remainder of the proof will proceed in two stages. First we will force all the off-diagonal values in $A(\mathcal{J}, \mathcal{J})$ to be $a_{r,s}$ and show the result for this modified matrix. We will then extend the result to our original matrix.

Stage 1. We define the matrix B to be identical to A outside of $B(\mathcal{J}, \mathcal{J})$, but all off-diagonal values of B within $B(\mathcal{J}, \mathcal{J})$ are set to $\alpha = a_{r,s}$. It follows from the hypotheses that B is an R-matrix. We define $\delta = l_{i,i}$ for $i \in \mathcal{J}$ and note that, by the R-matrix property, $\delta \leq (n-1)\alpha$.

We now define $\tilde{L_B} = L_B - (\delta + \alpha)I$ and consider the eigenvalue equation $\tilde{L_B} x = \tilde{\lambda}_2 x$. This matrix has the same eigenvectors as L_B with eigenvalues shifted by $\delta + \alpha$. Since $\tilde{l}_{ii} = \delta - (\delta + \alpha) = \alpha$ for $i \in \mathcal{J}$, all rows of $\tilde{L_B}$ in $\mathcal{J}$ are identical. Consequently, either all elements of x in $\mathcal{J}$ are equal, or $\tilde{\lambda}_2 = 0$ (which is equivalent to $\lambda_2 = \delta + \alpha$). We will show that irreducibility and $a_{n1} = 0$ implies $\lambda_2 \neq \delta + \alpha$, which will complete the proof of Stage 1.

We assume $\lambda_2 = \delta + \alpha$ and look for a contradiction. We introduce a new matrix

$\hat{B}$ as follows:

$$\hat{b}_{i,j} = \begin{cases} b_{i,j} & \text{if } i < r \text{ and } j < r, \\ b_{i,j} & \text{if } i > s \text{ and } j > s, \\ \alpha & \text{otherwise.} \end{cases}$$

Since B is an R-matrix, $\hat{B}$ is at least as large as B elementwise, so $\lambda_2(\hat{B}) \geq \lambda_2(B)$. We define the vector $\hat{y}$ by

$$\hat{y}_i = \begin{cases} -(n-s), & \text{if } i < r, \\ 0, & \text{if } r \leq i \leq s, \\ r-1, & \text{if } i > s, \end{cases}$$

and $\hat{x}$ to be the unit vector in the direction of $\hat{y}$. We note that $\hat{x}^T e = 0$, and that $\hat{x}^T L_{\hat{B}} \hat{x} = n\alpha$. We have the following chain of inequalities:

$$\lambda_2 = \min_{x^T e = 0, x^T x = 1} x^T L_B x \leq \hat{x}^T L_B \hat{x} < \hat{x}^T L_{\hat{B}} \hat{x} = n\alpha. \tag{3}$$

The last inequality is strict since $\hat{b}_{n,1} = \alpha$ while $b_{n,1} = 0$ and $(\hat{x}_n - \hat{x}_1)^2 > 0$.

If $\lambda_2 = \delta + \alpha$, then we can combine an inequality due to Fiedler [10],

$$\lambda_2 \leq \frac{n}{n-1} \min_i l_{ii},$$

with the observation that $\min_i l_{i,i} \leq \delta$ to obtain $\lambda_2 \leq \frac{n}{n-1}\delta \leq \delta + \alpha = \lambda_2$. This can only be true if equality holds throughout, implying that $\delta = (n-1)\alpha$ and $\lambda_2 = n\alpha$. But this contradicts (3), so $\lambda_2 \neq \delta + \alpha$ and the proof of Stage 1 is complete.

Stage 2. We will now show that A and B have the same Fiedler vectors. Since A is elementwise at least as large as B, for any vector z, $z^T L_A z \geq z^T L_B z$. From Stage 1 we know that any Fiedler vector of B satisfies $x_r = x_{r+1} = \cdots = x_s$. In this vector, $(x_i - x_j) = 0$ for $i, j \in \mathcal{J}$, so the contribution to the sum in (2) from $B(\mathcal{J}, \mathcal{J})$ is zero. But this contribution will also be zero when applied to $A(\mathcal{J}, \mathcal{J})$. Since A and B are identical outside of $A(\mathcal{J}, \mathcal{J})$ and $B(\mathcal{J}, \mathcal{J})$, we now have that a Fiedler vector of B gives an upper bound for the Fiedler value of A; that is, $\lambda_2(A) \leq \lambda_2(B)$. It follows that the Fiedler vectors of B are also Fiedler vectors of A and vice versa. □

5. A spectral algorithm for the seriation problem. We can now bring all the preceding results together to produce an algorithm for well-posed instances of the seriation problem. Specifically, given a well-posed correlation function we will generate all consistent orderings. Given a pre-R-matrix, our algorithm constructs a PQ-tree for the set of permutations that produce an R-matrix.

Our Spectral-Sort algorithm is presented in Fig. 1. It begins by translating all the correlations so that the smallest is 0. It then separates the irreducible blocks (if there are more than one) into the children of a P-node and recurses. If there is only one such block, it sorts the elements into the children of a Q-node based on their values in a Fielder vector. If there are ties in the entries of the Fiedler vector, the algorithm is invoked recursively.

We now prove that the algorithm is correct. Step (1) is justified by Lemma 4.1, and requires time proportional to the number of nonzeros in the matrix. The identification of irreducible blocks in step (2) can be performed with a breadth-first or depth-first search algorithm, also requiring time proportional to the number of nonzeros. Combining the permutations of the resulting blocks with a P-node is correct by Lemma 4.2.

```
     Input:    A, an n × n pre-R-matrix
               U, a set of indices for the rows/columns of A
     Output:   T, a PQ-tree that encodes the set of all permutations π
               such that A^π is an R-matrix
     begin
(1)      α := min_{i≠j} a_{i,j}
(1)      A := A − αee^T
(2)      {A_1, ..., A_k} := the irreducible blocks of A
(2)      {U_1, ..., U_k} := the corresponding index sets
(2)      if k > 1
(2)          for j := 1 : k
(2)              T_j := Spectral-Sort(A_j, U_j)
(2)          end
(2)          T := P-node(T_1, T_2, ..., T_k)
         else
(3)          if (n = 1)
(3)              T := u_1
(3)          else if (n = 2)
(3)              T := P-node(u_1, u_2)
             else
(4)              x := Fiedler vector for L_A
(4)              Sort x
(5)              t := number of distinct values in x
(5)              for j := 1 : t
(5)                  V_j := indices of elements in x with jth value
(5)                  T_j := Spectral-Sort(A(V_j, V_j), V_j)
(5)              end
                 T := Q-node(T_1, ..., T_T)
             end
         end
     end
```

FIG. 1. *Algorithm* Spectral-Sort.

Step (3) handles the boundary conditions of the recursion, while in step (4) the Fiedler vector is computed and sorted. If there are no repeated elements in the Fiedler vector then the Q-node for the permutation is correct by Theorem 3.3. Steps (3) and (4) are the dominant computational steps and we will discuss their run time below. The recursion in step (5) is justified by Theorem 4.7.

Note that this algorithm produces a tree whether A is pre-R or not. To determine whether A is pre-R, simply apply one of the generated permutations. If the result is an R-matrix, then all permutations in the PQ-tree will solve the seriation problem; otherwise the problem is not well posed.

The most expensive steps in algorithm Spectral-Sort are the generation and sorting of the eigenvector. Since the algorithm can invoke itself recursively, these operations can occur on problems of size n, $n-1$, ...,1. So if the time for an eigencalculation on a matrix of size n is $T(n)$, the run time of algorithm Spectral-Sort is $O(n(T(n) + n \log n))$.

A formal analysis of the complexity of the eigenvector calculation can be simpli-

fied by noting that for a pre-R-matrix, all that matters is the dominance relationships between matrix entries. So, without loss of generality, we can assume that all entries are integers less than n^2. With this observation, it is possible to compute the components of the Fiedler vector to a sufficient precision such that the components can be correctly sorted in polynomial time. We now sketch one way this can be done, although we don't recommend this procedure in a real-world implementation.

Let λ denote a specific eigenvalue of L, in our case the Fiedler value. This can be computed in polynomial time as discussed in [25]. Then we can compute the corresponding eigenvector x symbolically by solving

$$(L - zI)x = 0 \bmod p(z),$$

where $p(z)$ is the characteristic polynomial of L. Gaussian elimination over a field is in P [21], so if $p(z)$ is irreducible we obtain a solution x where each component x_i is given by a polynomial in z with bounded integer coefficients. We note that letting z be any eigenvalue will force x to be a true eigenvector. If $p(z)$ is reducible, we try the above. If we fail to solve the equation, we will instead find a factorization of $p(z)$ and proceed by replacing $p(z)$ with the factor containing λ as a root. This yields a polynomial formula for each x_i, and we can identify equal elements by, e.g., the method in [22]. To decide the order of the remaining components, we evaluate the root λ to a sufficient precision and then compute the x_i's numerically and sort. Since λ is algebraic, the x_i's cannot be arbitrarily close [22] and polynomial precision is sufficient.

In practice, eigencalculations are a mainstay of the numerical analysis community. To calculate eigenvectors corresponding to the few highest or lowest eigenvalues (like the Fiedler vector), the method of choice is known as the Lanczos algorithm. This is an iterative algorithm in which the dominant cost in each iteration is a matrix-vector multiplication which requires $O(m)$ time. The algorithm generally converges in many fewer than n iterations, often only $O(\sqrt{n})$ [26]. However, a careful analysis reveals a dependence on the difference between the distinct eigenvalues.

6. C1P. Ordering an R-matrix is closely related to C1P. As mentioned in section 1, a $(0,1)$-matrix C has the *consecutive ones property* if there exists a permutation matrix Π such that for each column in ΠC, all the ones form a consecutive sequence.[3] A matrix that has this property without any rearrangement (i.e., $\Pi = I$) is in Petrie form[4] and is called a P-matrix. Analogous to R-matrices, we say a matrix with the consecutive ones property is *pre-P*. C1P can be restated as: given a pre-P-matrix C, find a permutation matrix Π such that ΠC is a P-matrix.

There is a close relationship between P-matrices and R-matrices. The following results are due to D.G. Kendall and are proved in [19] and [33].

LEMMA 6.1. *If C is a P-matrix, then $A = CC^T$ is an R-matrix.*

LEMMA 6.2. *If C is pre-P and $A = CC^T$ is an R-matrix, then C is a P-matrix.*

THEOREM 6.3. *Let C be a pre-P matrix, let $A = CC^T$, and let Π be a permutation matrix. Then ΠC is a P-matrix if and only if $\Pi A \Pi^T$ is an R-matrix.*

This theorem allows us to use algorithm Spectral-Sort to solve C1P. First construct $A = CC^T$, and then apply our algorithm to A (note that the elements of A are small nonnegative integers). Now apply one of the permutations generated by the

[3] Some authors define this property in terms of rows instead of columns.

[4] Sir William M. F. Petrie was an archeologist who studied mathematical methods for seriation in the 1890s.

algorithm to C. If the result is a P-matrix then all the permutations produce C1P orderings. If not, then C has no C1P orderings.

The run time for this technique is not competitive with the linear time algorithm for this problem due to Booth and Lueker [5]. However, unlike their approach, our Spectral-Sort algorithm does not break down in the presence of errors and can instead serve as a heuristic.

Several other combinatorial problems have been shown to be equivalent to C1P. Among these are recognizing interval graphs [5, 12] and finding dense envelope orderings of matrices [5].

One generalization of P-matrices is to matrices with unimodal columns (a unimodal sequence is a sequence that is nondecreasing until it reaches its maximum, then nonincreasing). These matrices are called unimodal matrices [32]. Kendall [20] showed that the results of Lemmas 6.1 and 6.2 and Theorem 6.3 are also valid for unimodal matrices if the regular matrix product is replaced by the matrix *circle product* defined by

$$(A \circ B)_{ij} = \sum_k \min(a_{ik}, b_{kj}).$$

Note that P-matrices are just a special case of unimodal matrices, and that the circle product is equivalent to the matrix product for $(0, 1)$-matrices. Kendall's result implies that our spectral algorithm will correctly identify and order unimodal matrices.

Acknowledgments. We are indebted to Robert Leland for innumerable discussions about spectral techniques and to Sorin Istrail for his insights into the consecutive ones problem and his constructive feedback on an earlier version of this paper. We are further indebted to David Greenberg for his experimental testing of our approach on simulated genomic data and to Nabil Kahale for showing us how to simplify the proof of Theorem 3.2. We also appreciate the highly constructive feedback provided by an anonymous referee.

REFERENCES

[1] F. Alizadeh, *Interior point methods in semidefinite programming with applications to combinatorial optimization*, SIAM J. Optim., 5 (1995), pp. 13–51.

[2] N. Alon and N. Kahale, *A spectral technique for coloring random 3-colorable graphs*, in Proc. 26th Annual Symposium on Theory of Computing, ACM, New York, 1994, pp. 346–355.

[3] B. Aspvall and J. R. Gilbert, *Graph coloring using eigenvalue decomposition*, SIAM J. Alg. Disc. Meth., 5 (1984), pp. 526–538.

[4] S. T. Barnard, A. Pothen, and H. D. Simon, *A spectral algorithm for envelope reduction of sparse matrices*, in Proc. Supercomputing '93, IEEE, Piscataway, NJ, 1993, pp. 493–502.

[5] K. S. Booth and G. S. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 333–379.

[6] F. R. K. Chung and S.-T. Yau, *A near optimal algorithm for edge seperators (preliminary version)*, in Proc. 26th Annual Symposium on Theory of Computing, ACM, New York, 1994, pp. 1–8.

[7] D. M. Cvetković, M. Doob, I. Gutman, and A. Torgasev, *Recent Results in the Theory of Graph Spectra*, Annals of Discrete Mathematics 36, North-Holland, Amsterdam, 1988.

[8] D. M. Cvetković, M. Doob, and H. Sachs, *Spectra of Graphs: Theory and Application*, Academic Press, New York, 1980.

[9] W. E. Donath and A. J. Hoffman, *Lower bounds for the partitioning of graphs*, IBM J. Res. Develop., 17 (1973), pp. 420–425.

[10] M. Fiedler, *Algebraic connectivity of graphs*, Czech. Math. Journal, 23 (1973), pp. 298–305.

[11] M. Fiedler, *A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory*, Czech. Math. Journal, 25 (1975), pp. 619–633.

[12] D. R. Fulkerson and O. A. Gross, *Incidence matrices and interval graphs*, Pacific J. Math., 3 (1965), pp. 835–855.
[13] A. George and A. Pothen, *An analysis of spectral envelope-reduction via quadratic assignment problems*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 706–732.
[14] M. X. Goemans and D. P. Williamson, *.878-approximation algorithms for MAX CUT and MAX 2SAT*, in Proc. 26th Annual Symposium on Theory of Computing, ACM, New York, 1994, pp. 422–431.
[15] D. S. Greenberg and S. C. Istrail, *Physical mapping by STS hybridization: Algorithmic strategies and the challenge of software evaluation*, J. Comput. Biology, 2 (1995), pp. 219–274.
[16] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.
[17] M. Juvan and B. Mohar, *Optimal linear labelings and eigenvalues of graphs*, Disc. Appl. Math., 36 (1992), pp. 153–168.
[18] D. Karger, R. Motwani, and M. Sudan, *Approximate graph coloring by semidefinite programming*, in Proc. 35th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 2–13.
[19] D. G. Kendall, *Incidence matrices, interval graphs and seriation in archaeology*, Pacific J. Math., 28 (1969), pp. 565–570.
[20] D. G. Kendall, *Abundance matrices and seriation in archaeology*, Zeitschrift für Wahrscheinlichkeitstheorie, 17 (1971), pp. 104–112.
[21] D. C. Kozen, *The Design and Analysis of Algorithms*, Springer-Verlag, New York, 1992.
[22] M. Mignotte, *Identification of algebraic numbers*, J. Algorithms, 3 (1982), pp. 197–204.
[23] B. Mohar, *The Laplacian spectrum of graphs*, in Graph Theory, Combinatorics and Applications, Y. Alavi, G. Chartrand, O. Oellermann, and A. Schwenk, eds., Wiley, New York, 1991, pp. 871–898.
[24] B. Mohar, *Laplace eigenvalues of graphs – a survey*, Disc. Math., 109 (1992), pp. 171–183.
[25] V. Pan, *Algebraic complexity of computing polynomial zeros*, Comput. Math. Appl., 14 (1987), pp. 285–304.
[26] B. Parlett and D. Scott, *The Lanczos algorithm with selective orthogonalization*, Math. Comp., 33 (1979), pp. 217–238.
[27] O. Perron, *Zur Theorie der Matrizen*, Math. Ann., 1907, pp. 248–263.
[28] A. Pothen, H. D. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 430–452.
[29] W. S. Robinson, *A method for chronologically ordering archaeological deposits*, American Antiquity, 16 (1951), pp. 293–301.
[30] R. S. Varga, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.
[31] M. Veldhorst, *Approximation of the consecutive ones matrix augmentation problem*, SIAM J. Comput., 14 (1985), pp. 709–729.
[32] E. M. Wilkinson, *Mathematics in the archaeological and historical sciences*, in Archaeological Seriation and the Travelling Salesman Problem, University Press, Edinburgh, 1971, pp. 276–284.
[33] E. M. Wilkinson, *Techniques of data analysis and seriation theory*, in Technische und Naturwissenschaftliche Beiträge Zur Feldarchäologie, Rheinland-Verlag, Cologne, Germany, 1974, pp. 1–142.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 311–324

NEW COLLAPSE CONSEQUENCES OF NP HAVING SMALL CIRCUITS*

JOHANNES KÖBLER† AND OSAMU WATANABE‡

Abstract. We show that if a self-reducible set has polynomial-size circuits, then it is low for the probabilistic class ZPP(NP). As a consequence we get a deeper collapse of the polynomial-time hierarchy PH to ZPP(NP) under the assumption that NP has polynomial-size circuits. This improves on the well-known result in Karp and Lipton [*Proceedings of the 12th ACM Symposium on Theory of Computing*, ACM Press, New York, 1980, pp. 302–309] stating a collapse of PH to its second level Σ_2^{P} under the same assumption. Furthermore, we derive new collapse consequences under the assumption that complexity classes like UP, FewP, and $\mathrm{C}_{=}\mathrm{P}$ have polynomial-size circuits.

Finally, we investigate the circuit-size complexity of several language classes. In particular, we show that for every fixed polynomial s, there is a set in ZPP(NP) which does not have $O(s(n))$-size circuits.

Key words. polynomial-size circuits, advice classes, lowness, randomized computation

AMS subject classifications. 03D10, 03D15, 68Q10, 68Q15

PII. S0097539795296206

1. Introduction.

The question of whether intractable sets can be efficiently decided by nonuniform models of computation has motivated much work in structural complexity theory. In research from the early 1980s to the present, a variety of results has been obtained showing that this is impossible under plausible assumptions (see, e.g., the survey [18]). A typical model for nonuniform computations are circuit families. In the notation of Karp and Lipton [22], sets decidable by polynomial-size circuits are precisely the sets in P/poly; i.e., they are decidable in polynomial time with the help of a polynomial length bounded advice function [32].

Karp and Lipton (together with Sipser) [22] proved that no NP-complete set has polynomial size circuits (in symbols $\mathrm{NP} \not\subseteq \mathrm{P/poly}$) unless the polynomial-time hierarchy collapses to its second level. The proof given in [22] exploits a certain kind of self-reducibility of the well-known NP-complete problem SAT. More generally, it is shown in [8, 7] that every (Turing) self-reducible set in P/poly is low for the second level Σ_2^{P} of the polynomial time hierarchy. Intuitively speaking, a set is low for a relativizable complexity class if it gives no additional power when used as an oracle for that class.

In this paper, we show that every self-reducible set in P/poly is even low for the probabilistic class ZPP(NP), meaning that $\mathrm{ZPP}(\mathrm{NP}(A)) = \mathrm{ZPP}(\mathrm{NP})$. Since for every oracle A, $\Sigma_2^{\mathrm{P}}(A) = \exists \cdot \mathrm{ZPP}(\mathrm{NP}(A))$, lowness for ZPP(NP) implies lowness for Σ_2^{P}. As a consequence of our lowness result we get a deeper collapse of the polynomial-time hierarchy to ZPP(NP) under the assumption that NP has polynomial-size circuits. At least in some relativized world, the new collapse level is quite close to optimal: there

*Received by the editors December 6, 1995; accepted for publication (in revised form) January 27, 1997; published electronically June 15, 1998. A preliminary version of this work appeared in *Lecture Notes in Comput. Sci.* 944, Springer-Verlag, New York, 1995, pp. 196–207.

http://www.siam.org/journals/sicomp/28-1/29620.html

†Abteilung für Theoretische Informatik, Universität Ulm, Oberer Eselsberg, D-89069 Ulm, Germany (koebler@informatik.uni-ulm.de).

‡Department of Computer Science, Tokyo Institute of Technology, Meguro-ku, Tokyo 152, Japan (watanabe@cs.titech.ac.jp). Part of this work has been done while visiting the University of Ulm. This research was supported in part by the guest scientific program of the University of Ulm.

is an oracle relative to which NP is contained in P/poly, but PH does not collapse to P(NP) [17, 39].

We also derive new collapse consequences from the assumption that complexity classes like UP, FewP, and $C_=P$ have polynomial-size circuits. Furthermore, our lowness result implies new *relativizable* collapses for the case that $\mathrm{Mod}_m P$, PSPACE, or EXP have polynomial-size circuits. As a final application, we derive new circuit-size lower bounds. In particular, it is shown (by relativizing proof techniques) that for every fixed polynomial s, there is a set in ZPP(NP) which does not have $O(s(n))$-size circuits. This improves on the result of Kannan [21] that for every polynomial s, the class $\Sigma_2^P \cap \Pi_2^P$ contains such a set. It further follows that in every relativized world, there exist sets in the class ZPEXP(NP) that do not have polynomial-size circuits. It should be noted that there is a nonrelativizing proof for a stronger result. As a corollary to the result in [4], which is proved by a nonrelativizing technique, it is provable that $\mathrm{MA}_{\mathrm{exp}} \cap \text{co-}\mathrm{MA}_{\mathrm{exp}}$ (a subclass of ZPEXP(NP)) contains non-P/poly sets [12, 36].

Some explanation of how our work builds on prior techniques is in order. The proof of our lowness result heavily uses the universal hashing technique [13, 34] and builds on ideas from [2, 14, 24]. For the design of a *zero error* probabilistic algorithm which, with the help of an NP oracle, simulates a given ZPP(NP(A)) computation (where A is a self-reducible set in P/poly) we further make use of the newly defined concept of half-collisions. More precisely, we show how to compute on input 0^n in expected polynomial time a hash family H that can be used to decide all instances of A of length up to n by a strong NP computation. The way H is used to decide (non)membership to A is by checking whether H leads to a half-collision on certain sets. Very recently, Bshouty, Cleve, Gavaldà, Kannan, and Tamon [11] building on a result from [19] have shown that the class of all circuits is exactly learnable in (randomized) expected polynomial time with equivalence queries and the aid of an NP oracle. This immediately implies that for every set A in P/poly an advice function can be computed in FZPP(NP(A)), i.e., by a probabilistic oracle transducer T in expected polynomial time under an oracle in NP(A). More precisely, since the circuit produced by the probabilistic learning algorithm of [11] depends on the outcome of the coin flips, T computes a *multivalued* advice function; i.e., on input 0^n, T accepts with probability at least 1/2, and on every accepting path, T outputs some circuit that correctly decides all instances of length n w.r.t. A. Using the technique in [11] we are able to show that every self-reducible set A in P/poly even has an advice function in FZPP(NP). Although this provides a different way to deduce the ZPP(NP) lowness of all self-reducible sets in P/poly, we prefer to give a self-contained proof using the "half-collision technique" that does not rely on the mentioned results in [11, 19].

The paper is organized as follows: section 2 introduces notation and defines the self-reducibility that we use. In section 3 we prove the ZPP(NP) lowness of all self-reducible sets in P/poly. In section 4 we state the collapse consequences, and the new circuit-size lower bounds are derived in section 5.

2. Preliminaries and notation. All languages are over the binary alphabet $\Sigma = \{0, 1\}$. As usual, we denote the lexicographic order on Σ^* by $\leq$. The *length* of a string $x \in \Sigma^*$ is denoted by $|x|$. $\Sigma^{\leq n}$ ($\Sigma^{<n}$) is the set of all strings of length at most n (resp., of length smaller than n). For a language A, $A^{=n} = A \cap \Sigma^n$ and $A^{\leq n} = A \cap \Sigma^{\leq n}$. The cardinality of a finite set A is denoted by $|A|$. The *characteristic function* of A is defined as $A(x) = 1$ if $x \in A$, and $A(x) = 0$ otherwise. For a class $\mathcal{C}$ of sets, co-$\mathcal{C}$ denotes the class $\{\Sigma^* - A \mid A \in \mathcal{C}\}$. To encode pairs (or tuples) of

strings we use a standard polynomial-time computable pairing function denoted by $\langle \cdot, \cdot \rangle$ whose inverses are also computable in polynomial time. Where intent is clear we write $f(x_1, \ldots, x_k)$ in place of $f(\langle x_1, \ldots, x_k \rangle)$. $\mathcal{N}$ denotes the set of nonnegative integers. Throughout the paper, the base of log is 2.

The textbooks [9, 10, 25, 31, 33] can be consulted for the standard notation used in the paper and for basic results in complexity theory. For definitions of probabilistic complexity classes like ZPP, see also [15].

An *NP machine* M is a polynomial-time nondeterministic Turing machine. We assume that each computation path of M on a given input x either accepts, rejects, or outputs "?". *M accepts* on input x, if M performs at least one accepting computation, otherwise *M rejects x*. *M strongly accepts (strongly rejects) x* [26] if

- there is at least one accepting (resp., rejecting) computation path and
- there are no rejecting (resp., accepting) computation paths.

If M strongly accepts or strongly rejects x, M is said to perform a *strong computation* on input x. An NP machine that on every input performs a strong computation is called a *strong NP machine*. It is well known that exactly the sets in NP $\cap$ co-NP are accepted by strong NP machines [26].

Next we define the kind of self-reducibility that we use in this paper.

DEFINITION 2.1. *Let $\succ$ be an irreflexive and transitive order relation on Σ^*. A sequence $x_0, x_1, \ldots, x_k$ of strings is called a $\succ$-chain (of length k) from x_0 to x_k if $x_0 \succ x_1 \succ \cdots \succ x_k$. Relation $\succ$ is called* length checkable *if there is a polynomial q such that*

1. *for all $x, y \in \Sigma^*$, $x \succ y$ implies $|y| \leq q(|x|)$,*
2. *the language $\{\langle x, y, k \rangle \mid$ there is a $\succ$-chain of length k from x to $y\}$ is in* NP.

DEFINITION 2.2. *A set A is* self-reducible *if there is a polynomial-time oracle machine M_{self} and a length checkable order relation $\succ$ such that $A = L(M_{self}, A)$ and on any input x, M_{self} queries the oracle only about strings $y \prec x$.*

It is straightforward to check that the polynomially related self-reducible sets introduced by Ko [23] as well as the length-decreasing and word-decreasing self-reducible sets of Balcázar [6] are self-reducible in our sense. Furthermore, it is well-known (see, for example, [9, 6, 29]) that complexity classes like NP, Σ_k^{P}, Π_k^{P}, PP, $\mathrm{C}_{=}\mathrm{P}$, $\mathrm{Mod}_m\mathrm{P}$, PSPACE, and EXP have many-one complete self-reducible sets.

Karp and Lipton [22] introduced the notion of advice functions in order to characterize nonuniform complexity classes. A function $h : \mathcal{N} \to \Sigma^*$ is called a *polynomial-length function* if for some polynomial p and for all $n \geq 0$, $|h(n)| = p(n)$. For a class $\mathcal{C}$ of sets, let $\mathcal{C}/\mathrm{poly}$ be the class of sets A such that there are a set $I \in \mathcal{C}$ and a polynomial-length function h such that for all n and for all $x \in \Sigma^{\leq n}$,

$$x \in A \;\Leftrightarrow\; \langle x, h(n) \rangle \in I.$$

Function h is called an *advice function* for A, whereas I is the corresponding *interpreter set*.

In this paper we will heavily make use of the "hashing technique" which has been very fruitful in complexity theory. Here we review some notations and facts about hash families. We also extend the notion of "collision" by introducing the concept of a "half-collision" which is central to our proof technique.

Sipser [34] used universal hashing, originally invented by Carter and Wegman [13], to decide (probabilistically) whether a finite set X is large or small. A linear hash function h from Σ^m to Σ^k is given by a Boolean (k, m)-matrix (a_{ij}) and maps

any string $x = x_1 \dots x_m$ to a string $y = y_1 \dots y_k$, where y_i is the inner product $a_i \cdot x = \sum_{j=1}^{m} a_{ij} x_j \pmod 2$ of the ith row a_i and x.

Let $x \in \Sigma^m$, $Y \subseteq \Sigma^m$, and let h be a linear hash function from Σ^m to Σ^k. Then we say that x *has a collision on* Y *w.r.t.* h if there exists a string $y \in Y$, different from x, such that $h(x) = h(y)$. More generally, if X is a subset of Σ^m and H is a family[1] $(h_1, \dots, h_l)$ of linear hash functions from Σ^m to Σ^k, then we say that X *has a collision on* Y *w.r.t.* H (*Collision*(X, Y, H) for short) if there is some $x \in X$ that has a collision on Y w.r.t. every h_i in H. That is,

$$\begin{aligned} \textit{Collision}(X,Y,H) \quad \Leftrightarrow \quad & \exists\, x \in X\, \exists\, y_1, \dots, y_l \in Y : x \notin \{y_1, \dots y_l\} \\ & \text{and for all } i = 1, \dots, l : h_i(x) = h_i(y_i). \end{aligned}$$

If X has a collision on itself w.r.t. H, we simply say that X *has a collision w.r.t.* H. Next we extend the notion of "collision" in the following way. For any X and $Y \subseteq \Sigma^m$, and any family $H = (h_1, \dots, h_l)$ of linear hash functions, we say that X *has a half-collision on* Y *w.r.t.* H (*Half-Collision*(X, Y, H) for short) if there is some $x \in X$ that has a collision on Y w.r.t. at least $\lceil l/2 \rceil$ many of the hash functions h_i in H. That is,

$$\begin{aligned} \textit{Half-Collision}(X,Y,H) \quad \Leftrightarrow \quad & \exists\, x \in X\, \exists\, y_1, \dots, y_l \in Y : x \notin \{y_1, \dots y_l\} \\ & \text{and } |\{i \mid 1 \le i \le l, h_i(x) = h_i(y_i)\}| \ge \lceil l/2 \rceil. \end{aligned}$$

An important relationship between collisions and half-collisions is the following one: if X has a collision w.r.t. H on $Y = Y_1 \cup Y_2$, then X must have a half-collision w.r.t. H either on Y_1 or on Y_2.

Note that the predicate *Collision*(X, Y, H) can be decided in NP provided that membership in X and Y can be tested in NP. More precisely, the language $\{\langle v, H\rangle \mid \textit{Collision}(X_v, Y_v, H)\}$ (as well as the set $\{\langle v, H\rangle \mid \textit{Half-Collision}(X_v, Y_v, H)\}$) belongs to NP if the sets X_v and Y_v are succinctly represented in such a way that the languages $\{\langle x, v\rangle \mid x \in X_v\}$ and $\{\langle y, v\rangle \mid y \in Y_v\}$ are in NP.

We denote the set of all families $H = (h_1, \dots, h_l)$ of l linear hash functions from Σ^m to Σ^k by $\mathcal{H}(l, m, k)$. The following theorem is proved by a pigeon-hole argument. It says that every sufficiently large set must have a collision w.r.t. any hash family.

THEOREM 2.3 (see [34]). *For any hash family* $H \in \mathcal{H}(l, m, k)$ *and any set* $X \subseteq \Sigma^m$ *of cardinality* $|X| > l \cdot 2^k$, X *must have a collision w.r.t.* H.

On the other hand, we get from the next theorem (called the coding lemma in [34]) an upper bound on the collision probability for sufficiently small sets.

THEOREM 2.4 (see [34]). *Let* $X \subseteq \Sigma^m$ *be a set of cardinality at most* 2^{k-1}. *If we choose a hash family* H *uniformly at random from* $\mathcal{H}(k, m, k)$, *then the probability that* X *has a collision w.r.t.* H *is at most* $1/2$.

We will also make use of the following extension of Theorem 2.4 which can be proved along the same lines.

THEOREM 2.5. *Let* $X \subseteq \Sigma^m$ *be a set of cardinality at most* 2^{k-s}. *If we choose a hash family* H *uniformly at random from* $\mathcal{H}(l, m, k)$, *then the probability that* X *has a collision w.r.t.* H *is at most* $2^{k-s(l+1)}$.

Gavaldà [14] extended Sipser's coding lemma (Theorem 2.4) to the case of a collection $\mathcal{C}$ of exponentially many sets. The following theorem has a similar flavor.

THEOREM 2.6. *Let* $\mathcal{C}$ *be a collection of at most* 2^n *subsets of* Σ^m, *each of which has cardinality at most* 2^{k-s}. *If we choose a hash family* H *uniformly at random from*

[1] More precisely, sequence.

$\mathcal{H}(l,m,k)$, then the probability that some $X \in \mathcal{C}$ has a collision w.r.t. H is at most $2^{n+k-s(l+1)}$.

Proof. By Theorem 2.5, we have that for every fixed $X \in \mathcal{C}$, the probability that it has a collision w.r.t. a randomly chosen hash family $H \in \mathcal{H}(l,m,k)$ is at most $2^{k-s(l+1)}$. Hence, the probability that there exists such a set $X \in \mathcal{C}$ is at most $2^{n+k-s(l+1)}$. □

In this paper we make use of a corresponding result for the case of half-collisions.

THEOREM 2.7. *Let $X \subseteq \Sigma^m$ and let $\mathcal{C}$ be a collection of at most 2^n subsets of Σ^m, each of which has cardinality at most 2^{k-s-2}. If we choose a hash family H uniformly at random from $\mathcal{H}(l,m,k)$, then the probability that X has a half-collision on some $Y \in \mathcal{C}$ w.r.t. H is at most $|X| \cdot 2^{n-sl/2}$.*

Proof. For every fixed $Y \in \mathcal{C}$ and every fixed $x \in X$, the probability that x has a collision on Y w.r.t. a randomly chosen h is at most 2^{-s-2}. Hence, the probability that x has a collision on Y w.r.t. at least half of the functions in a randomly chosen hash family $H \in \mathcal{H}(l,m,k)$ is at most

$$\sum_{i=\lceil l/2 \rceil}^{l} \binom{l}{i} (2^{-s-2})^i (1-2^{-s-2})^{l-i} \leq 2^{-(s+2)l/2} \sum_{i=\lceil l/2 \rceil}^{l} \binom{l}{i} \leq 2^{l-(s+2)l/2} = 2^{-sl/2}.$$

That is, the probability that x has a half-collision on Y w.r.t. a randomly chosen hash family H is bounded by $2^{-sl/2}$. Hence, the probability that there exists a $Y \in \mathcal{C}$ and an $x \in X$ such that x has a half-collision on Y w.r.t. H is at most $|X| \cdot 2^{n-sl/2}$. □

3. Lowness of self-reducible sets in P/poly. In this section, we show that every self-reducible set A in (NP ∩ co-NP)/poly is low for ZPP(NP). Let $I \in$ NP ∩ co-NP be an interpreter set and h be an advice function for A. We construct a probabilistic algorithm T and an NP oracle O having the following two properties:

(a) The expected running time of T is polynomially bounded.
(b) On every computation path on input 0^n, T with oracle O outputs some information that can be used to determine the membership to A of any x up to length n by some strong NP computation (in the sense of [26]).

Using these properties, we can prove the lowness of A for ZPP(NP) as follows: in order to simulate any NP(A) computation, we first precompute the above-mentioned information for A (up to some length) by T^O, and then by using this information, we can simulate the NP(A) computation by some NP(NP ∩ co-NP) computation. Note that the precomputation (performed by T^O) can be done in ZPP(NP), and since NP(NP ∩ co-NP) = NP, the remaining computation can be done in NP. Hence, NP(A) ⊆ ZPP(NP), which implies further that ZPP(NP(A)) ⊆ ZPP(ZPP(NP)) (= ZPP(NP) [41]).

We will now make the term "information" precise. For this, we need some additional notation. Let the self-reducibility of A be witnessed by a polynomial-time oracle machine M_{self}, a length checkable order relation $\succ$, and a polynomial q. We assume that $|h(q(n))| = p(n)$ for some fixed polynomial $p > 0$. In the following, we fix n and consider instances of length up to $q(n)$ as well as advice strings of length exactly $p(n)$.

- A *sample* is a sequence $\langle x_1, b_1 \rangle \# \cdots \# \langle x_k, b_k \rangle$ of pairs, where the x_i's are instances of length up to $q(n)$ and $b_i = A(x_i)$ for $i = 1, \ldots, k$.
- For any sample $S = \langle x_1, b_1 \rangle \# \cdots \# \langle x_k, b_k \rangle$, let *Consistent*($S$) be the set of all advice strings w that are consistent with S, i.e.,

$$\mathit{Consistent}(S) = \{ w \in \Sigma^{p(n)} \mid \forall i \, (1 \leq i \leq k) : I(x_i, w) = b_i \}.$$

The cardinality of *Consistent*(S) is denoted by $c(S)$.

- For any sample S and any instance x, let *Accept*(x, S) (resp., *Reject*(x, S)) be the set of all consistent advice strings that *accept* x (resp., *reject* x):

$$\mathit{Accept}(x, S) = \{w \in \mathit{Consistent}(S) \mid I(x, w) = 1\}$$

and

$$\mathit{Reject}(x, S) = \{w \in \mathit{Consistent}(S) \mid I(x, w) = 0\}.$$

- Let *Correct*(x, S) be the set $\{w \in \mathit{Consistent}(S) \mid I(x, w) = A(x)\}$ of consistent advice strings that decide x correctly, and let *Incorrect*(x, S) be the complementary set $\{w \in \mathit{Consistent}(S) \mid I(x, w) \neq A(x)\}$.

Note that the sets *Accept*(x, S) and *Reject*(x, S) (as well as *Correct*(x, S) and *Incorrect*(x, S)) form a partition of the set *Consistent*(S), and that

$$\begin{array}{lll} x \in A & \Rightarrow & \mathit{Correct}(x, S) = \mathit{Accept}(x, S) \text{ and } \mathit{Incorrect}(x, S) = \mathit{Reject}(x, S), \\ x \notin A & \Rightarrow & \mathit{Correct}(x, S) = \mathit{Reject}(x, S) \text{ and } \mathit{Incorrect}(x, S) = \mathit{Accept}(x, S). \end{array}$$

The above condition (b) can now be precisely stated as follows.

(b) On every computation path on input 0^n, T^O outputs a pair $\langle S, H\rangle$ consisting of a sample S and a linear hash family H such that for all x up to length n, *Consistent*(S) has a half-collision w.r.t. H on *Correct*(x, S) but not on *Incorrect*(x, S).

Once we have a pair $\langle S, H\rangle$ satisfying condition (b), we can determine whether an instance x of length up to n is in A by simply checking whether *Consistent*(x, S) has a half-collision w.r.t. H on *Accept*(x, S) or on *Reject*(x, S). Since condition (b) guarantees that the half-collision can always be found, this checking can be done by a strong NP computation. Let us now prove our main lemma.

LEMMA 3.1. *For any self-reducible set A in* $(\mathrm{NP} \cap \text{co-NP})/\mathrm{poly}$*, there exist a probabilistic transducer T and an oracle O in NP satisfying the above two conditions.*

Proof. We use the notation introduced so far. Recall that $q(n)$ is a length bound on the queries occurring in the self-reduction tree produced by M_{self} on any instance of length n and that $p(n)$ is the advice length for the set of all instances of length up to $q(n)$. Let l be the polynomial defined as $l(n) = 2(q(n)+p(n)+1)$. Further, we denote by $\Sigma^{\preceq n}$ the set $\{y \mid \exists x \in \Sigma^{\leq n}, y \preceq x\}$. Then it is clear that $\Sigma^{\leq n} \subseteq \Sigma^{\preceq n} \subseteq \Sigma^{\leq q(n)}$. A description of T is given below.

input 0^n
$S := \emptyset$
loop
  for $k = 1, \ldots, p(n)$, choose H_k randomly from $\mathcal{H}(l(n), p(n), k)$,
  $k_{max} := \max\{k \mid \mathit{Consistent}(S)$ has a collision w.r.t. $H_k\}$
  if there exists an $x \in \Sigma^{\preceq n}$ such that *Consistent*(S) has
    a half-collision on *Incorrect*(x, S) w.r.t. $H_{k_{max}}$
  then
    use oracle O to find such a string x and to determine $A(x)$
    $S := S\#\langle x, A(x)\rangle$
  else exit(**loop**) **end**
end loop
output $\langle S, H_{k_{max}}\rangle$

Starting with the empty sample, T enters the main loop. During each execution of the loop, T first randomly guesses a series of $p(n)$ many hash families $H_k \in \mathcal{H}(l(n), p(n), k), 1 \leq k \leq p(n)$. Then T computes the integer k_{max} as the maximum $k \in \{1, \ldots, p(n)\}$ such that *Consistent*(S) has a collision w.r.t. H_k. Notice that by a padding trick we can assume that $c(S)$ is always larger than $2l(n)$, implying that *Consistent*(S) must have a collision w.r.t. H_1. Since, in particular, *Consistent*(S) has a collision w.r.t. $H_{k_{max}}$, it follows that for every instance $x \in \Sigma^{\preceq n}$, *Consistent*(S) has a half-collision w.r.t. $H_{k_{max}}$ on either *Correct*(x, S) or *Incorrect*(x, S). If there exists a string $x \in \Sigma^{\preceq n}$ such that *Consistent*(S) has a half-collision on *Incorrect*(x, S) w.r.t. $H_{k_{max}}$, then this string is added to the sample S, and T continues executing the loop. (We will describe below how T uses the NP oracle O to find x in this case.) Otherwise, the pair $\langle S, H_{k_{max}} \rangle$ fulfills the properties stated in condition (b) and T halts.

We now show that the expected running time of T is polynomially bounded. Since the initial size of *Consistent*(S) is $2^{p(n)}$, and since *Consistent*(S) never becomes empty, it suffices to prove that for some polynomial r, T eliminates in each single execution of the main loop with probability at least $1/r(n)$ at least a $1/r(n)$-fraction of the circuits in *Consistent*(S). In fact, we will show that each single extension of S by a pair $\langle x, A(x) \rangle$ reduces the size of *Consistent*(S) with probability at least $1 - 2^{-l(n)}$ by a factor smaller than $1 - 1/2^7 l(n)$. Since T can only perform more than $2^7 l(n) p(n)$ loop iterations if during some iteration of the main loop T extends S by a pair $\langle x, A(x) \rangle$ which does not shrink the size of *Consistent*(S) by a factor smaller than $1 - 1/2^7 l(n)$, the probability for this event is bounded by $2^7 l(n) p(n) \cdot 2^{-l(n)} = o(1)$.

Let S be a sample and let k_{max} be the corresponding integer as determined by T during some specific execution of the loop. We first derive a lower bound for k_{max}. Let k_0 be the smallest integer $k \geq 1$ such that $c(S) \leq l(n) 2^{k+1}$. Since either $k_{max} = p(n)$ or *Consistent*(S) does not have a collision w.r.t. the hash family $H_{k_{max}+1} \in \mathcal{H}(l(n), p(n), k_{max} + 1)$, we have (using Theorem 2.3) that $c(S) \leq l(n) 2^{k_{max}+1}$. Hence, $k_{max} \geq k_0$.

Since T expands S only by strings $x \in \Sigma^{\preceq n}$ such that *Consistent*(S) has a half-collision on *Incorrect*(x, S) w.r.t. $H_{k_{max}}$, and since *Consistent*$(S \# \langle x, A(x) \rangle) =$ *Consistent*$(S) -$ *Incorrect*(x, S), the probability that the size of *Consistent*(S) does not decrease by a factor smaller than $1 - 1/2^7 l(n)$ is bounded by the probability that, w.r.t. $H_{k_{max}}$, *Consistent*(S) has a half-collision on some set *Incorrect*(x, S) of size at most $c(S)/2^7 l(n)$. Let

$$\mathcal{C} = \{\textit{Incorrect}(x, S) \mid x \in \Sigma^{\preceq n}, |\textit{Incorrect}(x, S)| \leq c(S)/2^7 l(n)\}.$$

Since $|\mathcal{C}| < 2^{q(n)+1}$ and since $c(S)/2^7 l(n) \leq 2^{k_0 - 6} = 2^{k_0 + k - (k+4) - 2}$, it follows from Theorem 2.7 that the probability of *Consistent*(S) having a half-collision on some $Y \in \mathcal{C}$ w.r.t. a uniformly at random chosen hash family $H \in \mathcal{H}(l(n), p(n), k_0 + k)$ is at most

$$\begin{aligned} c(S) \cdot 2^{q(n)+1-(k+4)l(n)} &\leq 2^{p(n)+q(n)+1-(k+4)l(n)/2} \\ &= 2^{-(k+3)l(n)/2}. \end{aligned}$$

Thus the probability that for some $k \geq 0$, *Consistent*(S) has a half-collision w.r.t. H_{k_0+k} on some set *Incorrect*(x, S) which is of size at most $c(S)/2^7 l(n)$ is bounded by $\sum_{k \geq 0} 2^{-(k+3)l(n)/2} \leq 2^{-l(n)}$.

We finally show how T determines an instance $x \in \Sigma^{\preceq n}$ (if it exists) such that *Consistent*(S) has a half-collision on *Incorrect*(x, S) w.r.t. $H_{k_{max}}$. Intuitively, we use

the self-reducibility of A to test the "correctness" w.r.t. A of the "program" $\langle S, H_{k_{max}}\rangle$, where we say that

- a pair $\langle S, H\rangle$ accepts an instance x if *Consistent*(S) has a half-collision on *Accept*(x, S) w.r.t. H, and
- $\langle S, H\rangle$ rejects x if *Consistent*(S) has a half-collision on *Reject*(x, S) w.r.t. H.

Notice that an (incorrect) program might accept and at the same time reject an instance. The main idea to find out whether $\langle S, H_{k_{max}}\rangle$ is incorrect on some instance $x \in \Sigma^{\preceq n}$ (meaning that w.r.t. $H_{k_{max}}$ *Consistent*(S) has a half-collision on *Incorrect*(x, S)) is to test whether the program $\langle S, H_{k_{max}}\rangle$ is in accordance with the output of M_{self} when the oracle queries of M_{self} are answered according to the program $\langle S, H_{k_{max}}\rangle$. To be more precise, consider the NP set

$$\begin{aligned} B \;=\; \{\langle z, S, H\rangle \mid & \text{there is a computation path } \pi \text{ of } M_{self} \text{ on input } z \text{ fulfilling} \\ & \text{the following properties:} \\ & \text{- if a query } q \text{ is answered "yes," then } \langle S, H\rangle \text{ accepts } q, \\ & \text{- if a query } q \text{ is answered "no," then } \langle S, H\rangle \text{ rejects } q, \\ & \text{- if } \pi \text{ is accepting, then } \langle S, H\rangle \text{ rejects } z, \text{ and} \\ & \text{- if } \pi \text{ is rejecting, then } \langle S, H\rangle \text{ accepts } z\,\}. \end{aligned}$$

Then, as shown by the next claim, the correctness of $\langle S, H_{k_{max}}\rangle$ on an instance z can be decided by asking whether $\langle z, S, H_{k_{max}}\rangle$ belongs to B, provided that $\langle S, H_{k_{max}}\rangle$ is correct on all potential queries of M_{self} on input z.

CLAIM. *Assume that $\langle S, H_{k_{max}}\rangle$ is correct on all $y \prec z$. Then $\langle S, H_{k_{max}}\rangle$ is incorrect on z if and only if $\langle z, S, H_{k_{max}}\rangle$ belongs to B.*

Proof. Using the fact that for every instance $x \in \Sigma^{\preceq n}$, *Consistent*(S) has a half-collision w.r.t. $H_{k_{max}}$ on either *Correct*(x, S) or *Incorrect*(x, S), it is easy to see that if $\langle S, H_{k_{max}}\rangle$ is incorrect on z, then the computation path π followed by $M_{self}(z)$ under oracle A witnesses $\langle z, S, H_{k_{max}}\rangle \in B$. For the converse, assume that $\langle z, S, H_{k_{max}}\rangle$ belongs to B and let π be a computation path witnessing this fact. Note that all queries q on π are answered correctly w.r.t. A, since otherwise $\langle S, H_{k_{max}}\rangle$ were incorrect on $q \prec z$. Hence, π is the path followed by $M_{self}(z)$ under oracle A and therefore decides z correctly. On the other hand, since π witnesses $\langle z, S, H_{k_{max}}\rangle \in B$, $\langle S, H_{k_{max}}\rangle$ indeed is incorrect on z. □

Now we can define the oracle set O as $C \oplus D$, where

$$\begin{aligned} C \;=\; \{\langle 0^n, x, k, S, H\rangle \mid & \text{there is a } \succ\text{-chain of length (at least) } k \text{ from some} \\ & \text{string } y \in \Sigma^{\leq n} \text{ to some string } z \leq x \text{ such that } \langle z, S, H\rangle \in B\,\} \end{aligned}$$

and

$$\begin{aligned} D \;=\; \{\langle x, S, H\rangle \mid & \text{there is an accepting computation path } \pi \text{ of } M_{self} \text{ on} \\ & \text{input } x \text{ such that any query } q \text{ is only answered "yes" ("no") if} \\ & \textit{Consistent}(S) \text{ has a half-collision on } \textit{Accept}(q, S) \text{ (resp., } \textit{Reject}(q, S)) \\ & \text{w.r.t. } H\,\}. \end{aligned}$$

Note that the proof of the claim above also shows that for any $z \in \Sigma^{\preceq n}$ such that $\langle S, H_{k_{max}}\rangle$ is correct on all $y \prec z$, $z \in A$ if and only if $\langle z, S, H_{k_{max}}\rangle$ belongs to D. Now we can complete the description of T. T first asks whether the string $\langle 0^n, 1^{q(n)}, 0, S, H_{k_{max}}\rangle$ belongs to C. It is clear that a negative answer implies that $\langle S, H_{k_{max}}\rangle$ is correct on $\Sigma^{\preceq n}$. Otherwise, by asking queries of the form $\langle 0^n, 1^{q(n)}, i, S, H_{k_{max}}\rangle$, T computes by binary search i_{max} as the maximum value $i \leq 2^{q(n)+1}$ such that

input 0^n
$S := \emptyset$
loop
  for $k = 1, \ldots, p(n)$, choose H_k randomly from $\mathcal{H}(l(n), p(n), k)$,
  $k_{max} := \max\{k \mid \mathit{Consistent}(S)$ has a collision w.r.t. $H_k\}$
  if $\langle 0^n, 1^{q(n)}, 0, S, H_{k_{max}}\rangle \in C$ **then**
    $i_{max} := \max\{i \mid \langle 0^n, 1^{q(n)}, i, S, H_{k_{max}}\rangle \in C\}$
    $x_{min} := \min\{x \mid \langle 0^n, x, i_{max}, S, H_{k_{max}}\rangle \in C\}$
    $S := S\#\langle x, D(x_{min}, S, H_{k_{max}})\rangle$
  else exit(**loop**) **end**
end loop
output $\langle S, H_{k_{max}}\rangle$

$\langle 0^n, 1^{q(n)}, i, S, H_{k_{max}}\rangle$ belongs to C (a similar idea is used in [27]). Knowing i_{max}, T then determines the lexicographically smallest string x_{min} such that $\langle 0^n, x_{min}, i_{max}, S, H_{k_{max}}\rangle$ is in C. Since $\langle q, S, H_{k_{max}}\rangle \notin B$ holds for all instances $q \prec x_{min}$, it follows inductively from the claim that $\langle S, H_{k_{max}}\rangle$ is correct on all $q \prec x_{min}$. Hence, $\langle S, H_{k_{max}}\rangle$ must be incorrect on x_{min}, and, furthermore, T can determine the membership of x_{min} to A by asking whether the string $\langle x_{min}, S, H_{k_{max}}\rangle$ belongs to D. □

THEOREM 3.2. *Every self-reducible set A in the class* $(\mathrm{NP} \cap \text{co-NP})/\mathrm{poly}$ *is low for* $\mathrm{ZPP(NP)}$.

Proof. We first show that $\mathrm{NP}(A) \subseteq \mathrm{ZPP(NP)}$. Let L be a set in $\mathrm{NP}(A)$, and let M be a deterministic polynomial-time oracle machine such that for some polynomial t,

$$L = \{x \mid \exists y \in \Sigma^{t(|x|)} : \langle x, y\rangle \in L(M, A)\}.$$

Let $s(n)$ be a polynomial bounding the length of all oracle queries of M on some input $\langle x, y\rangle$ where x is of length n. Then L can be accepted by a probabilistic oracle machine N using the following NP oracle

$$O' = \{\langle x, S, H\rangle \mid \text{there is a } y \in \Sigma^{t(|x|)} \text{ such that } M \text{ on input } \langle x, y\rangle \text{ has an accepting path } \pi \text{ on which each query } q \text{ is answered "yes" ("no") only if } \mathit{Consistent}(S) \text{ has a half-collision on } \mathit{Accept}(q, S) \text{ (resp., } \mathit{Reject}(q, S)) \text{ w.r.t. } H\}.$$

Here is how N accepts L. On input x, N first simulates T on input $0^{s(|x|)}$ to compute a pair $\langle S, H_{k_{max}}\rangle$ as described above (T asks questions to some NP oracle O). Then N asks the query $\langle x, S, H_{k_{max}}\rangle$ to O' to find out whether x is in L.

This proves that $\mathrm{NP}(A) \subseteq \mathrm{ZPP(NP)}$. Since $\mathrm{ZPP(ZPP)} = \mathrm{ZPP}$ [41] via a proof that relativizes, it follows that $\mathrm{ZPP(NP}(A))$ is also contained in $\mathrm{ZPP(NP)}$, showing that A is low for $\mathrm{ZPP(NP)}$. □

4. Collapse consequences. As a direct consequence of Theorem 3.2 we get an improvement of Karp, Lipton, and Sipser's result in [22] that NP is not contained in P/poly unless the polynomial-time hierarchy collapses to Σ_2^{P}.

COROLLARY 4.1. *If* NP *is contained in* $(\mathrm{NP} \cap \text{co-NP})/\mathrm{poly}$ *then the polynomial-time hierarchy collapses to* $\mathrm{ZPP(NP)}$.

Proof. Since the NP-complete set SAT is self-reducible, the assumption that NP is contained in $(\mathrm{NP} \cap \text{co-NP})/\mathrm{poly}$ implies that SAT is low for $\mathrm{ZPP(NP)}$, and hence the polynomial-time hierarchy collapses to $\mathrm{ZPP(NP)}$. □

The collapse of the polynomial-time hierarchy deduced in Corollary 4.1 is quite close to optimal, at least in some relativized world [17, 39]: there is an oracle relative to which NP is contained in P/poly, but the polynomial-time hierarchy does not collapse to P(NP).

In the rest of this section we report some other interesting collapses which can be easily derived using (by now) standard techniques, and which have also been pointed out to the second author independently by several researchers. First, it is straightforward to check that Theorem 3.2 relativizes: for any oracle B, if A is a self-reducible set in the class $(\mathrm{NP}(B) \cap \text{co-NP}(B))/\mathrm{poly}$, then $\mathrm{NP}(A)$ is contained in $\mathrm{ZPP}(\mathrm{NP}(B))$. Consequently, Theorem 3.2 generalizes to the following result.

THEOREM 4.2. *If A is a self-reducible set in the class $(\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/\mathrm{poly}$, then $\mathrm{NP}(A) \subseteq \mathrm{ZPP}(\Sigma_k^{\mathrm{P}})$.*

As a direct consequence of Theorem 4.2 we get an improvement of results in [1, 20] stating (for $k = 1$) that Σ_k^{P} is not contained in $(\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/\mathrm{poly}$ unless the polynomial-time hierarchy collapses to $\Sigma_{k+1}^{\mathrm{P}}$.

COROLLARY 4.3. *Let $k \geq 1$. If Σ_k^{P} is contained in $(\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/\mathrm{poly}$, then the polynomial-time hierarchy collapses to $\mathrm{ZPP}(\Sigma_k^{\mathrm{P}})$.*

Proof. Since Σ_k^{P} contains complete self-reducible languages, the assumption that Σ_k^{P} is contained in $(\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/\mathrm{poly}$ implies that $\Sigma_{k+1}^{\mathrm{P}} = \mathrm{NP}(\Sigma_k^{\mathrm{P}}) \subseteq \mathrm{ZPP}(\Sigma_k^{\mathrm{P}})$. □

Yap [40] proved that Π_k^{P} is not contained in $\Sigma_k^{\mathrm{P}}/\mathrm{poly}$ unless the polynomial-time hierarchy collapses to $\Sigma_{k+2}^{\mathrm{P}}$. As a further consequence of Theorem 4.2 we get the following improvement of Yap's result.

COROLLARY 4.4. *For $k \geq 1$, if $\Pi_k^{\mathrm{P}} \subseteq \Sigma_k^{\mathrm{P}}/\mathrm{poly}$, then $\mathrm{PH} = \mathrm{ZPP}(\Sigma_{k+1}^{\mathrm{P}})$.*

Proof. The assumption that Π_k^{P} is contained in $\Sigma_k^{\mathrm{P}}/\mathrm{poly}$ implies that $\Sigma_{k+1}^{\mathrm{P}}$ is contained in $\Sigma_k^{\mathrm{P}}/\mathrm{poly} \subseteq (\Sigma_{k+1}^{\mathrm{P}} \cap \Pi_{k+1}^{\mathrm{P}})/\mathrm{poly}$. Hence we can apply Corollary 4.3. □

As corollaries to Theorem 4.2, we also have similar collapse results for many other complexity classes. What follows are some typical examples.

COROLLARY 4.5. *For $\mathcal{K} \in \{\mathrm{UP}, \mathrm{FewP}\}$, if $\mathcal{K} \subseteq (\mathrm{NP} \cap \text{co-NP})/\mathrm{poly}$ then $\mathcal{K}$ is low for $\mathrm{ZPP}(\mathrm{NP})$.*

Proof. It is well known that for every set A in UP (FewP), the left set of A [30] is word-decreasing self-reducible and in UP (resp., FewP). Thus, under the assumption that $\mathrm{UP} \subseteq (\mathrm{NP} \cap \text{co-NP})/\mathrm{poly}$ (resp., $\mathrm{FewP} \subseteq (\mathrm{NP} \cap \text{co-NP})/\mathrm{poly}$) it follows by Theorem 3.2 that the left set of A (and since A is polynomial-time many-one reducible to its left set, also A) is low for $\mathrm{ZPP}(\mathrm{NP})$. □

COROLLARY 4.6. *For every $k \geq 1$, if $\mathrm{C_{=}P} \subseteq (\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/\mathrm{poly}$ then $\mathrm{CH} = \mathrm{ZPP}(\Sigma_k^{\mathrm{P}})$.*

Proof. First, since $\mathrm{C_{=}P}$ has complete word-decreasing self-reducible languages [29], $\mathrm{C_{=}P} \subseteq (\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/\mathrm{poly}$ implies $\mathrm{C_{=}P} \subseteq \mathrm{ZPP}(\Sigma_k^{\mathrm{P}}) \subseteq \mathrm{PH}$. Second, since $\mathrm{PH} \subseteq \mathrm{BPP}(\mathrm{C_{=}P})$ [37, 35], $\mathrm{C_{=}P} \subseteq (\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/\mathrm{poly}$ implies $\mathrm{PH} \subseteq (\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/\mathrm{poly}$ and therefore PH collapses to $\mathrm{ZPP}(\Sigma_k^{\mathrm{P}})$ by Corollary 4.3. Finally, since $\mathrm{C_{=}P}(\mathrm{PH}) \subseteq \mathrm{BPP}(\mathrm{C_{=}P})$ [37], it follows that $\mathrm{C_{=}P}(\mathrm{PH}) \subseteq \mathrm{PH}$, and since $\mathrm{CH} = \mathrm{C_{=}P} \cup \mathrm{C_{=}P}(\mathrm{C_{=}P}) \cup \ldots$ (see [38]), we get inductively that $\mathrm{CH} \subseteq \mathrm{PH}$ $(\subseteq \mathrm{ZPP}(\Sigma_k^{\mathrm{P}}))$. □

COROLLARY 4.7. *Let $\mathcal{K} \in \{\mathrm{EXP}, \mathrm{PSPACE}, \mathrm{Mod}_m\mathrm{P}\}$, $m \geq 2$. If for some $k \geq 1$, $\mathcal{K} \subseteq (\Sigma_k^{\mathrm{P}} \cap \Pi_k^{\mathrm{P}})/\mathrm{poly}$, then $\mathcal{K} \subseteq \mathrm{PH}$ and PH collapses to $\mathrm{ZPP}(\Sigma_k^{\mathrm{P}})$.*

Proof. The proof for $\mathcal{K} \in \{\mathrm{EXP}, \mathrm{PSPACE}\}$ is immediate from Theorem 4.2 since PSPACE has complete (length-decreasing) self-reducible languages and since EXP has complete (word-decreasing) self-reducible languages [6].

The proof for $\mathcal{K} \in \{\mathrm{Mod}_m\mathrm{P} \mid m \geq 2\}$ is analogous to the one of Corollary 4.6

using the fact that $\mathrm{Mod}_m\mathrm{P}$ has complete word-decreasing self-reducible languages [29], and that $\mathrm{PH} \subseteq \mathrm{BPP}(\mathrm{Mod}_m\mathrm{P})$ [37, 35]. □

Since our proof technique is relativizable, the above results hold for every relativized world. On the other hand, it is known that for some classes stronger collapse consequences can be obtained by using nonrelativizable arguments.

THEOREM 4.8 (see [28, 4, 3]). *For* $\mathcal{K} \in \{\mathrm{PP}, \mathrm{Mod}_m\mathrm{P}, \mathrm{PSPACE}, \mathrm{EXP}\}$, *if* $\mathcal{K} \subseteq \mathrm{P/poly}$ *then* $\mathcal{K} \subseteq \mathrm{MA}$.

Harry Buhrman pointed out to us that Corollary 4.7 can also be derived from Theorem 4.8.

5. Circuit complexity. Kannan [21] proved that for every fixed polynomial s, there is a set in $\Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}}$ which cannot be decided by circuits of size $s(n)$. Using a padding argument, he obtained the existence of sets in $\mathrm{NEXP(NP)} \cap \text{co-NEXP(NP)}$ not having polynomial-size circuits.

THEOREM 5.1 (see [21]).

1. *For every polynomial* s, *there is a set in* $\Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}}$ *that does not have circuits of size* $s(n)$.
2. *For every increasing time-constructible super-polynomial function* $f(n)$, *there is a set in* $\mathrm{NTIME}[f(n)](\mathrm{NP}) \cap \text{co-NTIME}[f(n)](\mathrm{NP})$ *that does not have polynomial size circuits.*

As an application of our results in section 3, we can improve Kannan's results in every relativized world from the class $\Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}}$ to $\mathrm{ZPP(NP)}$ and from the class $\mathrm{NTIME}[f(n)](\mathrm{NP}) \cap \text{co-NTIME}[f(n)](\mathrm{NP})$ to $\mathrm{ZPTIME}[f(n)](\mathrm{NP})$, respectively. Here $\mathrm{ZPTIME}[f(n)](\mathrm{NP})$ denotes the class of all sets that are accepted by some zero error probabilistic machine in expected running time $O(f(n))$ relative to some NP oracle.

Note that for all sets in the class P/poly we may fix the interpreter set to some appropriate one in P. Let I_{univ} denote such a fixed interpreter set. Furthermore, P/poly remains the same class if we relax the notion of an advice function h (w.r.t. I_{univ}) as follows: for every x, $A(x) = I_{univ}(x, h(|x|))$; i.e., $h(n)$ has to decide correctly only $A^{=n}$ (instead of $A^{\leq n}$).

A sequence of circuits C_n, $n \geq 0$, is called a circuit family for A if for every $n \geq 0$, C_n has n input gates, and for all n-bit strings $x_1 \cdots x_n$, $C_n(x_1, \ldots, x_n) = A(x_1 \cdots x_n)$. It is well known (see, e.g., [9]) that I_{univ} can be chosen in such a way that advice length and circuit size (i.e., number of gates) are polynomially related to each other. More precisely, we can assume that there is a polynomial p such that the following holds for every set A.

- If h is an advice function for A w.r.t. I_{univ}, then there exists a circuit family C_n, $n \geq 0$, for A of size $|C_n| \leq p(n + |h(n)|)$.
- If C_n, $n \geq 0$, is a circuit family for A, then there exists an advice function h for A w.r.t. I_{univ} of length $|h(n)| \leq p(|C_n|)$.

Moreover, we can assume that for every polynomial-time interpreter set I there is a constant c_I such that if h is an advice function for A w.r.t. I, then there exists an advice function h' for A w.r.t. I_{univ} of length $|h'(n)| \leq |h(n)| + c_I$ for all n.

The following lemma is obtained by a direct diagonalization (cf. the corresponding result in [21]). A set S is called $\mathrm{P}^{\mathcal{C}}$-printable (see [16]) if there is a polynomial-time oracle transducer T and an oracle set $A \in \mathcal{C}$ such that on any input 0^n, T^A outputs a list of all strings in $S^{\leq n}$.

LEMMA 5.2. *For every fixed polynomial* s, *there is a* Δ_3^{P}*-printable set* A *such that every advice function* h *for* A *is of length* $|h(n)| \geq s(n)$ *for almost all* n.

Proof. For a given n, let $x_1, x_2, \ldots, x_{2^n}$ be the sequence of strings of length n, enumerated in lexicographic order. Consider the two sets *Have-Advice* and *Find-A* defined as follows:

$$\langle n, a_1 \cdots a_{s(n)} \rangle \in \textit{Have-Advice} \Leftrightarrow \exists\, w \in \Sigma^{<s(n)},\, \forall\, i, 1 \le i \le \min(s(n), 2^n) : a_i = I_{univ}(x_i, w),$$

$$\langle n, a_1 \cdots a_j 10^{s(n)-j} \rangle \in \textit{Find-A} \Leftrightarrow \exists\, a_{j+1} \cdots a_{s(n)} : \langle n, a_1 \cdots a_j a_{j+1} \cdots a_{s(n)} \rangle \notin \textit{Have-Advice}.$$

Since there are only $2^{s(n)} - 1$ advice strings w in $\Sigma^{<s(n)}$, at least one pair of the form $\langle n, a_1 \cdots a_{s(n)} \rangle$ is not contained in *Have-Advice* (provided that $s(n) \le 2^n$). Let α_n denote the lexicographically smallest such pair $\langle n, a_1 \cdots a_{s(n)} \rangle$; i.e., there is no advice of length smaller than $s(n)$ that accepts the strings $x_1, \ldots, x_{s(n)}$ according to $a_1, \ldots, a_{s(n)}$.

Define A as the set of all strings x_i ($|x_i| = n$) such that $1 \le i \le s(n) \le 2^n$ and the ith bit of α_n (i.e., a_i) is 1. By a binary search using oracle *Find-A*, α_n is computable in polynomial time. Since *Have-Advice* is in NP and thus *Find-A* is in NP(NP), it follows that A is P(NP(NP))-printable. Since, furthermore, for almost all n, $A^{=n}$ has no advice of length smaller than $s(n)$, the lemma follows. □

Corollary 5.3. *For every fixed polynomial s, there is a set A in* ZPP(NP) *that does not have circuits of size $s(n)$.*

Proof. If NP does not have polynomial-size circuits, then we can take $A = \mathrm{SAT}$. Otherwise, PH = ZPP(NP) by Corollary 4.1, and thus the theorem easily follows from Lemma 5.2. □

Corollary 5.4. *Let f be an increasing, time-constructible, super-polynomial function. Then* ZPTIME$[f(n)]$(NP) *contains a set A that does not have polynomial-size circuits.*

Proof. If NP does not have polynomial-size circuits, then we can take $A = \mathrm{SAT}$. Otherwise, PH = ZPP(NP) by Corollary 4.1, and thus it follows from Lemma 5.2 that there is a set B in ZPTIME$[n^k]$(NP) such that every advice function h for B is of length $|h(n)| \ge n$ for almost all n. By the proof technique of Lemma 5.2, we can assume that in all length n strings of B, 1's only occur at the $O(\log n)$ rightmost positions. Now consider the following set (where n denotes $|x|$)

$$A = \{x \mid 0^{\lfloor f(n)^{1/k} \rfloor - n} x \in B\}$$

and the interpreter set

$$I = \{\langle 0^{\lfloor f(n)^{1/k} \rfloor - n} x, w \rangle \mid \langle x, w \rangle \in I_{univ}\}.$$

Clearly, A belongs to ZPTIME$[f(n)]$(NP) and I belongs to P. Furthermore, if h is an advice function for A, then we have for every y of the form $0^{\lfloor f(n)^{1/k} \rfloor - n} x$, $|x| = n$, that

$$\begin{aligned} y \in B &\Leftrightarrow \langle y, h(n) \rangle \in I \\ &\Leftrightarrow \langle y, h'(n) \rangle \in I_{univ}, \end{aligned}$$

where $h'(n)$ is a suitable advice function of length $|h'(n)| \le |h(n)| + c_I$. Thus, it follows for almost all n that

$$|h(n)| \ge |h'(n)| - c_I \ge |y| - c_I = \lfloor f(n)^{1/k} \rfloor - c_I.$$

This shows that the length of h is super-polynomial. □

COROLLARY 5.5. *In every relativized world,* ZPEXP(NP) *contains sets that do not have polynomial-size circuits.*

We remark that the above results are proved by relativizable arguments. On the other hand, Harry Buhrman [12] and independently Thomas Thierauf [36] pointed out to us that Theorem 4.8 (which is proved by a nonrelativizable proof technique) can be used to show that $\mathrm{MA}_{\mathrm{exp}} \cap \text{co-MA}_{\mathrm{exp}}$ contains non P/poly sets. Here, $\mathrm{MA}_{\mathrm{exp}}$ denotes the exponential-time version of Babai's class MA [5]. That is, $\mathrm{MA}_{\mathrm{exp}} = \mathrm{MA}[2^{n^{O(1)}}]$, where a language L is in $\mathrm{MA}[f(n)]$ if there exists a set $B \in \mathrm{DTIME}[O(n)]$ such that for all x of length n,

$$\begin{aligned} x \in L &\Rightarrow \exists y, |y| = f(n) : \Pr[\langle x, y, z\rangle \in B] > 2/3, \\ x \notin L &\Rightarrow \forall y, |y| = f(n) : \Pr[\langle x, y, z\rangle \in B] < 1/3, \end{aligned}$$

where z is chosen uniformly at random from $\Sigma^{f(n)}$.

COROLLARY 5.6 (see [12, 36]). $\mathrm{MA}_{\mathrm{exp}} \cap \text{co-MA}_{\mathrm{exp}}$ *contains sets that do not have polynomial-size circuits.*

Since there exist recursive oracles relative to which all sets in EXP(NP) have polynomial-size circuits [39, 17], it is not possible to extend Corollary 5.5 by relativizing techniques to the class EXP(NP).

6. Concluding remarks. An interesting question concerning complexity classes $\mathcal{C}$ that are known to be not contained in P/poly but are not known to have complete sets is whether the existence of sets in $\mathcal{C} - \mathrm{P/poly}$ can be *constructively* shown. For example, by Corollary 5.5 we know that the class ZPEXP(NP) contains sets that do not have polynomial-size circuits. But we were not able to give a *constructive* proof of this fact. To the best of our knowledge, no explicit set is known even in $\mathrm{NEXP(NP)} \cap \text{co-NEXP(NP)} - \mathrm{P/poly}$.

Acknowledgments. We thank the referees for several remarks that improved this paper. For helpful discussions and suggestions regarding this work we are very grateful to H. Buhrman, R. Gavaldà, L. Hemaspaandra, M. Ogihara, U. Schöning, R. Schuler, and T. Thierauf. We like to thank H. Buhrman, L. Hemaspaandra, and M. Ogihara for permitting us to include their observations in the paper.

REFERENCES

[1] M. ABADI, J. FEIGENBAUM, AND J. KILIAN, *On hiding information from an oracle*, J. Comput. System Sci., 39 (1989), pp. 21–30.
[2] D. ANGLUIN, *Queries and concept learning*, Mach. Learning, 2 (1988), pp. 319–342.
[3] L. BABAI AND L. FORTNOW, *Arithmetization: A new method in structural complexity*, Comput. Complexity, 1 (1991), pp. 41–66.
[4] L. BABAI, L. FORTNOW, AND C. LUND, *Non-deterministic exponential time has two-prover interactive protocols*, Comput. Complexity, 1 (1991), pp. 1–40.
[5] L. BABAI AND S. MORAN, *Arthur-merlin games: A randomized proof system and a hierarchy of complexity classes*, J. Comput. System Sci., 36 (1988), pp. 254–276.
[6] J. BALCÁZAR, *Self-reducibility*, J. Comput. System Sci., 41 (1990), pp. 367–388.
[7] J. BALCÁZAR, R. BOOK, AND U. SCHÖNING, *The polynomial-time hierarchy and sparse oracles*, J. Assoc. Comput. Mach., 33 (1986), pp. 603–617.
[8] J. BALCÁZAR, R. BOOK, AND U. SCHÖNING, *Sparse sets, lowness and highness*, SIAM J. Comput., 23 (1986), pp. 679–688.
[9] J. BALCÁZAR, J. DÍAZ, AND J. GABARRÓ, *Structural Complexity* I, 2nd ed., Springer-Verlag, Berlin, New York, 1995.
[10] D. BOVET AND P. CRESCENZI, *Introduction to the Theory of Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1994.

[11] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon, *Oracles and queries that are sufficient for exact learning*, J. Comput. System Sci., 52 (1996), pp. 421–433.
[12] H. Buhrman, *Personal communication*, 1994.
[13] J. L. Carter and M. N. Wegman, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.
[14] R. Gavaldà, *Bounding the complexity of advice functions*, J. Comput. System Sci., 50 (1995), pp. 468–475.
[15] J. Gill, *Computational complexity of probabilistic complexity classes*, SIAM J. Comput., 6 (1977), pp. 675–695.
[16] J. Hartmanis and Y. Yesha, *Computation times of NP sets of different densities*, Theoret. Comput. Sci., 34 (1984), pp. 17–32.
[17] H. Heller, *On relativized exponential and probabilistic complexity classes*, Inform. and Control, 71 (1986), pp. 231–243.
[18] L. A. Hemachandra, M. Ogiwara, and O. Watanabe, *How hard are sparse sets?*, in Proceedings of the 7th Structure in Complexity Theory Conference, IEEE Computer Society Press, Piscataway, NJ, 1992, pp. 222–238.
[19] M. Jerrum, L. Valiant, and V. Vazirani, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.
[20] J. Kämper, *Non-uniform proof systems: A new framework to describe non-uniform and probabilistic complexity classes*, Theoret. Comput. Sci., 85 (1991), pp. 305–331.
[21] R. Kannan, *Circuit-size lower bounds and non-reducibility to sparse sets*, Inform. and Control, 55 (1982), pp. 40–56.
[22] R. M. Karp and R. J. Lipton, *Some connections between nonuniform and uniform complexity classes*, in Proceedings of the 12th ACM Symposium on Theory of Computing, ACM Press, New York, 1980, pp. 302–309.
[23] K. Ko, *On self-reducibility and weak p-selectivity*, J. Comput. System Sci., 26 (1983), pp. 209–221.
[24] J. Köbler, *Locating P/poly optimally in the extended low hierarchy*, Theoret. Comput. Sci., 134 (1994), pp. 263–285.
[25] J. Köbler, U. Schöning, and J. Torán, *The Graph Isomorphism Problem: Its Structural Complexity*, Birkhäuser, Boston, 1993.
[26] T. Long, *Strong nondeterministic polynomial-time reducibilities*, Theoret. Comput. Sci., 21 (1982), pp. 1–25.
[27] A. Lozano and J. Torán, *Self-reducible sets of small density*, Math. Systems Theory, 24 (1991), pp. 83–100.
[28] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, J. Assoc. Comput. Mach., 39 (1992), pp. 859–868.
[29] M. Ogiwara and A. Lozano, *On sparse hard sets for counting classes*, Theoret. Comput. Sci., 112 (1993), pp. 255–275.
[30] M. Ogiwara and O. Watanabe, *On polynomial-time bounded truth-table reducibility of NP sets to sparse sets*, SIAM J. Comput., 20 (1991), pp. 471–483.
[31] C. Papadimitriou, *Computational Complexity*, Addison–Wesley, Reading, MA, 1994.
[32] N. Pippenger, *On simultaneous resource bounds*, in Proceedings of the 20th IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society Press, Piscataway, NJ, 1979, pp. 307–311.
[33] U. Schöning, *Complexity and Structure*, Lecture Notes in Computer Science 211, Springer-Verlag, New York, 1986.
[34] M. Sipser, *A complexity theoretic approach to randomness*, in Proceedings of the 15th ACM Symposium on Theory of Computing, ACM Press, New York, 1983, pp. 330–335.
[35] J. Tarui, *Probabilistic polynomials, AC^0 functions, and the polynomial time hierarchy*, Theoret. Comput. Sci., 113 (1993), pp. 167–183.
[36] T. Thierauf, *Personal communication*, 1994.
[37] S. Toda and M. Ogiwara, *Counting classes are at least as hard as the polynomial-time hierarchy*, SIAM J. Comput., 21 (1992), pp. 316–328.
[38] J. Torán, *Complexity classes defined by counting quantifiers*, J. Assoc. Comput. Mach., 38 (1991), pp. 753–774.
[39] C. Wilson, *Relativized circuit complexity*, J. Comput. System Sci., 31 (1985), pp. 169–181.
[40] C. Yap, *Some consequences of non-uniform conditions on uniform classes*, Theoret. Comput. Sci., 26 (1983), pp. 287–300.
[41] S. Zachos, *Robustness of probabilistic computational complexity classes under definitional perturbations*, Inform. and Control, 54 (1982), pp. 143–154.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 325–340

SUBLOGARITHMIC BOUNDS ON SPACE AND REVERSALS*

VILIAM GEFFERT†, CARLO MEREGHETTI‡, AND GIOVANNI PIGHIZZINI‡

Abstract. The complexity measure under consideration is SPACE × REVERSALS for Turing machines that are able to branch both existentially and universally. We show that, for any function $h(n)$ between $\log\log n$ and $\log n$, Π_1SPACE × REVERSALS$(h(n))$ is separated from Σ_1SPACE × REVERSALS$(h(n))$ as well as from coΣ_1SPACE×REVERSALS$(h(n))$, for *middle*, *accept*, and *weak* modes of this complexity measure. This also separates determinism from the higher levels of the alternating hierarchy. For "well-behaved" functions $h(n)$ between $\log\log n$ and $\log n$, almost all of the above separations can be obtained by using *unary* witness languages.

In addition, the construction of separating languages contributes to the research on minimal resource requirements for computational devices capable of recognizing nonregular languages. For any (arbitrarily slow growing) unbounded monotone recursive function $f(n)$, a nonregular unary language is presented that can be accepted by a *middle* Π_1 alternating Turing machine in $s(n)$ space and $i(n)$ input head reversals, with $s(n) \cdot i(n) \in \mathcal{O}(\log\log n \cdot f(n))$. Thus, there is no exponential gap for the optimal lower bound on the product $s(n) \cdot i(n)$ between unary and general nonregular language acceptance—in sharp contrast with the one-way case.

Key words. alternation, computational complexity, computational lower bounds, formal languages

AMS subject classifications. 68Q05, 68Q15, 68Q68

PII. S0097539796301306

1. Introduction.

In the last few years, some exciting results have been obtained in the field of space bounded computations. First of all, we have a surprisingly short proof that nondeterministic space is closed under complementation [12, 21]. Among others, this result has a fundamental consequence on alternation, namely, the collapse of the alternating space hierarchy to the Σ_1 level.

It is worth noticing that these results were obtained for *strong* space complexity classes with $s(n) \in \Omega(\log n)$. (A Turing machine M works in *strong* space $s(n)$ if no reachable configuration, on any input of length n, uses space above $s(n)$.) The closure under complementation does not hold for *weak* space complexity classes above $\log n$ [23]. (*weak* space $s(n)$: for any accepted input of length n, there exists at least one accepting computation not using more space than $s(n)$.) Below $\log n$, moreover, the above alternating hierarchy is infinite [2, 6, 15]. For the sublogarithmic world, many other results, almost self-evident in the superlogarithmic case, either do not hold or the proofs are quite different and are often highly involved.

In this paper, we continue in this line of research toward the simplest possible nonregular complexity classes by investigating Turing machines having sublogarithmic bounds on the product of *space*—in its different definitions—by the *number of input head reversals*. Turing machines working even within such limited resources can still

*Received by the editors March 29, 1996; accepted for publication (in revised form) by J. Hartmanis February 18, 1997; published electronically June 15, 1998.

http://www.siam.org/journals/sicomp/28-1/30130.html

†Katedra Matematickej Informatiky, Univerzita P.J. Šafárika, Jesenná 5, 04154 Košice, Slovakia (geffert@kosice.upjs.sk). This research was supported by the Slovak Grant Agency for Science (VEGA), under contract "Combinational Structures and Complexity of Algorithms."

‡Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, via Comelico 39, 20135 Milano, Italy (mereghc@dsi.unimi.it, pighizzi@dsi.unimi.it). This research was partially supported by Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST).

recognize nonregular languages [3, 4], while the corresponding bound on the product of space by *work head* reversals must be at least linear [9].

Besides *strong* and *weak* space, mentioned above, some intermediate measures have also been proposed. (*accept* space $s(n)$: all accepting computations obey the space bound, *middle* space $s(n)$: all computations obey the space bound for each accepted input of length n. For more precise definitions, see section 2.) Such differences are irrelevant for fully space constructible bounds above $\log n$, but several results (see, e.g., [23, 4, 7]) witness that one must pay special attention to the actual definition when dealing with such limited resources as is sublogarithmic space.

For the product of space by input head reversals, we are able to show several separation results that are still unknown if sublogarithmic bounds *on space only* are considered. Namely, for each of the modes $c \in \{middle, accept, weak\}$ and any function $h(n)$ between $\log\log n$ and $\log n$, c-Π_1SPACE$\times$REVERSALS$(h(n))$ is separated from c-Σ_1SPACE$\times$REVERSALS$(h(n))$ and, somewhat more surprisingly, also from *co*-c-Σ_1SPACE$\times$REVERSALS$(h(n))$. (Here c-XSPACE$\times$REVERSALS$(h(n))$ denotes the class of languages accepted by $X \in \{\Sigma_k, \Pi_k\}$ machines of type $c \in \{strong, middle, accept, weak\}$ in $s(n)$ space and $i(n)$ input head reversals, satisfying $s(n) \cdot i(n) \in \mathcal{O}(h(n))$. We add prefix "*co*-" for complements of such languages, and we use $X = \mathrm{D}$ for deterministic Turing machines, i.e., for $\Sigma_0 = \Pi_0$.)

In other words, for *middle*, *accept*, or *weak* space$\times$input head reversals bounded Turing machines, the class of languages accepted by machines making only universal decisions does not coincide with the class of complements of languages recognizable by machines making only existential decisions. Further, we get that *weak*-DSPACE$\times$REVERSALS$(h(n))$ is properly included in *weak*-Π_1-, *co-weak*-Π_1-, *weak*-Σ_1-, and in *co-weak*-Σ_1SPACE$\times$REVERSALS$(h(n))$, for each $h(n)$ between $\log\log n$ and $\log n$.

The input head motion for machines accepting nonregular languages has been studied in [3, 4]. It turns out that the minimal resource requirements for machines accepting *unary*[1] languages become important: a recognizer, already having too little space to remember an input head position, must also cope with the lack of any structure on the input tape. So the problem arises of whether these results hold even if the corresponding language classes are restricted to unary languages. Using an additional assumption that the function $h(n)$ is "well behaved," we are able to show that the above separations hold even in the case of unary languages, except for the separation of *weak*-Π_1SPACE$\times$REVERSALS$(h(n))$ from *co-weak*-Σ_1SPACE$\times$REVERSALS$(h(n))$, which we leave as an open problem.

Here "well behaved" means $h(n) \in o(\log n)$ with $\frac{h(n)}{\log\log n}$ unbounded and monotone increasing. We only require $h(n)$ to be recursive but do not claim any kind of space constructibility.

The above separations are obtained by exhibiting, for any (arbitrarily slow growing) unbounded monotone recursive function $f(n)$, a nonregular unary language $\mathcal{L}_f$ that can be accepted by a *middle* $s(n)$ space and $i(n)$ input head reversals bounded Π_1 machine with $s(n) \cdot i(n) \in \mathcal{O}(\log\log n \cdot f(n))$. On the other hand, using mainly number theoretical and pumping arguments, we show that, for every $h(n) \in o(\log n)$, $\mathcal{L}_f$ does not belong to *weak*-Σ_1SPACE$\times$REVERSALS$(h(n))$.

The complexity of recognizing $\mathcal{L}_f$ also gives meaningful insights into the study of *minimal resource requirements* for computational devices recognizing nonregular languages, an important research area dating back to works by J. Hartmanis, P. Lewis,

[1] That is, built over a single letter alphabet.

and R. Stearns in 1965. In [14, 19, 10], the authors settled the problem of determining the minimal *strong* space requirement for one–way and two–way, deterministic and nondeterministic Turing machines that accept nonregular languages. Subsequently, the same problem has been widely studied for other and more general paradigms of computation and space notions (for *strong* alternation in [20], for *middle* machines in [22], for *accept* machines in [4], and for *weak* machines in [1, 13]). The analysis of computational lower bounds for nonregular languages is tightly related to the world of sublogarithmic space (see, e.g., [23, 7]) and, more particularly, plays an important role in revealing sharp differences among various space definitions [4].

In this regard, we shall concentrate on the problem of determining the optimal lower bound on the product space×input head reversals for *middle* alternating Turing machines that accept unary nonregular languages. The reason why we focus on the *middle* mode of acceptance is that *one-way middle* alternating Turing machines exhibit a very interesting behavior having no analogue in any of the other space bounded computational models considered, e.g., in [23, 4]. On the one hand, we have an optimal $\log\log n$ space lower bound for nonregular languages built on binary alphabets (hence, on general alphabets as well) [22]. On the other hand, a tight $\log n$ space lower bound is proved in [16] whenever we restrict our machines to accept *unary* nonregular languages (see also Table 1.1).

A problem left open in [4] asks whether the same gap situation holds for the lower bound on $s(n) \cdot i(n)$ for *middle* space×input head reversals bounded alternating machines accepting nonregular languages. For such machines, a tight $\log\log n$ lower bound on $s(n) \cdot i(n)$ for general nonregular languages is observed in [4]. Should we expect a corresponding exponential gap when accepting unary nonregular languages, as in the one-way case?

Here we provide a negative answer to this open question since, as stated before, for any unbounded monotone recursive function $f(n)$, we have a unary nonregular language $\mathcal{L}_f$, recognizable by a *middle* space×input head reversals bounded alternating Turing machine satisfying $s(n) \cdot i(n) \in \mathcal{O}(\log\log n \cdot f(n))$. With $f(n)$ being *arbitrarily slow growing*, we can approach the optimal $\log\log n$ lower bound for binary languages as much as we like.

Though this does not completely prove the optimality of the lower bound $s(n) \cdot i(n) \notin o(\log\log n)$ for *middle* alternating Turing machines recognizing unary nonregular languages, it shows that such a lower bound cannot be raised to any "well-behaved" function $g(n)$ above $\log\log n$, i.e., to any $g(n)$ that is recursive with $\frac{g(n)}{\log\log n}$ unbounded and monotone increasing. This definitively rules out the possibility of an exponential gap observed on the corresponding one-way devices between the general and the unary cases.

Table 1.1 briefly summarizes the lower bounds for *middle* space×input head reversals bounded Turing machines recognizing nonregular languages (see Theorem 3.1 and [4, 16]).

This paper is organized as follows: section 2 contains basic definitions, in particular, the basic notions of space complexity and the SPACE×REVERSALS resource measure. In section 3, we state our main result: we exhibit, for any (arbitrarily slow growing) unbounded monotone recursive function $f(n)$, a unary nonregular language $\mathcal{L}_f$, mentioned above, and analyze the complexity of recognizing $\mathcal{L}_f$. As a consequence, section 4 proves the above-claimed separation results for the unary case. Finally, in section 5, we improve these separations for the case of general alphabets.

TABLE 1.1

Best lower bounds obtained for $s(n) \cdot i(n)$ on middle *alternating and nondeterministic* (Σ_1) *Turing machines accepting nonregular languages* ($i(n) = 1$ *for one-way machines*). *These bounds are known to be optimal* [4, 16] *except the bound for two-way alternating Turing machines on unary inputs which is "quasi" optimal* (*see Theorem* 3.1).

	Unary languages	General languages
One-way alternating	$\log n$	$\log \log n$
One-way nondeterministic (Σ_1)	$\log n$	$\log n$
Two-way alternating	**lower:** $\log \log n$ **upper:** $\log \log n \cdot f(n)$	$\log \log n$
Two-way nondeterministic (Σ_1)	$\log n$	$\log n$

2. Preliminaries. In this section, we briefly recall some very basic definitions concerning space bounded models of computation. For more details, we refer to [4, 23]. Furthermore, we consider machines having simultaneous bounds on both working space and number of input head reversals, and we introduce SPACE$\times$REVERSALS complexity classes.

Let Σ^* be the set of all strings over an alphabet Σ. Given any language $L \subseteq \Sigma^*$, $L^c = \Sigma^* \setminus L$ denotes the *complement* of L. L is said to be *unary* (or *tally*) whenever it is built on an alphabet consisting of exactly one symbol (usually "1").

The Turing machine model we shall deal with has been presented in [14, 19] to study sublinear space bounded computations. It consists of a finite state control, a two-way read-only input tape (with input enclosed between two end markers), and a separate semi-infinite two-way read-write work tape (initially empty, containing only *blank* symbols). A *memory state* of a Turing machine is an ordered triple $m = (q, u, j)$, where q is a control state, u is a string of work tape symbols (the nonblank content of the work tape), and j is an integer satisfying $1 \leq j \leq |u| + 1$ (the position of the work tape head). A *configuration* is an ordered pair $c = (m, i)$, where m is a memory state and i is an integer denoting the position of the input head.

The reader is assumed to be familiar with the notion of *alternating Turing machine*, introduced in [5], which is, at the same time, a generalization of nondeterminism and parallelism. A Σ_k (Π_k) *machine* is an alternating Turing machine beginning its computation in an existential (universal, respectively) state, and making at most $k - 1$ switches between existential and universal states along each computation path on any input. It can be easily seen that Σ_1 machines are actually nondeterministic machines. It is stipulated that Σ_0 and Π_0 machines are *deterministic* machines.

Let us now review notions of *space complexity* in the literature. In what follows, the space used by a computation of an alternating machine is, by definition, the maximal number of work tape cells used by any configuration in the tree corresponding to that computation. (For deterministic and nondeterministic machines, the computation reduces to a single computation path.) Let M be a deterministic, nondeterministic, or alternating Turing machine. Then the following hold.

- M works in *strong* $s(n)$ space if and only if, for each input of length n, no reachable configuration uses more space than $s(n)$ [14, 19, 10].
- M works in *middle* $s(n)$ space if and only if, for each accepted input of length n, no reachable configuration uses more space than $s(n)$ [22].
- M works in *accept*[2] $s(n)$ space if and only if, for each accepted input of length n, each accepting computation uses at most space $s(n)$ [11, 4, 18].

[2]The designation "*accept*" has been adopted first in [4].

- M works in *weak* $s(n)$ space if and only if, for each accepted input of length n, there exists an accepting computation using at most space $s(n)$ [17, 1, 13].

The above definitions are given in increasing order of generality, as one may easily verify. We remark that, although several differences have been emphasized in the literature [23, 4, 7], the above space notions turn out to be equivalent when considering *fully space constructible*[3] bounds, e.g., "normal" functions above $\log n$: once the space limit $s(n)$ can be computed in advance, each computation consuming too much space may be aborted. Also notice that, for Π_1 machines, *middle*, *accept*, and *weak* notions coincide for arbitrary space complexities.

With a slight abuse of terminology, we will often say, for instance, "a *middle* alternating Turing machine" instead of "an alternating Turing machine working in *middle* space."

The other computational resource we are interested in is the number of *input head reversals*. In general, we say that a Turing machine M is *one-way* if it can never move its input head toward the left; otherwise M is a *two-way* device. To emphasize the role of input head motion, we introduce a bound $i(n)$ on the number of input head reversals. We always compute $i(n)$ by considering those computations by which the space is defined. Thus, for instance, we say that a *middle* alternating Turing machine works *simultaneously* in $s(n)$ space and $i(n)$ input head reversals if, for each accepted input of length n, no reachable configuration uses more than $s(n)$ work tape cells, nor can it be accessed by a path that reverses the input head direction more than $i(n)$ times. For technical reasons, we stipulate $i(n) = 1$ for one-way machines.

Throughout the rest of the paper, we use the following notation for complexity classes. Let $c \in \{strong, middle, accept, weak\}$ and $X \in \{\Sigma_k, \Pi_k\}$. Then, we define

$$c\text{-}X\mathrm{SPACE}\times\mathrm{REVERSALS}(h(n))$$

to be the class of the languages accepted by X machines of type c in $s(n)$ space and $i(n)$ input head reversals satisfying $s(n) \cdot i(n) \in \mathcal{O}(h(n))$. In particular, we let $X = \mathrm{D}$ for $\Sigma_0 = \Pi_0$, i.e., for deterministic Turing machines. By *co-c-*$X\mathrm{SPACE}\times\mathrm{REVERSALS}(h(n))$, we denote the class of the languages L^c such that $L \in c\text{-}X\mathrm{SPACE}\times\mathrm{REVERSALS}(h(n))$. Finally, the restriction of these classes to unary languages will be denoted by $c\text{-}X\mathrm{SPACE}\times\mathrm{REVERSALS}^{1^*}(h(n))$ or by *co-c-*$X\mathrm{SPACE}\times\mathrm{REVERSALS}^{1^*}(h(n))$.

3. A family of nonregular unary languages. In this section, we introduce, for each unbounded (arbitrarily slow growing) monotone increasing recursive function $f(n)$, a nonregular unary language $\mathcal{L}_f$ that can be accepted by an alternating Turing machine in *middle* $s(n)$ space with $i(n)$ input head reversals satisfying $s(n) \cdot i(n) \in \mathcal{O}(\log\log n \cdot f(n))$. Hence, the optimal $\log\log n$ lower bound on $s(n) \cdot i(n)$ for general nonregular languages acceptance can be arbitrarily approached by unary languages. Subsequently, we prove that $\mathcal{L}_f$ cannot be accepted by any nondeterministic Turing machine working in *weak* $s(n)$ space and $i(n)$ input head reversals such that $s(n)\cdot i(n) \in o(\log n)$. This implies the nonregularity of $\mathcal{L}_f$ and will later be used to separate several $\mathrm{SPACE}\times\mathrm{REVERSALS}$ complexity classes.

In what follows, p_i denotes the ith prime. Furthermore, $f : \mathbf{N} \to \mathbf{N}$ is assumed to be any (effectively given) unbounded monotone increasing recursive function. That is, we have a deterministic Turing machine A which, for any binary input x, prints

[3] A function $s(n)$ is said to be *fully space constructible* whenever there exists a deterministic Turing machine which, on any input of length n, uses exactly space $s(n)$.

out a binary representation of $f(x)$. (Alternatively, we can avoid any ambiguity by using, in any standard enumeration $A_1, A_2, \ldots$ of Turing machines, the first machine computing f. It will be seen later that the upper and lower bounds proved in this section hold for any choice of A.)

We are now ready for the definition of $\mathcal{L}_f$. For each $n \in \mathbf{N}$, we first define the following statements.

(i) Let p_i be the first prime not dividing n.
(ii) Let x be the smallest integer satisfying $x \geq 2^{p_i}$ and $f(x) \geq p_i$.
(iii) Let $y = \max\{x, 2^{s_1}, 2^{s_2}, \ldots, 2^{s_x}\}$, where s_i denotes the amount of work tape space used by machine A when computing the value of $f(i)$.

Then $1^n \in \mathcal{L}_f$ if and only if the following holds:

(iv) for all prime powers ${p_j}^k \leq y$, with $p_j \leq \sqrt{y}$, $j \neq i$, and $k \geq 1$, we have that ${p_j}^k$ divides n.

In other words, item (iv) states that $1^n \in \mathcal{L}_f$ if and only if all the prime powers ${p_j}^k \leq y$, with $p_j \leq \sqrt{y}$, $j \neq i$, and $k \geq 1$, divide n, $\log y$ denoting the amount of space claimed by A to compute any of the values $f(1), f(2), \ldots, f(x)$ or to represent the integer x (see item (iii)), and $x \geq 2^{p_i}$ being the smallest integer satisfying $f(x) \geq p_i$ as required in (ii). Note that such x must exist: take the first x such that $f(x) > \max\{p_i - 1, f(1), f(2), \ldots, f(2^{p_i} - 1)\}$, using the fact that $f(x)$ is unbounded.

The proof of nonregularity of $\mathcal{L}_f$ will be given later as a consequence of Theorem 3.2. Now we shall study the complexity of $\mathcal{L}_f$ on alternating machines.

THEOREM 3.1. *Let $\mathcal{L}_f$ be a unary language defined as above. Then $\mathcal{L}_f$ can be accepted by a middle alternating Turing machine within $s(n)$ space and $i(n)$ input head reversals satisfying*

$$s(n) \cdot i(n) \in \mathcal{O}\left(\log\log n \cdot f(n)\right) .$$

Proof. First, we shall determine possible values of n so that the string 1^n belongs to $\mathcal{L}_f$. Let p_i, x, and y be defined as stated in items (i) – (iii). Item (iv) in the definition of $\mathcal{L}_f$ requires that possible factorizations of n must be of the following form:

$$n = {p_1}^{\alpha_1+\beta_1} \cdot {p_2}^{\alpha_2+\beta_2} \cdot \ldots \cdot {p_{i-1}}^{\alpha_{i-1}+\beta_{i-1}} \cdot {p_{i+1}}^{\alpha_{i+1}+\beta_{i+1}} \cdot \ldots \cdot {p_s}^{\alpha_s+\beta_s} \cdot \nu \,, \tag{3.1}$$

where p_s represents the largest prime less than or equal to $\sqrt{y}$. The prime p_i does not appear in (3.1) since it does not divide n. Numbers α_j, with $j \in \{1, 2, \ldots, s\} \setminus \{i\}$, are the maximal exponents which the corresponding primes p_j can be raised to in order to have ${p_j}^{\alpha_j} \leq y$. It is easy to see that $\alpha_j = \lfloor \log_{p_j} y \rfloor$ and $\beta_j \geq 0$, where, for any given number z, $\lfloor z \rfloor$ denotes the greatest integer less than or equal to z. Finally, ν contains the (possibly empty) part of the factorization consisting of powers of those primes greater than $\sqrt{y}$.

By (3.1), we are able to obtain a relation between n and y. In fact, we observe that

$$n \geq \prod_{p \leq \sqrt{y},\, p \neq p_i} p^{\lfloor \log_p y \rfloor} \,, \tag{3.2}$$

where the product is taken over all primes not exceeding $\sqrt{y}$ and different from p_i. By noticing that $\log_p y \geq 2$ for $p \leq \sqrt{y}$, we get the following limitation:

$$\sqrt{y} = p^{\frac{1}{2} \cdot \log_p y} < p^{\lfloor \log_p y \rfloor} \,,$$

which, together with (3.2), yields

$$n > \prod_{p \le \sqrt{y},\ p \ne p_i} \sqrt{y} = y^{\frac{1}{2}\cdot(\pi(\sqrt{y})-1)}. \tag{3.3}$$

Here $\pi(z)$ denotes the number of primes not exceeding $\lfloor z \rfloor$. A well-known theorem due to P. Čebyšev (see [8, Thm. 7]) states that

$$c_1 \cdot \frac{z}{\log z} \le \pi(z) \le c_2 \cdot \frac{z}{\log z}$$

for some positive constants c_1 and c_2. We can use this in (3.3) and obtain

$$n > y^{\frac{1}{2}\cdot\left(c_1 \cdot \frac{\sqrt{y}}{\log \sqrt{y}} - 1\right)} = 2^{\frac{\log y}{2}\cdot\left(\frac{2c_1\sqrt{y}}{\log y} - 1\right)} > d^{\sqrt{y}} \tag{3.4}$$

for a suitable constant $d > 1$.

With these results in our hands, we are now ready to estimate the amount of space and input head reversals sufficient for a *middle* alternating Turing machine M to accept $\mathcal{L}_f$. On input 1^n, M runs a two-phase algorithm. First, it computes the smallest prime p_i not dividing n. Then, it checks the truth of predicate in item (iv).

PHASE 1. M deterministically computes p_i by means of the following routine:

/* input is 1^n */
$p := 2$
while $n \bmod p = 0$ **do**
    begin
        $p := p + 1$
        while p not a prime **do** $p := p + 1$
    end
/* now p contains p_i */

The amount $s_1(n)$ of space used in this phase equals the number of bits needed to represent p_i in binary notation, i.e., $s_1(n) \in \mathcal{O}(\log p_i)$. Furthermore, it is easy to see that, for each value of p, the test "$n \bmod p = 0$" can be accomplished by scanning the input only once. Therefore, the number $i_1(n)$ of input head reversals equals the number of primes not exceeding p_i. By Čebyšev's theorem, we obtain $i_1(n) \in \mathcal{O}(\frac{p_i}{\log p_i})$.

PHASE 2. M deterministically computes the smallest integer $x \ge 2^{p_i}$ such that $f(x) \ge p_i$ (p_i being already stored on the work tape during the former phase). To this purpose, M simply loops as follows:

$x := 1$
while $x < 2^{p_i}$ or $f(x) < p_i$ **do**
    $x := x + 1$

Note that M computes each of the values $f(1), f(2), \ldots, f(x)$ by simulating A; therefore, it marks off exactly $\max\{\log x, s_1, s_2, \ldots, s_x\} = \log y$ space on the work tape. Recall that s_i denotes the amount of space used by A when computing $f(i)$.

Subsequently, M *universally* generates all the prime powers whose binary representation takes at most $\log y$ bits and checks whether each ${p_j}^k \le y$, with $p_j \le \sqrt{y}$, $j \ne i$, and $k \ge 1$, divides n. Clearly, the space requirement in this phase is bounded by $s_2(n) \in \mathcal{O}(\log y)$.

For input head reversals, we observe that the computations of $f(1), f(2), \ldots, f(x)$ are performed deterministically with the input head parked at one end marker, and that the divisibility of n by each ${p_j}^k$ is tested universally in parallel by one input scan, whence $i_2(n) = 1$.

Let us now evaluate $s(n) \cdot i(n)$ *in case* 1^n *belongs to* $\mathcal{L}_f$. The algorithm uses a total amount of space and input head reversals along each computation path of M to be estimated as

$$s(n) = s_1(n) + s_2(n) \in \mathcal{O}(\log p_i + \log y) = \mathcal{O}(\log y)\,,$$
$$i(n) = i_1(n) + i_2(n) \in \mathcal{O}\left(\frac{p_i}{\log p_i}\right)\,,$$

using $p_i \leq \log x \leq \log y$ by (ii) and (iii) in the definition of $\mathcal{L}_f$. This gives

$$s(n) \cdot i(n) \in \mathcal{O}\left(\log y \cdot \frac{p_i}{\log p_i}\right)\,.$$

Items (ii) and (iii) in the definition of $\mathcal{L}_f$ require that $p_i \leq f(x)$ and $x \leq y$. Further, $\frac{t}{\log t}$ is monotone increasing. Hence, $\frac{f(x)}{\log f(x)}$ is also monotone increasing whenever $f(x)$ is monotone increasing. Therefore, in case of $1^n \in \mathcal{L}_f$, we are able to obtain the following bounds:

$$\log y \cdot \frac{p_i}{\log p_i} \leq \log y \cdot \frac{f(x)}{\log f(x)} \leq \log y \cdot \frac{f(y)}{\log f(y)}\,.$$

Moreover, inequality (3.4) states that $n > d^{\sqrt{y}}$, i.e., $y \leq k \cdot \log^2 n$ for a suitable positive constant k. Hence, for sufficiently large n, we get

$$\begin{aligned}\log y \cdot \frac{f(y)}{\log f(y)} &\leq \log(k \cdot \log^2 n) \cdot \frac{f(k \cdot \log^2 n)}{\log f(k \cdot \log^2 n)} \\ &\leq \log(k \cdot \log^2 n) \cdot \frac{f(n)}{\log f(n)} \;\in\; \mathcal{O}\left(\log\log n \cdot f(n)\right)\,,\end{aligned}$$

which completes the proof. □

As a consequence of Theorem 3.1, it turns out that, using unary languages, we can arbitrarily approach the optimal $\log\log n$ lower bound on $s(n) \cdot i(n)$ holding for general nonregular language acceptance by *middle* space×input head reversals bounded alternating Turing machines (see Table 1.1). Take, for instance, $f(x) = \log^* x = \min\{k \in \mathbf{N} \mid \log^{(k)} x \leq 1\}$, with $\log^{(0)} x = x$ and, for each $k \geq 1$, $\log^{(k)} x = \log^{(k-1)} \log x$.

We now show that the language $\mathcal{L}_f$ is nonregular. Actually, we prove a stronger result that, together with Theorem 3.1, will also allow us to obtain some separations in the case of weakly bounded computations.

THEOREM 3.2. *Let* $\mathcal{L}_f$ *be a unary language defined as above. Then for any* $h(n) \in o(\log n)$, $\mathcal{L}_f$ *is not in weak-*Σ_1SPACE×REVERSALS$^{1^*}(h(n))$.

Proof. Suppose, by contradiction, that $\mathcal{L}_f$ can be accepted by a *weak* nondeterministic machine M in $s(n)$ space and $i(n)$ input head reversals, with $s(n) \cdot i(n) = h(n) \in o(\log n)$. The number of different memory states of M not using more space than $s(n)$ on the work tape can be bounded by $c^{s(n)}$, where c is a constant dependent on the number of work tape symbols and finite-control states of M. Since $\lim_{n\to\infty} \frac{s(n)\cdot i(n)}{\log n} = 0$, there exists $n_0 \in \mathbf{N}$ such that

$$c^{s(n)\cdot i(n)} < \sqrt{n} \quad \text{for each } n \geq n_0\,. \tag{3.5}$$

Now, consider the string $1^{n'}$, where n' is defined in the following way:

(i) First, let $p_i \geq 5$ be a sufficiently large prime such that there is another prime $\tilde{p}$ between n_0 and p_i, i.e., $p_i > \tilde{p} > n_0$.
(ii) Let x be the smallest integer satisfying $x \geq 2^{p_i}$ and $f(x) \geq p_i$.
(iii) Now, let $y = \max\{x, 2^{s_1}, 2^{s_2}, \ldots, 2^{s_x}\}$, where s_i denotes the space used by the machine A to compute $f(i)$.
(iv) Finally, define

$$n' = \prod_{p \leq \sqrt{y},\ p \neq p_i} p^{\lfloor \log_p y \rfloor},$$

where the product is taken over all primes not exceeding $\sqrt{y}$ and different from p_i.

We want to show that $1^{n'} \in \mathcal{L}_f$. First, we prove that p_i is the first prime that does not divide n'. Clearly, p_i does not divide n'. For primes $p < p_i$, we obtain

$$p^{\lfloor \log_p y \rfloor + 1} > y \geq \sqrt{y} \geq \log y \geq \log x \geq p_i > p = p^1,$$

using (iii) and (ii) in the definition of n'. (The inequality $\sqrt{y} \geq \log y$ follows from $\log y \geq p_i \geq 5$ by (i).) Thus, $\lfloor \log_p y \rfloor > 0$ and $p < \sqrt{y}$; i.e., n' is divisible by p for each prime $p < p_i$.

At this point, to conclude that $1^{n'} \in \mathcal{L}_f$, it is enough to prove that each prime power ${p_j}^k \leq y$, with $p_j \leq \sqrt{y}$, $j \neq i$, and $k \geq 1$, divides n'. This can be immediately shown by observing that $k \leq \log_{p_j} y$ and $k \in \mathbf{N}$; hence,

$$k = \lfloor k \rfloor \leq \lfloor \log_{p_j} y \rfloor.$$

This gives that $1^{n'} \in \mathcal{L}_f$.

As a consequence, there must exist an accepting computation path $\mathcal{C}$ of M on the input $1^{n'}$, using at most $s(n')$ space and $i(n')$ input head reversals. By (i) in the definition of n', we have a prime $\tilde{p}$ satisfying $p_i > \tilde{p} > n_0$. Since p_i is the first prime not dividing n', $\tilde{p}$ divides n', and hence $n' \geq \tilde{p} > n_0$. Therefore, by (3.5), the bounds $s(n')$ and $i(n')$ must satisfy

$$c^{s(n')} \leq c^{s(n') \cdot i(n')} < \sqrt{n'} \leq n'. \tag{3.6}$$

Along the path $\mathcal{C}$, we can consider $r_1, r_2, \ldots, r_A$, the sequence of all configurations in which the input head scans either of the end markers. Let $b_1, e_1, b_2, e_2, \ldots, b_B, e_B$ be the subsequence of $r_1, r_2, \ldots, r_A$ such that, for each $j = 1, 2, \ldots, B$, machine M begins with the input head positioned at the left or right end marker in b_j, traverses across the entire input $1^{n'}$, and ends in e_j positioned at the opposite end marker, without visiting either of the end markers in the meantime. The segments of computation between e_j and b_{j+1} always return the input head back to the same end marker (or, possibly, $e_j = b_{j+1}$).

Since, by (3.6), the number of different memory states is bounded by $c^{s(n')} < n'$, the machine must enter *a loop* when traversing the entire input $1^{n'}$ from b_j to e_j; i.e., it enters some memory state twice in some configurations (q_j, d_j) and $(q_j, d_j + \ell_j)$ for $\ell_j \neq 0$. To avoid any ambiguity, we take the first loop the machine gets into, i.e., the first pair of configurations having the same memory state, along each input traversal. Observe that

$$\begin{aligned} \ell_j &\leq c^{s(n')} \quad \text{for each } j = 1, 2, \ldots, B, \\ B &\leq i(n'), \end{aligned} \tag{3.7}$$

since M is simultaneously $s(n)$ space and $i(n)$ input head reversals bounded.

Now, define

$$\ell = \prod_{j=1}^{B} \ell_j \,. \tag{3.8}$$

It is not too hard to see that the machine M must also accept the inputs $1^{n'+\mu\cdot\ell}$ for each $\mu \in \mathbf{N}$. In fact, we can replace the accepting computation path $\mathcal{C}$ for the input $1^{n'}$ by a new computation path $\mathcal{C}_\mu$ that visits the end markers in the same sequence of configurations $r_1, r_2, \ldots, r_A$. The path $\mathcal{C}_\mu$ is obtained from $\mathcal{C}$ by iterating, $\frac{\mu\cdot\ell}{\ell_j}$ more times, the loop of length ℓ_j in the segment of computation connecting b_j and e_j, for each $j = 1, 2, \ldots, B$. Note that ℓ is a common multiple of $\ell_1, \ell_2, \ldots, \ell_B$ and hence $\frac{\mu\cdot\ell}{\ell_j} \in \mathbf{N}$; i.e., the new segments of computation traverse exactly $n' + \frac{\mu\cdot\ell}{\ell_j} \cdot \ell_j = n' + \mu\cdot\ell$ positions, beginning and ending in the same configurations b_j and e_j, respectively, for each $j = 1, 2, \ldots, B$. The segments between e_j and b_{j+1} (always returning back to the same end marker) are left unchanged. Hence, for each $\mu \in \mathbf{N}$, $\mathcal{C}_\mu$ is a valid accepting computation path on the input $1^{n'+\mu\cdot\ell}$.

To complete the proof, we are now going to show that $1^{n'+\mu'\cdot\ell} \notin \mathcal{L}_f$, where μ' is defined by

$$\mu' = \prod_{p \le p_i} p \,, \tag{3.9}$$

which is a contradiction. The product is taken over all primes not exceeding p_i; hence, $(n' + \mu'\cdot\ell) \bmod p = n' \bmod p$ for each prime $p \le p_i$. This gives that n' and $n' + \mu'\cdot\ell$ share the same "least prime nondivisor" p_i. Therefore, to show that $1^{n'+\mu'\cdot\ell} \notin \mathcal{L}_f$, there only remains to exhibit a prime power ${p_j}^k \le y$, with $p_j \le \sqrt{y}$, $j \ne i$, and $k \ge 1$, such that ${p_j}^k$ does not divide $n' + \mu'\cdot\ell$.

Since we have shown that $1^{n'} \in \mathcal{L}_f$, each such prime power divides n', and hence

$$(n' + \mu'\cdot\ell) \bmod {p_j}^k = (\mu'\cdot\ell) \bmod {p_j}^k \,. \tag{3.10}$$

Therefore, the problem reduces to exhibit ${p_j}^k$ not dividing $\mu'\cdot\ell$.

To this aim, observe that ℓ, as defined by (3.8), is bounded by

$$\ell = \prod_{j=1}^{B} \ell_j \le \prod_{j=1}^{B} c^{s(n')} = \left(c^{s(n')}\right)^B \le c^{s(n')\cdot i(n')} < \sqrt{n'} \,, \tag{3.11}$$

using (3.7) and (3.6). On the other hand, the factorization of ℓ must be of the form

$$\ell = \nu \cdot \prod_{p \le \sqrt{y},\ p \ne p_i} p^{\alpha_p} \,, \tag{3.12}$$

where the product is taken over all primes not exceeding $\sqrt{y}$ and different from p_i, while ν contains the (possibly empty) part of the factorization consisting of powers of those primes p greater than $\sqrt{y}$ or $p = p_i$.

Let us now show that there exists a prime $p' \le \sqrt{y}$, $p' \ne p_i$, such that $\alpha_{p'} < \lfloor \log_{p'} y \rfloor - 1$. Suppose, by contradiction, that $\alpha_p \ge \lfloor \log_p y \rfloor - 1$ for each prime $p \le \sqrt{y}$, $p \ne p_i$. Then $\log_p y \ge 2$ and hence

$$\lfloor \log_p y \rfloor - 1 \ge \frac{1}{2} \cdot \lfloor \log_p y \rfloor \,.$$

This gives

$$\ell = \nu \cdot \prod_{p \le \sqrt{y},\ p \ne p_i} p^{\alpha_p} \ge \prod_{p \le \sqrt{y},\ p \ne p_i} p^{\lfloor \log_p y \rfloor - 1} \ge \prod_{p \le \sqrt{y},\ p \ne p_i} p^{\frac{1}{2} \cdot \lfloor \log_p y \rfloor} = \sqrt{n'}\,,$$

i.e., $\ell \ge \sqrt{n'}$, which contradicts (3.11). Hence, there exists a prime $p' \le \sqrt{y}$, $p' \ne p_i$ such that $\alpha_{p'} < \lfloor \log_{p'} y \rfloor - 1$.

By (3.12), this implies that ℓ is not an integer multiple of $p'^{\lfloor \log_{p'} y \rfloor - 1}$ and therefore, by (3.9), $\mu' \cdot \ell$ is not an integer multiple of $p'^{\lfloor \log_{p'} y \rfloor}$. But then, we have a prime power $p'^{\lfloor \log_{p'} y \rfloor} \le y$, with $p' \le \sqrt{y}$ and $p' \ne p_i$, that does not divide $n' + \mu' \cdot \ell$, using (3.10). Therefore, $1^{n' + \mu' \cdot \ell} \notin \mathcal{L}_f$, which is a contradiction and completes the proof of the theorem. □

4. Separation results in the unary case. We now draw some important structural consequences from the previous results. Note that the alternating algorithm provided for $\mathcal{L}_f$ in Theorem 3.1 consists of a deterministic phase followed by a single universal branching. Hence, it can be run on a Π_1 machine. Moreover, a *middle* space×input head reversals bounded machine can be viewed as an *accept* device, which in turn is a special case of a *weak* machine. Thus we have the following corollary.

COROLLARY 4.1. *Let $\mathcal{L}_f$ be a unary language defined as above for any unbounded monotone increasing recursive function $f(n)$. Then $\mathcal{L}_f \in$ middle-Π_1SPACE×REVERSALS$^{1^*}(\log\log n \cdot f(n))$. Hence, for $c \in \{middle, accept, weak\}$, $\mathcal{L}_f \in c$-Π_1SPACE×REVERSALS$^{1^*}(\log\log n \cdot f(n))$.*

On the other hand, from Theorem 3.2 we get Corollary 4.2.

COROLLARY 4.2. *Let $\mathcal{L}_f$ be a unary language defined as above. Then $\mathcal{L}_f \notin$ weak-Σ_1SPACE × REVERSALS$(h(n))$ for any $h(n) \in o(\log n)$. Hence, for $c \in \{middle, accept, weak\}$ and any $h(n) \in o(\log n)$, $\mathcal{L}_f \notin c$-Σ_1SPACE × REVERSALS $(h(n))$.*

Combining the above two corollaries, we have the first separation for "well-behaved" functions $h(n)$ between $\log\log n$ and $\log n$ in Theorem 4.3.

THEOREM 4.3. *Let $h(n) \in o(\log n)$ be any recursive function such that $\frac{h(n)}{\log\log n}$ is unbounded and monotone increasing. Then, for $c \in \{middle, accept, weak\}$,*

$$c\text{-}\Pi_1\text{SPACE}\times\text{REVERSALS}^{1^*}(h(n)) \ne c\text{-}\Sigma_1\text{SPACE}\times\text{REVERSALS}^{1^*}(h(n))\,.$$

Proof. Clearly, if $h(n)$ is recursive, then so is $f(n) = \frac{h(n)}{\log\log n}$. Then, by Corollary 4.1, $\mathcal{L}_f$ belongs to c-Π_1SPACE × REVERSALS$^{1^*}(h(n))$ for each $c \in \{middle, accept, weak\}$. On the other hand, by Corollary 4.2, $\mathcal{L}_f$ is not in c-Σ_1SPACE × REVERSALS$^{1^*}(h(n))$, since $h(n) \in o(\log n)$. □

For further separations we need the fact that the class of regular languages is closed under complementation and that regular languages are accepted by one-way Turing machines working in constant space. Hence, from Theorem 3.2, we get Corollary 4.4.

COROLLARY 4.4. *Let $\mathcal{L}_f$ be a unary language defined as above. Then neither $\mathcal{L}_f$ nor $\mathcal{L}_f{}^{\mathrm{c}}$ are regular.*

As recalled in the Introduction, apart from *middle* alternation, for any other combination of determinism, nondeterminism, or alternation with the space notions defined in section 2, computational lower bounds for nonregular unary and general languages coincide and are optimal [4]. Thus, for *accept* alternating machines, Table 1.1 can be shrunk as shown in Table 4.1.

TABLE 4.1
Optimal lower bounds on $s(n) \cdot i(n)$ for accept *alternating and nondeterministic (Σ_1) Turing machines recognizing nonregular languages* [4].

	Unary and general languages
One-way alternating	$\log \log n$
One-way nondeterministic (Σ_1)	$\log n$
Two-way alternating	$\log \log n$
Two-way nondeterministic (Σ_1)	$\log n$

Now, from Table 4.1 (for proof, see [4, Thm. 6]), we get the following corollary.

COROLLARY 4.5. *Let L be a nonregular language. Then $L \notin$ accept-Σ_1SPACE$\times$ REVERSALS$(h(n))$ for any $h(n) \in o(\log n)$. Hence, for $c \in \{middle, accept\}$ and any $h(n) \in o(\log n)$, $L \notin c$-Σ_1SPACE$\times$REVERSALS$(h(n))$.*

This allows us to present some further separations for *middle* and *accept* complexity classes.

THEOREM 4.6. *Let $h(n) \in o(\log n)$ be any recursive function such that $\frac{h(n)}{\log \log n}$ is unbounded and monotone increasing. Then, for $c \in \{middle, accept\}$,*

$$c\text{-}\Pi_1\text{SPACE}\times\text{REVERSALS}^{1^*}(h(n)) \neq co\text{-}c\text{-}\Sigma_1\text{SPACE}\times\text{REVERSALS}^{1^*}(h(n))\,,$$
$$co\text{-}c\text{-}\Pi_1\text{SPACE}\times\text{REVERSALS}^{1^*}(h(n)) \neq c\text{-}\Sigma_1\text{SPACE}\times\text{REVERSALS}^{1^*}(h(n))\,.$$

Proof. Note that, by Corollary 4.5, the classes c-Σ_1SPACE$\times$REVERSALS$^{1^*}(h(n))$ and co-c-Σ_1SPACE$\times$REVERSALS$^{1^*}(h(n))$ coincide with the class of regular languages. On the other hand, for $f(n) = \frac{h(n)}{\log \log n}$, neither $\mathcal{L}_f$ nor $\mathcal{L}_f{}^{\mathrm{c}}$ are regular, by Corollary 4.4, and they belong, respectively, to c-Π_1SPACE$\times$REVERSALS$^{1^*}(h(n))$ and to co-c-Π_1SPACE$\times$REVERSALS$^{1^*}(h(n))$, by Corollary 4.1. □

Theorem 4.6 shows, for *middle* or *accept* SPACE$\times$REVERSALS complexity measure, that the class of the languages accepted by machines making only universal decisions does not coincide with the class of the complements of languages recognizable by machines making only existential decisions.

At this point, it is quite natural to investigate whether or not this separation can be extended even to the *weak* case. The argument of Theorem 4.6 cannot be applied here, since *weak*-Σ_1SPACE$\times$REVERSALS$(\log \log n)$ does contain unary nonregular languages [4]. We conjecture that even $\mathcal{L}_f{}^{\mathrm{c}}$ does not belong to *weak*-Σ_1SPACE$\times$ REVERSALS$^{1^*}(h(n))$ for a suitable function $h(n) \in o(\log n)$. Nevertheless, the "pumping" argument used to prove Theorem 3.2 does not work in the case of $\mathcal{L}_f{}^{\mathrm{c}}$. So, we leave the separation of *weak*-Π_1SPACE$\times$REVERSALS$^{1^*}(h(n))$ from *co-weak*-Σ_1SPACE$\times$REVERSALS$^{1^*}(h(n))$ as an open problem. However, in the next section, we will show how to solve this problem by using witness languages defined over a binary alphabet.

Finally, the separation of the higher levels of the alternating hierarchy from determinism can be obtained by recalling the following result in [4, Thm. 6].

THEOREM 4.7. *Let M be a* weak *deterministic (or nondeterministic) Turing machine recognizing a nonregular language within $s(n)$ space and $i(n)$ input head reversals. Then $s(n) \cdot i(n) \notin o(\log n)$ (or $s(n) \cdot i(n) \notin o(\log \log n)$, respectively). These bounds are optimal for both the unary and general cases.*

Using this result, we can easily get Theorem 4.8.

THEOREM 4.8. *Let $h(n)$ be a function satisfying $h(n) \in o(\log n) \cap \Omega(\log \log n)$.*

Then

weak-DSPACE$\times$REVERSALS$^{1^*}(h(n))$ *is properly included in*
weak-Σ_1SPACE$\times$REVERSALS$^{1^*}(h(n))$ *and in*
co-weak-Σ_1SPACE$\times$REVERSALS$^{1^*}(h(n))$.

If, moreover, $h(n)$ *is recursive, with* $\frac{h(n)}{\log\log n}$ *unbounded and monotone increasing, then*

weak-DSPACE$\times$REVERSALS$^{1^*}(h(n))$ *is properly included even in*
weak-Π_1SPACE$\times$REVERSALS$^{1^*}(h(n))$ *and in*
co-weak-Π_1SPACE$\times$REVERSALS$^{1^*}(h(n))$.

Proof. Just note that, by Theorem 4.7, *weak*-DSPACE $\times$ REVERSALS$(h(n))$ coincides with the class of regular languages for each $h(n) \in o(\log n)$. On the other hand, again by Theorem 4.7, *weak*-Σ_1SPACE $\times$ REVERSALS$^{1^*}(\log\log n)$ contains a nonregular unary language. Moreover, by Corollary 4.1, even *weak*-Π_1SPACE $\times$ REVERSALS$^{1^*}(h(n))$ contains unary nonregular languages whenever $h(n)$ satisfies the conditions of the theorem.

The remaining proper inclusions follow by observing that the class of nonregular languages is closed under complement. □

5. Separation results for general languages. In this section, we study separations in the case of languages defined over alphabets of at least two symbols. In particular, we show that all separations in section 4, proved in the unary case for "well-behaved" $h(n)$ between $\log\log n$ and $\log n$, hold in the general case for *any function* $h(n) \in o(\log n) \cap \Omega(\log\log n)$ and in particular for $\log\log n$.

Further, we get that, for general languages, *weak*-Π_1SPACE$\times$REVERSALS$(h(n))$ is separated from *co-weak*-Σ_1SPACE$\times$REVERSALS$(h(n))$. Symmetrically, we have *weak*-Σ_1SPACE$\times$REVERSALS$(h(n)) \neq$ *co-weak*-Π_1SPACE$\times$REVERSALS$(h(n))$ for any function $h(n) \in o(\log n) \cap \Omega(\log\log n)$.

In order to state these results, we consider the language $\mathcal{L}_1$ defined by

$$\mathcal{L}_1 = \{a^k b^{k+m} : m > 0 \text{ is a common multiple of all } r \leq k\}.$$

THEOREM 5.1. *Let* $\mathcal{L}_1$ *be the language defined as above. Then* $\mathcal{L}_1$ *can be accepted by a one-way* Π_1 *machine in* middle $\mathcal{O}(\log\log n)$ *space.*

Proof. The proof is similar to that of Theorem 3.2 in [22]. On input $x = a^k b^t$ of length $n = t + k$, the Π_1 machine M accepting $\mathcal{L}_1$ first counts, on the work tape, the number k of a's at the beginning of the input. Next, it moves the input head k positions to the right, rejecting if the right end marker is reached, i.e., if $t \leq k$. Finally, M universally generates all integers r less than or equal to k, and, by counting the length of the remaining part of the input modulo r, it checks whether each r divides $t - k$.

Clearly, M is a one-way machine. The space used by M on the input $x = a^k b^t$ is proportional to the space needed to represent k in binary notation, i.e., $\mathcal{O}(\log k)$. Moreover, if $x \in \mathcal{L}_1$, then $t - k$ is a common multiple of all $r \leq k$. Since the least common multiple of $\{1, 2, \ldots, k\}$ is bounded from below by d^k, for some constant $d > 1$ (for proof, see [23, Lem. 4.1.2]), we get $k \leq \log_d(t - k) \leq \log_d(t + k) = \log_d n$. Hence, $s(n) \in \mathcal{O}(\log\log n)$. □

COROLLARY 5.2. $\mathcal{L}_1 \in c$-Π_1SPACE$\times$REVERSALS$(\log\log n)$ *for* $c \in \{$*middle*, *accept*, *weak*$\}$.

We now state a lower bound on the product space$\times$input head reversals for *weak* nondeterministic Turing machines accepting $\mathcal{L}_1$.

THEOREM 5.3. *Let $\mathcal{L}_1$ be the language defined as above. Then, for any $h(n) \in o(\log n)$, $\mathcal{L}_1$ is not in weak-Σ_1SPACE$\times$REVERSALS$(h(n))$.*

Proof. Suppose, by contradiction, that $\mathcal{L}_1$ can be accepted by some nondeterministic machine in *weak* $s(n)$ space and $i(n)$ reversals, with $s(n) \cdot i(n) = h(n) \in o(\log n)$. Then $\mathcal{L}_1$ can even be accepted by a *one-way* nondeterministic machine M in *weak* $h(n)$ space. (The simulating machine nondeterministically guesses, at each input tape position, the crossing sequence of memory states that corresponds to an accepting computation path and checks the compatibility of the neighboring crossing sequences. Since the original *weak* machine is simultaneously $s(n)$ space and $i(n)$ input head reversals bounded, then *weak* $\mathcal{O}(s(n) \cdot i(n))$ space suffices for processing the input from left to right. For details, see [4, Lem. 5].)

The number of different memory states of M using no more than $h(n)$ work tape cells can be bounded by $c^{h(n)}$ for a suitable constant $c > 1$. Since $h(n) \in o(\log n)$, there exists $n_0 \in \mathbf{N}$ (cf. (3.5) in Theorem 3.2), such that

$$c^{h(n)} < \sqrt{n} \quad \text{for each } n \geq n_0\,. \tag{5.1}$$

Now, consider an input $x = a^k b^{k+m}$ of length $n = 2k + m \geq n_0$, such that m is the least common multiple of all $r \leq k$. Clearly, the string $x = a^k b^{k+m}$ belongs to $\mathcal{L}_1$. In addition, we choose $k > 0$ sufficiently large, so that $2k \leq d^k \leq m$. Here $d > 1$ is the constant used in the proof of Theorem 5.1; i.e., d^k is a lower bound for m, the least common multiple of $\{1, 2, \ldots, k\}$. Hence, b^m is a "dominant" portion of the input; i.e., we have

$$m \geq \frac{n}{2} > \sqrt{n}\,.$$

On input $x = a^k b^{k+m}$, M must have an accepting computation path $\mathcal{C}$ that uses at most $h(n)$ space. Since $m > \sqrt{n}$ and, by (5.1), the number of different memory states is bounded by $\sqrt{n}$, M must enter some memory state twice while traversing the segment b^m on the input. That is, $\mathcal{C}$ enters some configurations (q, d) and $(q, d+\ell)$, with $2k \leq d < d+\ell \leq n$.

But then M must also accept the input $a^k b^{k+m-\ell}$ by means of the accepting computation path $\mathcal{C}'$ obtained from $\mathcal{C}$ by "skipping" the segment between the configurations (q, d) and $(q, d+\ell)$. Since m is the *least* common multiple of all $r \leq k$ and $\ell > 0$, it turns out that $m - \ell$ is not a multiple of all $r \leq k$. Hence, M accepts $a^k b^{k+m-\ell} \notin \mathcal{L}_1$, which is a contradiction. □

Note that, as a consequence of Theorem 5.3, the language $\mathcal{L}_1$ is not regular. Using Corollary 5.2 and Theorem 5.3, we are now able to rewrite Theorem 4.3 for languages defined over general alphabets.

COROLLARY 5.4. *For each $h(n) \in o(\log n) \cap \Omega(\log \log n)$ and $c \in \{$middle, accept, weak$\}$,*

$$c\text{-}\Pi_1\text{SPACE}\times\text{REVERSALS}(h(n)) \neq c\text{-}\Sigma_1\text{SPACE}\times\text{REVERSALS}(h(n))\,.$$

Even Theorem 4.8 can easily be rewritten in the case of general alphabets as follows.

COROLLARY 5.5. *Let $h(n)$ be a function satisfying $h(n) \in o(\log n) \cap \Omega(\log \log n)$. Then weak-*DSPACE$\times$REVERSALS$(h(n))$ *is properly included in the following classes:*

weak-Σ_1SPACE$\times$REVERSALS$(h(n))$,

co-weak-Σ_1SPACE$\times$REVERSALS$(h(n))$,

weak-Π_1SPACE$\times$REVERSALS$(h(n))$,
co-weak-Π_1SPACE$\times$REVERSALS$(h(n))$.

Let us now compare the classes defined by machines making universal decisions with the classes of complements of languages accepted by machines making existential decisions. For sublogarithmic bounds in the unary case, these classes have been separated for the *middle* and *accept* modes (Theorem 4.6), while the separation is an open problem for the *weak* mode.

Here we prove this separation using witness languages defined over alphabets containing at least two symbols. To this purpose, we extend the lower bound of Theorem 5.3 to the language ${\mathcal{L}_1}^{\mathrm{c}}$. First, consider the following language:

$$\mathcal{L}_2 = \{a^k b^t : t \leq k \text{ or } \exists r \leq k \text{ which does not divide } t-k\}\,.$$

Note that $\mathcal{L}_2 = {\mathcal{L}_1}^{\mathrm{c}} \cap \{a\}^*\{b\}^*$.

THEOREM 5.6. *Let $\mathcal{L}_2$ be the language defined as above. Then, for any $h(n) \in o(\log n)$, $\mathcal{L}_2$ is not in weak-Σ_1SPACE$\times$REVERSALS$(h(n))$.*

Proof. As in Theorem 5.3, suppose, by contradiction, that $\mathcal{L}_2$ can be accepted by some nondeterministic machine in *weak* $s(n)$ space and $i(n)$ input head reversals, with $s(n) \cdot i(n) = h(n) \in o(\log n)$. Then $\mathcal{L}_2$ can even be accepted by a *one-way* nondeterministic machine M in *weak* $h(n)$ space [4, Lem. 5]. The number of different memory states of M using at most $h(n)$ work tape cells is bounded by $c^{h(n)}$ for a suitable constant $c > 1$. Since $h(n) \in o(\log n)$, there exists $n_0 \in \mathbf{N}$, such that

$$c^{h(n)} < \frac{n}{2} \quad \text{for each } n \geq n_0\,. \tag{5.2}$$

Now, for some even $n \geq n_0$, consider the string $x = a^{\frac{n}{2}} b^{\frac{n}{2}}$, easily seen to be in $\mathcal{L}_2$. Hence, M must have an accepting computation path on the input x, consisting of memory states that use at most space $h(n)$. Since, by (5.2), the number of different memory states is bounded by $\frac{n}{2}$, this computation path enters a *loop* while traversing the suffix $b^{\frac{n}{2}}$. More precisely, M enters some memory state twice, in some configurations (q, d) and $(q, d+\ell)$, with $d > \frac{n}{2}$ and $\ell > 0$. Thus, it is not hard to see that M must also accept the strings of the form

$$a^{\frac{n}{2}} b^{\frac{n}{2}+\mu\cdot\ell} \in \mathcal{L}_2 \quad \text{for each } \mu \in \mathbf{N}\,. \tag{5.3}$$

Now, let $\mu' = \left(\frac{n}{2}\right)!$. It is easy to see that $\mu' \cdot \ell > 0$ and that each integer $r \leq \frac{n}{2}$ divides $\mu' \cdot \ell$. That is, the string $a^{\frac{n}{2}} b^{\frac{n}{2}+\mu'\cdot\ell}$ does not belong to $\mathcal{L}_2$, which contradicts (5.3). □

Using Theorem 5.6, we are now able to show Corollary 5.7.

COROLLARY 5.7. *Let $\mathcal{L}_1$ be the language defined as above. Then the complement of $\mathcal{L}_1$ is not in weak-Σ_1SPACE$\times$REVERSALS$(h(n))$ for any $h(n) \in o(\log n)$.*

Proof. Should ${\mathcal{L}_1}^{\mathrm{c}}$ be in *weak*-Σ_1SPACE$\times$REVERSALS$(h(n))$, for some $h(n) \in o(\log n)$, then so would $\mathcal{L}_2 = {\mathcal{L}_1}^{\mathrm{c}} \cap \{a\}^*\{b\}^*$, which contradicts Theorem 5.6. □

Finally, by combining Corollaries 5.2 and 5.7, we get Theorem 5.8.

THEOREM 5.8. *Let $h(n)$ be a function satisfying $h(n) \in o(\log n) \cap \Omega(\log\log n)$. Then, for $c \in \{middle, accept, weak\}$,*

$$c\text{-}\Pi_1\text{SPACE}\times\text{REVERSALS}(h(n)) \neq co\text{-}c\text{-}\Sigma_1\text{SPACE}\times\text{REVERSALS}(h(n))\,,$$
$$c\text{-}\Sigma_1\text{SPACE}\times\text{REVERSALS}(h(n)) \neq co\text{-}c\text{-}\Pi_1\text{SPACE}\times\text{REVERSALS}(h(n))\,.$$

Acknowledgments. The authors wish to thank an anonymous referee for helpful comments and remarks.

REFERENCES

[1] M. Alberts, *Space complexity of alternating Turing machines*, in Fundamentals of Computation Theory, Proceedings, Lecture Notes in Computer Science 199, Springer-Verlag, Berlin, New York, 1985, pp. 1–7.

[2] B. von Braunmühl, R. Gengler, and R. Rettinger. *The alternation hierarchy for machines with sublogarithmic space is infinite*, in Symposium on Theoretical Aspects of Computer Science 1994, Proceedings, Lecture Notes in Computer Science 775, Springer-Verlag, Berlin, New York, 1994, pp. 85–96.

[3] A. Bertoni, C. Mereghetti, and G. Pighizzini, *An optimal lower bound for nonregular languages*, Inform. Process. Lett., 50 (1994), pp. 289–292; *Corrigendum*, 52 (1994), p. 339.

[4] A. Bertoni, C. Mereghetti, and G. Pighizzini, *Strong optimal lower bounds for Turing machines that accept nonregular languages*, in Mathematical Foundations of Computer Science 1995, Proceedings, Lecture Notes in Computer Science 969, Springer-Verlag, Berlin, New York, 1995, pp. 309–318.

[5] A. Chandra, D. Kozen, and L. Stockmeyer, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114–133.

[6] V. Geffert, *A hierarchy that does not collapse: Alternations in low level space*, RAIRO Inform. Théor. Appl., 28 (1994), pp. 465–512.

[7] V. Geffert, *Bridging across the* $\log(n)$ *space frontier*, in Mathematical Foundations of Computer Science 1995, Proceedings, Lecture Notes in Computer Science 969, Springer-Verlag, Berlin, New York, 1995, pp. 50–65.

[8] G. Hardy and E. Wright. *An Introduction to the Theory of Numbers*, 5th ed., Oxford University Press, 1979.

[9] J.-W. Hong, *A tradeoff theorem for space and reversal*, Theoret. Comput. Sci., 32 (1984), pp. 221–224.

[10] J. Hopcroft and J. Ullman, *Some results on tape–bounded Turing machines*, J. Assoc. Comput. Mach., 16 (1969), pp. 168–177.

[11] J. Hromkovič, B. Rovan, and A. Slobodová, *Deterministic versus nondeterministic space in terms of synchronized alternating machines*, Theoret. Comput. Sci., 132 (1994), pp. 319–336.

[12] N. Immerman, *Nondeterministic space is closed under complement*, SIAM J. Comput., 17 (1988), pp. 935–938.

[13] K. Iwama, $ASPACE(o(\log\log n))$ *is regular*, SIAM J. Comput., 22 (1993), pp. 136–146.

[14] P. Lewis, R. Stearns, and J. Hartmanis, *Memory bounds for the recognition of context free and context sensitive languages*, in IEEE Conference Record on Switching Circuit Theory and Logical Design, 1965, pp. 191–202.

[15] M. Liśkiewicz and R. Reischuk, *The sublogarithmic alternating space world*, SIAM J. Comput., 25 (1996), pp. 828–861.

[16] C. Mereghetti and G. Pighizzini, *A remark on middle space bounded alternating Turing machines*, Inform. Process. Lett., 56 (1995), pp. 229–232.

[17] W. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.

[18] A. Slobodová, *On the power of one–way globally deterministic synchronized alternating Turing machines and multihead automata*, Internat. J. Found. Comput. Sci., 6 (1995), pp. 431–446,

[19] R. Stearns, J. Hartmanis, and P. Lewis, *Hierarchies of memory limited computations*, in IEEE Conference Record on Switching Circuit Theory and Logical Design, 1965, pp. 179–190.

[20] I. Sudborough, *Efficient algorithms for path system problems and applications to alternating and time–space complexity classes*, in 21st IEEE Symposium on Foundations of Computer Science, Proceedings, 1980, pp. 62–73.

[21] R. Szelepcsényi, *The method of forced enumeration for nondeterministic automata*, Acta Inform., 26 (1988), pp. 279–284.

[22] A. Szepietowski, *Remarks on languages acceptable in* $\log\log n$ *space*, Inform. Process. Lett., 27 (1988), pp. 201–203.

[23] A. Szepietowski, *Turing Machines with Sublogarithmic Space*, Lecture Notes in Computer Science 843, Springer-Verlag, Berlin, New York, 1994.

SIAM J. COMPUT.
Vol. 28, No. 1, pp. 341–381

SEPARATOR-BASED SPARSIFICATION II: EDGE AND VERTEX CONNECTIVITY*

DAVID EPPSTEIN†, ZVI GALIL‡, GIUSEPPE F. ITALIANO§, AND THOMAS H. SPENCER¶

Abstract. We consider the problem of maintaining a dynamic planar graph subject to edge insertions and edge deletions that preserve planarity but that can change the embedding. We describe algorithms and data structures for maintaining information about 2- and 3-vertex-connectivity, and 3- and 4-edge-connectivity in a planar graph in $O(n^{1/2})$ amortized time per insertion, deletion, or connectivity query. All of the data structures handle insertions that keep the graph planar without regard to any particular embedding of the graph. Our algorithms are based on a new type of sparsification combined with several properties of separators in planar graphs.

Key words. analysis of algorithms, dynamic data structures, edge connectivity, vertex connectivity, planar graphs

AMS subject classifications. 68P05, 68Q20, 68R10

PII. S0097539794269072

1. Introduction.

Sparse certificates, small graphs that retain some property of a larger graph, appear often in graph theory, especially in problems of edge and vertex connectivity [2, 13, 31, 35]. The main motivation for studying sparse certificates lies in the fact that they are effective tools for speeding up many graph algorithms. To check whether a graph G has a given property $\mathcal{P}$, one can first compute a sparse certificate $\mathcal{C}$ for property $\mathcal{P}$ and then run an algorithm for $\mathcal{P}$ on the certificate rather than on G itself. This is favorable whenever computing certificates is faster than checking property $\mathcal{P}$. This method has led to improved algorithms for testing k-edge- and k-vertex-connectivity sequentially [16, 31, 35] and in parallel [2], for finding three independent spanning trees [1], and for reliability in distributed networks [26]. With the *sparsification* technique [8], sparse certificates additionally became an important tool for speeding up dynamic graph algorithms, in which edges may be inserted into and deleted from a graph while some graph property must be maintained throughout the sequence of modifications.

*Received by the editors June 6, 1994; accepted for publication (in revised form) October 30, 1996; published electronically June 15, 1998. Portions of this paper were presented at the 25th Annual ACM Symp. on Theory of Computing, San Diego, CA, 1993 [10].

http://www.siam.org/journals/sicomp/28-1/26907.html

†Department of Information and Computer Science, University of California, Irvine, CA 92697-3425 (eppstein@ics.uci.edu, http://www.ics.uci.edu/∼eppstein/). The research of this author was supported in part by NSF grant CCR-9258355 and by matching funds from Xerox Corp.

‡Computer Science Department, Columbia University, New York, NY 10027 (galil@cs.columbia.edu) and Computer Science Department, Tel-Aviv University, Tel-Aviv, Israel. The research of this author was supported in part by NSF grant CCR-90-14605 and CISE Institutional Infrastructure grant CCR-90-24735.

§Dipartimento di Matematica Applicata e Informatica, Università "Ca' Foscari," Venice, Italy (italiano@dsi.unive.it, http://www.dsi.unive.it/∼italiano/). The research of this author was supported in part by EU ESPRIT Long Term Research Project ALCOM-IT under contract no. 20244, by a Research Grant from University "Ca' Foscari" of Venice, and by the Italian MURST Project "Efficienza di Algoritmi e Progetto di Strutture Informative."

¶Department of Computer Science, University of Nebraska at Omaha, Omaha, NB 68182-0243 (spencer@unocss.unomaha.edu). Current address: 5740 S. 100th Plaza #2A, Omaha, NB 68127. This research was partially supported by the University Committee on Research, University of Nebraska at Omaha and by NSF grant CCR-9319772.

While sparsification has many applications in algorithms for general graphs, it seemed unlikely that it could be used to speed up algorithms for special families of graphs that are already sparse, such as planar graphs. However, algorithms for planar graphs are especially important, as these graphs arise frequently in applications. In the companion paper [11] we developed a new, general technique for dynamic planar graph problems, based upon the notion of *compressed certificates*, which have both fewer edges and fewer vertices than the original graph. We expanded the notion of certificate to a definition for graphs in which a subset of the vertices is denoted as *interesting*; these *compressed certificates* may reduce the size of the graph by removing uninteresting vertices. Note that this is a generalization of the previous certificates, as compressed certificates reduce to sparse certificates in the special case where all the vertices are interesting. Using the notion of compressed certificates, we defined a type of *separator-based sparsification* based on *separators*, small sets of vertices the removal of which splits the graph into roughly equal-size components. We then applied separator-based sparsification to maintain information about the minimum spanning forest, connectivity, and 2-edge-connectivity of a planar graph. We further showed how to maintain a graph subject to *arbitrary* edge insertions and deletions, with queries that test whether the graph is currently planar or whether a potential new edge would violate planarity.

In this paper we extend these ideas in several ways. Our first constribution is to adapt separator-based sparsification from the companion paper [11] to work on more general certificates and properties. Namely, we extend the notion of compressed certificates to properties that can be defined with respect to a particular pair of vertices rather than on the whole graph. We refer to these as *local* certificates as opposed to *global* certificates. The most general notion of certificate is a *full* certificate, which is at the same time a local and global certificate.

Our second contribution is to prove a number of structural properties of certificates for edge connectivity in general graphs. Among these properties, we give necessary and sufficient conditions for a graph to be a local or global certificate of k-edge-connectivity for any k. This characterization is not only a powerful algorithmic tool, as we show in this paper, but also contributes a new insight into the structural properties of edge connectivity and improves our understanding of certificates. We believe that these structural properties may be of independent interest and find applications to other graph-theoretical areas.

Thirdly, as a first application of our compressed certificates, we use them to develop dynamic planar graph algorithms. We maintain information about 3- and 4-edge-, and 2- and 3-vertex-connectivity in a planar graph during an intermixed sequence of edge deletions, edge insertions that keep the graph planar, and connectivity queries in $O(n^{1/2})$ amortized time per operation. All our algorithms improve previous bounds: for 2- and 3-vertex- and 3-edge-connectivity, the best previous time bound was $O(n^{2/3})$ amortized [8, 19, 21], while for 4-edge-connectivity nothing better than testing the graph from scratch after each update was known. These bounds apply to problems in which insertions need not respect a fixed embedding of the graph; a number of other papers have worked on dynamic graph problems such as minimum spanning forests, connectivity, and planarity testing for graphs with a fixed planar embedding [12, 14, 15, 18, 19, 22, 21, 24, 32, 33].

Finally, our methods apply to static as well as dynamic graph problems. A general certificate construction method from our companion paper, together with the certificates defined here, gives a unified method of testing 3- and 4-edge-, and 2- and

3-vertex-connectivity in planar graphs, in linear time. In recent work, Eppstein [7] has shown how to compute k-edge- or k-vertex-connectivity in planar graphs in linear time for any constant k.

The remainder of this paper consists of the following sections. Section 2 contains basic definitions. In section 3 we recall some properties of separator-based sparsification and compressed certificates from reference [11]. In section 4 we prove the properties and describe the tools we will be using for our certificates for edge connectivity. In section 5 compressed certificates for edge connectivity are developed, and our bounds for 3- and 4-edge-connectivity are proved. In section 6 sparsification is applied to fully dynamic 2- and 3-vertex-connectivity by using compressed certificates already available in the literature. Finally, in section 7 we list some open problems and concluding remarks.

2. Preliminaries. In this section we introduce the notions of vertex and edge connectivity of a graph. Next, we review a generalized tree, due to Dinitz, Karzanov, and Lomonosov [4], that describes an elegant decomposition of a graph using its connectivity edge-cuts.

2.1. Edge and vertex connectivity. Let $G = (V, E)$ be an undirected graph, possibly with parallel edges. Throughout the paper, we denote by m the number of edges and by n the number of vertices in G. Given an integer $k \geq 2$, a pair of vertices $\langle u, v\rangle$ is said to be *k-edge-connected* in G if the removal of any $(k-1)$ edges in G leaves u and v connected. It is well known that u and v are k-edge-connected if and only if there are k edge-disjoint paths between u and v. k-edge-connectivity is an equivalence relationship, and we denote it by $\equiv_k$; i.e., if a pair of vertices $\langle x, y\rangle$ is k-edge-connected, we write $x \equiv_k y$. The vertices of a graph G are partitioned by this relationship into equivalence classes that we call *k-edge-connected classes*. Note that according to this definition a k-edge-connected class of G is a subset of vertices of G. G is said to be *k-edge-connected* if the removal of any $(k-1)$ edges leaves G connected. As a result of these definitions, G is k-edge-connected if and only if any two vertices of G are k-edge-connected. An edge set $E' \subseteq E$ is an *edge-cut for vertices x and y* if the removal of all the edges in E' disconnects G into two graphs, one containing x and the other containing y. An edge-set $E' \subseteq E$ is an *edge-cut for G* if the removal of all the edges in E' disconnects G into two graphs. An edge-cut E' for G (for x and y, respectively) is *minimal* if removing any edge from E' and re-inserting it back into G reconnects G (x and y, respectively). The cardinality of an edge-cut E', denoted by $|E'|$, is equal to the number of edges in E'. An edge-cut E' for G (for x and y, respectively) is said to be a *minimum edge-cut* or a *connectivity edge-cut*, if there is no other edge-cut E'' for G (for x and y, respectively) such that $|E''| < |E'|$. A connectivity edge-cut of cardinality 1 is called a *bridge*. The graphs left after deleting all the bridges of G are called the *2-edge-connected components* of G. Note the difference between 2-edge-connected classes and 2-edge-connected components: a 2-edge-connected class is a *subset of vertices* of G, while a 2-edge-connected component is a *subgraph* of G. However, 2-edge-connected classes and 2-edge-connected components are strictly related: indeed, a 2-edge-connected component is a subgraph of G induced by a 2-edge-connected class.

We now list some well-known properties that are an immediate consequence of the previous definitions and that will be used throughout this paper. First, any connectivity edge-cut must be minimal and must disconnect G exactly into two graphs. Furthermore, all the connectivity edge-cuts for G must have the same cardinality. Thus, the notion of *cardinality of the connectivity edge-cuts for G* is well defined: it

gives exactly the minimum number of edges whose deletion disconnects G. Similarly, given any two vertices x and y in G, all the connectivity edge-cuts for x and y have the same cardinality, and we speak about *the cardinality of the connectivity edge-cuts for x and y*. Given a graph G, the *edge connectivity of G* is defined as the cardinality of the connectivity edge-cuts for G, i.e., the minimum number of edges whose deletion disconnects G. We denote the edge connectivity of G by $\lambda(G)$: note that G is k-edge-connected if and only if $k \leq \lambda(G)$. Similarly, given any two vertices x and y in G, the *edge connectivity between x and y in G* is defined as the cardinality of the connectivity edge-cuts for x and y, i.e., the minimum number of edges whose deletion disconnects x and y in G. We denote the edge connectivity between x and y in G by $\lambda_{x,y}(G)$. As a consequence of this definition, $x \equiv_k y$ if and only if $k \leq \lambda_{x,y}(G)$.

Analogous definitions can be given for the case of vertex connectivity. A vertex set $V' \subset V$ (respectively, $V' \subseteq V - \{x, y\}$) is a *vertex-cut* for G (respectively, for vertices x and y) if the removal of all the vertices in V' disconnects G (respectively, x and y). The cardinality of a vertex-cut V', denoted by $|V'|$, is given by the number of vertices in V'. A vertex-cut V' for G (for x and y, respectively) is said to be a *minimum vertex-cut* or a *connectivity vertex-cut* if there is no other vertex-cut V'' for G (x and y, respectively) such that $|V''| < |V'|$. Two vertices x and y are k-vertex-connected if and only if a minimum vertex-cut for x and y contains at least k vertices. G is said to be *k-vertex-connected* if all its pairs of vertices are k-vertex-connected; equivalently, the minimum vertex-cut of G has cardinality k or more. A minimum vertex-cut of cardinality 1 is called an *articulation point*. Again, two vertices x and y in G are k-vertex-connected if and only if there are at least k vertex-disjoint paths between x and y.

2.2. The cactus tree. We now describe a tree-like decomposition of a k-edge-connected graph G into its $(k+1)$-edge-connected classes, which can be found in the beautiful work of Dinitz, Karzanov, and Lomonosov [4], and which we will be using throughout this paper. The generalized tree that describes this decomposition is called the *cactus tree*: note that it need not be a standard tree. We do not give many details here, referring the interested reader to references [4, 28]. The heart of this decomposition is the *Crossing Lemma* of [4], which can be informally stated as follows. Let G be a graph of edge connectivity λ: if any two λ-edge-cuts of G divide $V(G)$ into four (nonempty) parts, then shrinking these four parts produces a super-cycle having four super-nodes and exactly $\lambda/2$ parallel edges between any two neighbor super-nodes.

We mention some consequences of the Crossing Lemma, referring to [4, 28] for the full details and explanations. First of all, for λ odd, $\lambda/2$ is not an integer. Thus, according to the Crossing Lemma, for λ odd two different λ-edge-cuts of G cannot divide $V(G)$ into four nonempty parts. The Crossing Lemma also implies that there can be only $O(n)$ connectivity edge-cuts if λ is odd, and $O(n^2)$ of them if λ is even [4]. The set of all the connectivity edge-cuts of G can be compactly represented by a "tree-like" graph with weights on the edges, called the *cactus tree of G* and denoted by $\mathcal{T}(G)$, which can be constructed in $O(m + \lambda^2 n \log n)$ time [17]. Each vertex in G maps to exactly one node in $\mathcal{T}(G)$, so that any node of $\mathcal{T}(G)$ corresponds to a (possibly empty) subset of vertices from G. An edge-cut $(\mathcal{A}, \bar{\mathcal{A}})$ in $\mathcal{T}(G)$ corresponds to an edge-cut $(A, \bar{A})$ of G, where A consists of all the vertices of G that are mapped into nodes of $\mathcal{A}$. A λ-cut of $\mathcal{T}(G)$ is an edge-cut of $\mathcal{T}(G)$ of total weight λ. Each minimal edge-cut of $\mathcal{T}(G)$ (i.e., an edge-cut from which no edge can be removed) is also a λ-cut in $\mathcal{T}(G)$, and it corresponds to a connectivity edge-cut in G. Each connectivity edge-cut in G

corresponds to one or more λ-cuts in $\mathcal{T}(G)$. Thus, the minimal edge-cuts of $\mathcal{T}(G)$ compactly represent all the connectivity edge-cuts of G.

For λ odd, the cactus tree is particularly simple: it is actually a tree, all its edges are of weight λ, and any minimal edge-cut of $\mathcal{T}(G)$ is obtained by removing one of its edges. For λ even, we can have crossing connectivity edge-cuts, and $\mathcal{T}(G)$ is a tree of cycles. Namely, $\mathcal{T}(G)$ consists of cycles, such that any two cycles have at most one single node in common; thus, no edge of $\mathcal{T}(G)$ can be in more than one cycle. Every edge in a cycle is called a *cycle-edge* and has weight $\lambda/2$. Note that there can be cycles consisting only of two edges: we refer to these cycles as 2-*cycles*. Minimal edge-cuts of $\mathcal{T}(G)$ are obtained by removing any pair of cycle-edges that lies on the same cycle. Hence, a 2-cycle defines only one minimal edge-cut of $\mathcal{T}(G)$, while a cycle with $p \geq 3$ edges defines exactly $p(p-1)/2$ distinct minimal edge-cuts of $\mathcal{T}(G)$. The cactus tree of a graph is basically unique, up to the following convention: if λ is even, either three nodes can be in a triangle of cycle-edges, or they can all be joined to another new node with 2-cycles.

We now describe in more detail the cactus trees we will be using, namely, the cases $\lambda = 1, 2, 3$. The interested reader can find further details on cactus trees in references [4, 28]. If G has edge connectivity $\lambda = 1$, the cactus tree is actually a tree, called the *bridge-block tree of* G: its nodes correspond to the 2-edge-connected classes (and thus components) of G, and its edges to the bridges of G. For $\lambda = 2$, the cactus tree is a tree of cycles: indeed, if we shrink the 3-edge-connected classes of a 2-edge-connected graph, each biconnected component of the shrunken graph is a simple cycle. Even though the shrunken graph is not a tree, it can easily be represented as such (see Figure 1).

For $\lambda = 3$ the cactus tree is actually a tree having one node for each 4-edge-connected class and one edge for each 3-edge-cut. There might also be nodes in the cactus that correspond to no 4-edge-connected classes of G, as shown in Figure 2. We refer the interested reader to [4, 28] for a more detailed explanation about these nodes and only mention here that they are needed to keep the correspondence between minimal cuts of $\mathcal{T}(G)$ and connectivity edge-cuts of G.

3. Separator-based sparsification. *Sparsification* was introduced in [8] as a technique for designing fully dynamic graph algorithms, in which edges may be inserted into and deleted from a graph while some graph property must be maintained. This technique is based upon a suitable combination of graph decomposition and edge elimination and can be described as follows. Let G be a graph with m edges and n vertices: we partition the edges of G into a collection of sparse subgraphs (i.e., subgraphs with $O(n)$ edges) and summarize the relevant information for each subgraph in an even sparser *certificate*. Intuitively, a certificate for a given property is a smaller graph that retains the same property (we will give a more precise definition later). We merge certificates in pairs, producing larger subgraphs which we make sparse by again applying the certificate reduction. The result is a balanced binary tree in which each node is represented by a sparse certificate. Each edge insertion or deletion causes changes in $\log(m/n)$ tree nodes, but each such change occurs in a subgraph with $O(n)$ edges, reduced from the m edges in the original graph. This reduces a time bound of $T(m, n)$ to $O(T(O(n), n) \log(m/n))$. Using a more sophisticated approach (described in [9]), we can eliminate the logarithmic factor from this bound. This reduces the time bounds for many dynamic graph problems, including vertex and edge connectivity and minimum spanning forests, to exactly match the bounds known for sparse graphs.

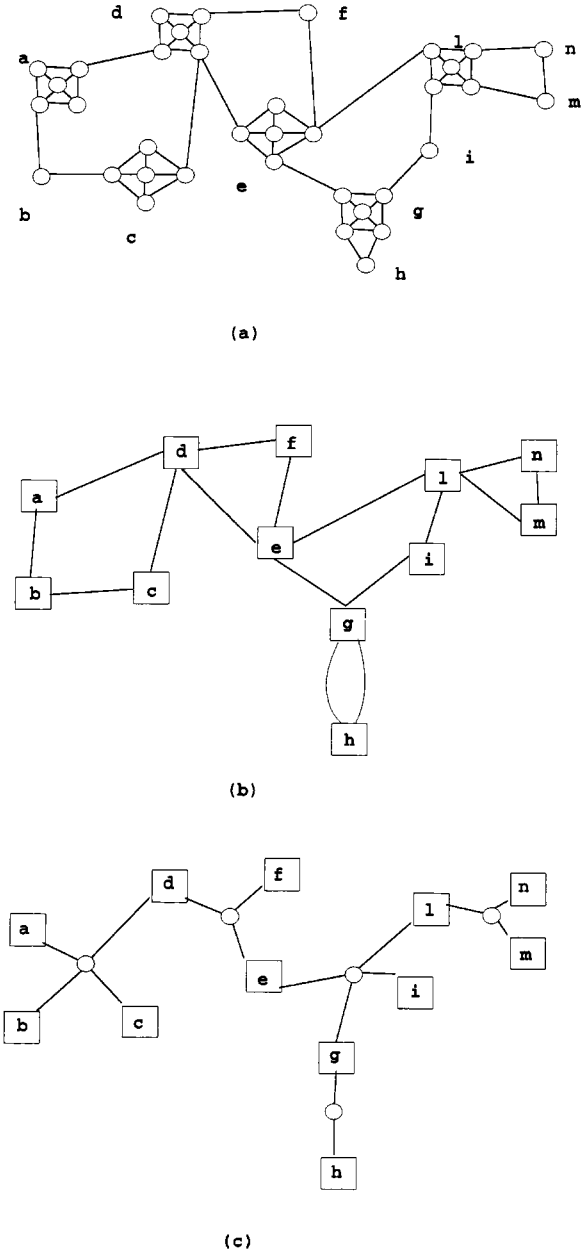


FIG. 1. (a) *A 2-edge-connected graph G;* (b) *the cactus tree of G;* (c) *the tree obtained by replacing each cycle of the cactus tree with a new vertex.*

In the companion paper [11], we developed a new, general technique for dynamic planar graph problems. In all these problems, we deal with either arbitrary or planarity-preserving insertions and therefore allow changes of the embedding. The new ideas behind this technique are the following. We expand the notion of a certificate to a definition for graphs in which a subset of the vertices is denoted as *interesting*; these *compressed certificates* may reduce the size of the graph by remov-

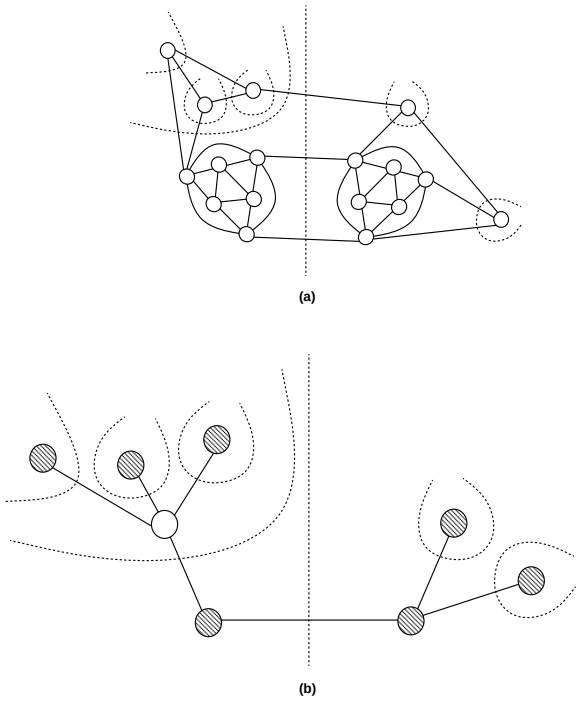


FIG. 2. *A 3-edge-connected graph G and its cactus tree. The 3-edge-cuts of G are shown as dashed lines. Nodes of the cactus not corresponding to 4-edge-connected classes of G are shown in white.*

ing uninteresting vertices. Using this notion of compressed certificates, we define a type of sparsification based on *separators*, small sets of vertices the removal of which splits the graph into roughly equal-size components. Recursively finding separators in these components gives a *separator tree* which we also use as our *sparsification tree*; the interesting vertices in each certificate will be those vertices used in separators at higher levels of the tree. We introduce the notion of a *balanced separator tree*, which also partitions the interesting vertices evenly in the tree. In [11], we show how to compute such a tree in linear time and how to maintain it dynamically. We call this technique *separator-based sparsification*.

Separator-based sparsification, as described in [11], applies to properties of the entire graph (such as planarity). However, there are some other properties that can be described either as a global graph property or in terms of pairs of vertices. More generally, let $\mathcal{P}$ a property of graphs: we say that $\mathcal{P}$ is *local* if it can be defined with respect to a particular pair (x, y) of vertices in the graph. A query related to property $\mathcal{P}$ is referred to as *global* if $\mathcal{P}$ is meant for the entire graph and is referred to as *local* if $\mathcal{P}$ is meant for a particular pair (x, y). Examples of local properties are edge and vertex connectivity, which are defined both for the entire graph (*global property*) and for any pair of vertices in the graph (*local property*). While maintaining a graph under edge insertions and deletions, at any time we might ask queries on whether the entire graph is k-vertex- (edge-) connected (*global k-connectivity query*), or rather, we might want to ask queries on whether any two given vertices are k-vertex- (edge-) connected (*local k-connectivity query*).

In this paper, we extend separator-based sparsification to work also on local properties. We first need the notion of *certificate* from [11], for which we adopt the new terminology of *global certificate*,

DEFINITION 3.1. *Let graph property $\mathcal{P}$ be fixed and let G be a graph with a set $X \subseteq V(G)$. A* global certificate for X in G *is a graph C, with $X \subseteq V(C)$, such that for any H with $V(G) \cap V(H) \subseteq X$, $V(C) \cap V(H) \subseteq X$, $G \cup H$ has property $\mathcal{P}$ if and only if $C \cup H$ has the property.*

The set X in Definition 3.1 represents the interesting vertices of G. According to this definition, a global certificate C captures the behavior of the entire graph G with respect to additions that only touch the interesting vertices. For instance, let $\mathcal{P}$ be k-edge-connectivity and let C be a global certificate for X in G: then, for any H such that $V(H) \cap V(G) \subseteq X$, $C \cup H$ is k-edge-connected if and only if $G \cup H$ is k-edge-connected. Note that when $X = V(G)$, this definition reduces to the one in [8].

For local properties, we need a slightly different notion of certificate, which we call a *local certificate*.

DEFINITION 3.2. *Let $\mathcal{P}$ be a local property of graphs and let G be a graph with a set $X \subseteq V(G)$. A graph C is a* local certificate *of $\mathcal{P}$ for X in G if and only if for any H with $V(H) \cap V(G) \subseteq X$, $V(C) \cap V(H) \subseteq X$, and any x and y in $V(H)$, $\mathcal{P}$ is true for (x, y) in $G \cup H$ if and only if it is true for (x, y) in $C \cup H$.*

Note that a local certificate C has to preserve the behavior of the property not only with respect to the interesting vertices in G, but also with respect to all vertices in H. For instance, let $\mathcal{P}$ be k-edge-connectivity, and let C be a local certificate for X in G: then, for any H such that $V(H) \cap V(G) \subseteq X$, and any $x, y \in V(H)$, $x \equiv_k y$ in $C \cup H$ if and only if $x \equiv_k y$ in $G \cup H$.

The two notions of global and local certificate can be combined together to yield a stronger notion of certificate.

DEFINITION 3.3. *Let graph property $\mathcal{P}$ be fixed and let G be a given graph with $X \subseteq V(G)$. A* full certificate for X in G *is graph C, which is both a global and a local certificate for X in G.*

For instance, let $\mathcal{P}$ be k-edge-connectivity and let C be a full certificate for X in G: then, for any H such that $V(H) \cap V(G) \subseteq X$, we have that

(i) $C \cup H$ is k-edge-connected if and only if $G \cup H$ is k-edge-connected; and
(ii) for any $x, y \in V(H)$, $x \equiv_k y$ in $C \cup H$ if and only if $x \equiv_k y$ in $G \cup H$.

Note that both conditions are needed, since neither (i) implies (ii), nor (ii) implies (i). As an example of full certificate, let $\mathcal{P}$ be connectivity. We partition the vertices of X into their connected components in G and replace each connected component by any spanning tree. We claim that this yields a full certificate for connectivity. Indeed, if two vertices in $G \cup H$ are connected by a path, then at each point the path switches between edges of G and edges of H, it will pass through a vertex $x \in X$, and the portion of the path in G can be replaced by a path through the spanning forest of the partition set containing x. Thus vertices are connected in $G \cup H$ if and only if they are connected in $C \cup H$ (i.e., C is a local certificate) and $G \cup H$ is connected if and only if $C \cup H$ is connected (i.e., C is a global certificate). This shows that C is indeed a full certificate.

We will use the term certificates to refer in general to certificates of all types (full, global, or local). We will use explicitly the terms full certificates, global certificates, and local certificates when we refer to these kinds of certificates only. Our method hinges upon the notion of *compressed* certificate.

DEFINITION 3.4. *Let graph property $\mathcal{P}$ be fixed and let G be a given graph with $X \subseteq V(G)$. A* compressed certificate for X in G *is a certificate C for X in G, such that $|C| = O(|X|)$.*

Some basic lemmas about global certificates were proved in the companion paper [11]. They can be easily extended to local and full certificates as well.

LEMMA 3.1 (see [11]). *Let C be a certificate for some set X in a given graph G and let C' be a certificate for X in C. Then C' is also a certificate for X in G.*

LEMMA 3.2 (see [11]). *Let C' be a certificate for X' in G' and let C'' be a certificate for X'' in G'', with $V(G') \cap V(G'') \subseteq X' \cap X''$. Then $C' \cup C''$ is a certificate for $X' \cup X''$ in $G' \cup G''$.*

The following lemma is an immediate consequence of the definition of certificate.

LEMMA 3.3. *Let C be a certificate for some set X in a given graph G. Then C is also a certificate for any set $X' \subset X$ in G.*

Proof. Fix any $X' \subset X$. Let H be given, with $V(H) \cap V(G) \subseteq X'$. Since $X' \subset X$, we have that $V(H) \cap V(G) \subset X$. We claim that this is enough to prove the lemma. Indeed, let $\mathcal{P}$ be the property for which C is a certificate. If C is a global certificate for X in G, it follows that $C \cup H$ has property $\mathcal{P}$ if and only if $G \cup H$ has property $\mathcal{P}$. Thus, C is a global certificate for X' in G. If C is a local certificate for X in G, it follows that, given any two vertices $x, y \in V(H)$, $\mathcal{P}$ is true for x and y in $C \cup H$ if and only if $\mathcal{P}$ is true for x and y in $G \cup H$. Thus, C is a local certificate for X' in G too. If C is both a local and a global certificate for X in G, then by the previous argument, C will be both a local and a global certificate for X' in G. □

We showed in [11] that, under certain weak assumptions, the existence of compressed certificates for all G and X is sufficient to prove the existence of a linear-time algorithm for computing such certificates. We require our certificates to satisfy the following additional property.

DEFINITION 3.5. *Given a graph G and a set of interesting vertices X, we say that a certificate C for X in G* preserves planarity *if, for any H such that $V(H) \cap V(G) \subseteq X$, if $G \cup H$ is planar, $C \cup H$ will also be planar.*

According to Definition 3.5, $C \cup H$ may be planar even when $G \cup H$ is not. As examples of planarity-preserving certificates, C may itself be a certificate for planarity; alternately, C may be a subgraph or minor of G.

The following lemma is proved in the companion paper [11] for global certificates and the proof can be easily extended to local and full certificates.

LEMMA 3.4 (see [11]). *Let $\mathcal{P}$ be a property for which there exist compressed certificates that preserve planarity. Then in linear time we can compute a compressed certificate for $\mathcal{P}$.*

We now describe an abstract version of our sparsification technique. We first consider global certificates (used to maintain global graph properties) and then show how the same technique can be made to work with local certificates, used to maintain local properties.

Let $\mathcal{P}$ be a property of planar graphs for which we can find compressed global certificates in time $T(n) = \Omega(n)$ and such that we can construct a data structure for testing property $\mathcal{P}$ in time $P(n)$ which can answer queries in time $Q(n)$. Then we wish to use these global certificates to maintain $\mathcal{P}$ quickly.

We construct a separator tree for the graph, by finding a set of $cn^{1/2}$ vertices (for some constant c) which splits the remaining graph into two components of less than $2n/3$ vertices each, and repeatedly split each component until there are $O(n^{1/2})$ components of size $O(n^{1/2})$ each; we call these the *leaf components*. This can all

be done in $O(n)$ time [23]. The resulting tree has height $O(\log n)$. When an edge connects two separator vertices, we arbitrarily choose which component to include it in, so each edge is included in a unique leaf component. Each time we insert a new edge, we will include its two endpoints in the separator for the node in the tree (if one exists) for which the two vertices are in the two separate components. After $O(n^{1/2})$ insertions, we reconstruct the separator tree, in amortized time $O(n^{1/2})$ per insertion.

At each node in the tree, the *interesting vertices* are those that are used either in the separator for that node or for separators at higher levels in the tree. Note that there will initially be at most

$$cn^{\frac{1}{2}} \sum_{i=0}^{\lceil \log n \rceil} \left(\frac{2}{3}\right)^{\frac{i}{2}} = O\left(n^{\frac{1}{2}}\right)$$

interesting vertices per tree node, and at most $O(n^{1/2})$ interesting vertices can be added by insertions before we reconstruct the tree. By the construction above, leaf components can share only interesting vertices. Furthermore, a vertex that is not interesting (in any leaf component) belongs to exactly one leaf component.

Each tree node corresponds to a subgraph which will be represented by a compressed global certificate for its interesting vertices. We form this global certificate by taking the union of the two compressed global certificates for the two daughter nodes (which by Lemma 3.2 is a global certificate for the graph at the node itself), and then computing a compressed global certificate of this union (which by Lemma 3.1 is also a global certificate for the node). We construct the data structure for testing property $\mathcal{P}$ using the global certificate at the tree root. This allows us to test property $\mathcal{P}$ in $Q(O(n^{1/2}))$ time.

When we reconstruct the separator tree, we must also reconstruct the global certificates, in $T(O(n^{1/2}))$ time per tree node. There are $O(n^{1/2})$ tree nodes, and we reconstruct after every $O(n^{1/2})$ insertions, so the amortized time per insertion is $T(O(n^{1/2}))$.

When we perform an insertion of an edge (x, y) that does not reconstruct the separator tree, we may move the two vertices x and y into the separator of a tree node N; then in all nodes descending from N and containing either of the two vertices, x and y may become newly interesting, and we must recompute the global certificates. However, this can happen only if either x or y was not interesting already. In other words, only the global certificates in the path between N and at most two leaves need to be updated. Furthermore, we must also recompute certificates for all tree nodes containing the newly inserted edge: these are exactly the nodes between N and the root of the separator tree. In either case, $O(\log n)$ tree nodes need recomputation, and the time to recompute certificates in each node is $T(O(n^{1/2}))$. Finally, we reconstruct the data structure for testing property $\mathcal{P}$ in the global certificate at the tree root in $P(O(n^{1/2}))$ time. The implementation of deletion is similar; here, too, we recompute global certificates in $O(\log n)$ nodes, in the same time bound.

Thus there is a fully dynamic algorithm for maintaining $\mathcal{P}$, which takes $P(O(n^{1/2}))$ $+ T(O(n^{1/2}))O(\log n)$ amortized time per edge insertion or deletion, and $Q(O(n^{1/2}))$ time per query. The amortized bound can be made worst-case by standard techniques of keeping two copies of the data structure, one of which can be gradually rebuilt while the other is being used.

In the companion paper [11], we develop a more complicated variant of this technique that allows us to save an $O(\log n)$ factor in the time bound above. The basic

idea is to use a separator tree which also partitions the interesting vertices evenly in the tree. In this way the nodes at lower levels of the separator tree will be able to have certificates smaller than $O(n^{1/2})$. In order to maintain this property of the separator tree we must then recompute lower-level separators after smaller numbers of updates.

DEFINITION 3.6. *Let G be a planar graph. A* balanced separator tree for G *is a separator tree such that a node at level i, $i \geq 0$, has at most $ab^i n^{1/2}$ interesting vertices, for some constants $a > 0$ and $0 < b < 1$.*

The proofs of the following two theorems are in [11].

THEOREM 3.1 (see [11]). *A balanced separator tree can be constructed in linear time.*

THEOREM 3.2 (see [11]). *Let $\mathcal{P}$ be a graph property for which we can find compressed global certificates in time $T(n) = \Omega(n)$ and such that we can construct, in $P(n)$ time, a data structure that tests property $\mathcal{P}$ in $Q(n)$ time. Then there is a fully dynamic algorithm for maintaining $\mathcal{P}$ in a planar graph subject to insertions and deletions preserving planarity, which takes $P(O(n^{1/2})) + T(O(n^{1/2}))$ amortized time per edge insertion or deletion and $Q(O(n^{1/2}))$ time per global query.*

We next describe how sparsification may apply to local properties such as vertex and edge connectivity.

THEOREM 3.3. *Let $\mathcal{P}$ be a local graph property for which we can find compressed local certificates in time $T(n) = \Omega(n)$ and such that we can construct a data structure for testing property $\mathcal{P}$ in time $Q(n)$. Then there is a fully dynamic algorithm for maintaining $\mathcal{P}$ in a planar graph, which takes amortized time $T(O(n^{1/2}))$ per edge insertion or deletion, and worst-case time $Q(O(n^{1/2})) + T(O(n^{1/2}))$ per local query.*

Proof. We use the balanced separator tree of Theorem 3.1, but this time we store at its nodes local certificates rather than global certificates. The amortized bound for updates follow now from Theorem 3.2. To test the local property $\mathcal{P}$ for two given vertices x and y, we first make x and y interesting vertices in the local certificate at the tree root. Once x and y are interesting, it is then easily verified that a local certificate for local property $\mathcal{P}$ is a global certificate for the simple property $\mathcal{P}(x, y)$. To make x and y interesting, we reconstruct the local certificates of all nodes containing either one of them. We do not reconstruct the separator tree even if the operation should normally do so. As in the proof of Theorem 3.2, this involves recomputing local certificates in $O(\log n)$ nodes in the separator tree of sizes increasing in a geometric series, and therefore can be done in $T(O(n^{1/2}))$ time. To answer a query regarding property $\mathcal{P}$ for vertices x and y, we construct the data structure for testing property $\mathcal{P}$ in the local certificate at the tree root in $P(O(n^{1/2}))$ time. Finally, we undo all the changes we made. □

Theorems 3.2 and 3.3 can be combined as follows.

THEOREM 3.4. *Let $\mathcal{P}$ be a local graph property for which we can find compressed full certificates in time $T(n) = \Omega(n)$ and such that we can construct a data structure for testing property $\mathcal{P}$ in time $Q(n)$. Then there is a fully dynamic algorithm for maintaining $\mathcal{P}$ in a planar graph, which takes amortized time $T(O(n^{1/2}))$ per edge insertion or deletion, and worst-case time $Q(O(n^{1/2})) + T(O(n^{1/2}))$ per either global or local query.*

4. Properties of certificates for edge connectivity. Let G be an undirected graph, with interesting vertices $X \subseteq V(G)$. In this section we give necessary and sufficient conditions for a graph C to be a full certificate of k-edge-connectivity for X in G. Namely, we will show that it is crucial to keep information about the connectivity edge-cuts that involve only the interesting vertices in X. This will be exploited in the

next sections in order to build our full certificates for edge connectivity.

Let V_1 and V_2 be any two nonempty disjoint subsets of vertices in G. We say that a set of edges $E' \subseteq E(G)$ *disconnects* V_1 *and* V_2 if removing all the edges in E' from G leaves no path between vertices in V_1 and vertices in V_2. We denote by $\lambda_{V_1,V_2}(G)$ the *minimum* number of edges of G whose removal disconnects V_1 and V_2. Note that $\lambda_{V_1,V_2}(G) = \lambda_{V_2,V_1}(G)$. When $V_1 = \{u\}$ and $V_2 = \{v\}$, we obtain the definition of edge connectivity between vertices u and v. The edge connectivity $\lambda(G)$ of G satisfies the equality

$$\lambda(G) = \min_{u,v \in V(G)} \{\lambda_{u,v}(G)\} = \min_{\substack{\emptyset \subset V_1, V_2 \subset V(G) \\ V_1 \cap V_2 = \emptyset}} \{\lambda_{V_1,V_2}(G)\}.$$

Let $\emptyset \subset R \subset X$; then we denote by $\lambda_R(G)$ the quantity $\lambda_{R,X-R}(G)$. If $R = \emptyset$ or $R = X$, we instead let $\lambda_R(G)$ denote the minimum number of edges that must be removed from G in order to disconnect X from at least one other vertex in $[V(G) - X]$.

We can use this notation to characterize full certificates of edge connectivity. Before doing this, we need a technical lemma. Often the easiest way to show that a graph C is a certificate of edge connectivity is to show that appropriate edge-cuts exist. These edge-cuts are edge-cuts of a graph that is the union of two other graphs. The following lemma gives sufficient conditions for constructing an edge-cut of $G \cup H$ from an edge-cut of G and an edge-cut of H.

LEMMA 4.1. *Let G and H be graphs such that $V(G) \cap V(H) \subseteq X$. Furthermore, let γ_G and γ_H be edge-cuts of G and H, respectively. Suppose that γ_G divides G into G_1 and G_2 and that γ_H divides H into H_1 and H_2. Then, if $(V(G_1) \cap V(H_2)) \cup (V(G_2) \cap V(H_1)) = \emptyset$, then $\gamma = \gamma_G \cup \gamma_H$ is an edge-cut of $G \cup H$ that divides it into $G_1 \cup H_1$ and $G_2 \cup H_2$.*

Proof. No edge outside $\gamma_G \cup \gamma_H$ can cross from $G_1 \cup H_1$ to $G_2 \cup H_2$. For if such an edge belonged to G, it would have to cross from $(G_1 \cup H_1) \cap G = G_1$ to $(G_2 \cup H_2) \cap G = G_2$ and would therefore belong to γ_G, and symmetrically, if such an edge belonged to H, it would belong to γ_H. □

Suppose that we are interested in k-edge-connectivity for $k \leq \mathcal{K}$. Then, for C to be a certificate of k-edge-connectivity for X in G, the edge-cuts with k or fewer edges in G should correspond to edge-cuts of the same size in C. Making this formal, we have the following.

LEMMA 4.2. *If C is a full certificate of k-edge-connectivity for X in G, for every $k \leq \mathcal{K}$, then, for any subset $\emptyset \subseteq R \subseteq X$ of interesting vertices,*

$$\min\{\lambda_R(G), \mathcal{K}\} = \min\{\lambda_R(C), \mathcal{K}\}. \tag{4.1}$$

Proof. Suppose that for some $\emptyset \subseteq R \subseteq X$, $\min\{\lambda_R(G), \mathcal{K}\} \neq \min\{\lambda_R(C), \mathcal{K}\}$. We want to find a graph H that shows that C is not a full certificate of k-edge-connectivity for X in G. Note that either $\lambda_R(C) < \mathcal{K}$ or $\lambda_R(G) < \mathcal{K}$.

If $R = \emptyset$ or $R = X$, let H have vertices $X \cup \{z\}$ and $\mathcal{K}$ edges between z and each vertex in X. Any minimal edge-cut of $G \cup H$ that contains any edges in H must then contain all $\mathcal{K}$ of the edges between z and some vertex $x \in X$. Any edge-cut that does not contain any edges from H must separate X from some other vertex in G. Thus, if $\lambda(G \cup H) < \mathcal{K}$, then $\lambda(G \cup H) = \lambda_\emptyset(G) = \lambda_R(G)$. Similarly, the edge connectivity $\lambda(C \cup H)$ is $\lambda_\emptyset(C) = \lambda_R(C)$ unless $\lambda_\emptyset(C) > \mathcal{K}$, in which case the edge connectivity is at least $\mathcal{K}$. Therefore, since $\lambda_\emptyset(C) < \mathcal{K}$ or $\lambda_\emptyset(G) < \mathcal{K}$, we have that

$\lambda(G \cup H) \neq \lambda(C \cup H)$, so C is not a global certificate of k-edge-connectivity for some $k \leq \mathcal{K}$.

Alternately, it may be the case that $\emptyset \subset R \subset X$. In this case, construct H with vertices $X \cup \{z_1, z_2\}$, $\mathcal{K}$ edges from z_1 to each vertex in R, and $\mathcal{K}$ edges from z_2 to each vertex in $X - R$. Again, no minimal edge-cut separating z_1 and z_2 in $G \cup H$ contains an edge of H unless it contains all $\mathcal{K}$ of the edges between two vertices in H. Thus, any edge-cut separating z_1 and z_2 and containing no edges of H must separate R and $X - R$. This implies that if $\lambda_{z_1,z_2}(G \cup H) < \mathcal{K}$, then $\lambda_{z_1,z_2}(G \cup H) = \lambda_R(G)$. Similarly, the edge-connectivity between z_1 and z_2 in $C \cup H$ is $\lambda_R(C)$ unless $\lambda_R(C) > \mathcal{K}$, in which case the edge-connectivity is at least $\mathcal{K}$. Therefore, since $\lambda_R(C) < \mathcal{K}$ or $\lambda_R(G) < \mathcal{K}$, we have that $\lambda_{z_1,z_2}(G \cup H) \neq \lambda_{z_1,z_2}(C \cup H)$, so C is not a local certificate of k-edge-connectivity for some $k \leq \mathcal{K}$. □

Not only is (4.1) a necessary condition for C to be a full certificate, it is also sufficient. First, let us see that it is a sufficient condition for C to be a local certificate.

LEMMA 4.3. *Let $\mathcal{K}$ be a given integer. Let G be a given graph, with interesting vertices $X \subseteq V(G)$. If for any proper subset R of interesting vertices, $\emptyset \subset R \subset X$:*

$$\min\{\lambda_R(G), \mathcal{K}\} = \min\{\lambda_R(C), \mathcal{K}\},$$

then C is a local certificate of k-edge-connectivity for X in G, for every $k \leq \mathcal{K}$.

Proof. This is equivalent to showing that, for any graph H such that $V(H) \cap V(G) \subseteq X$, $V(H) \cap V(C) \subseteq X$, and for any two vertices $x, y \in H$, the following is true:

1. if $\lambda_{x,y}(G \cup H) < \mathcal{K}$, then $\lambda_{x,y}(C \cup H) \leq \lambda_{x,y}(G \cup H)$, and
2. if $\lambda_{x,y}(C \cup H) < \mathcal{K}$, then $\lambda_{x,y}(G \cup H) \leq \lambda_{x,y}(C \cup H)$.

Now we would like to see that 1 holds. Let γ be a minimum edge-cut that disconnects x from y in $G \cup H$. Since it is a minimal edge-cut, it separates $G \cup H$ into exactly two pieces S_1 and S_2. Without loss of generality, assume that $x \in S_1$ and $y \in S_2$. Let $X_1 = S_1 \cap X$, $X_2 = S_2 \cap X$, $G_1 = S_1 \cap G$, $G_2 = S_2 \cap G$, $H_1 = S_1 \cap H$, and $H_2 = S_2 \cap H$. Note that $x \in H_1$ and $y \in H_2$. Further, let $\gamma_G = \gamma \cap E(G)$ and $\gamma_H = \gamma \cap E(H)$.

If $X_2 = \emptyset$, then $X = X_1$, $G_2 = \emptyset$, and $G = G_1$. So, $\gamma_G = \emptyset$ and $\gamma = \gamma_H$. In this case, $\gamma = \gamma_H$ disconnects $G \cup H$ into $G \cup H_1$ and H_2, with $x \in H_1$ and $y \in H_2$. Since $V(C) \cap V(H) \subseteq X = X_1$, replacing G with C yields that $\gamma = \gamma_H$ disconnects $C \cup H$ into $C \cup H_1$ and H_2, with $x \in (G \cup H_1)$ and $y \in H_2$. Hence, the very same edge-cut γ disconnects x and y in $C \cup H$. Similarly, we can assume that $X_1 \neq \emptyset$.

If $|\gamma| \geq \mathcal{K}$, there is nothing to prove. Otherwise, we need to construct γ' as an edge-cut of $C \cup H$ that disconnects x from y and such that $|\gamma'| \leq |\gamma|$. Note that γ_G disconnects X_1 from X_2 in G, so $|\gamma_G| \geq \lambda_{X_1,X_2}(G)$. Since $X = X_1 \cup X_2$, $\lambda_{X_1,X_2}(G) = \lambda_{X_1}(G)$. Moreover, since $|\gamma_G| \leq |\gamma| < \mathcal{K}$, $\lambda_{X_1}(G) = \lambda_{X_1}(C) = \lambda_{X_1,X_2}(C)$. This means that there is an edge-cut γ_C of C that disconnects X_1 from X_2 such that $|\gamma_C| \leq |\gamma_G|$. Thus, if $\gamma' = \gamma_H \cup \gamma_C$, then $|\gamma'| \leq |\gamma|$.

Now we would like to see that γ' disconnects x from y in $H \cup C$. Define C_1 and C_2 so that γ_C divides C into C_1 and C_2. Note that γ_C and γ_H both partition X in the same way, so we can assume that $X_1 \subseteq V(C_1)$ and $X_2 \subseteq V(C_2)$. Since $V(C) \cap V(H) \subseteq X$, this means that $(V(C_1) \cup V(H_1)) \cap (V(C_2) \cup V(H_2)) = \emptyset$, so Lemma 4.1 says that γ' is an edge-cut of $C \cup H$ that disconnects x from y.

To prove that 2 holds, we use exactly the same proof with the roles of G and C switched. Therefore, C is a local certificate of k-edge-connectivity for X in G. □

If the condition also holds when $R = \emptyset$ or $R = X$, then C is also a global certificate.

LEMMA 4.4. *Let $\mathcal{K}$ be a given integer. Let G be a given graph, with interesting vertices $X \subseteq V(G)$. If for any subset R of interesting vertices, $\emptyset \subseteq R \subseteq X$:*

$$\min\{\lambda_R(G), \mathcal{K}\} = \min\{\lambda_R(C), \mathcal{K}\},$$

then C is a global certificate of k-edge-connectivity for X in G, for every $k \leq \mathcal{K}$.

Proof. Let H be any graph such that $V(H) \cap V(G) \subseteq X$. Suppose that γ is a minimum k-edge-cut of $G \cup H$ that divides the graph into S_1 and S_2, and that $k < \mathcal{K}$. If both S_1 and S_2 contain vertices in H, then the fact that C is a local certificate of k-edge-connectivity for X in G shows that there is a k-edge-cut of $C \cup H$. Alternately, if S_1 (say) contains no vertices from H, then γ contains only edges from G, since every edge in a minimal edge-cut connects a vertex in S_1 to a vertex in S_2. This means that $\lambda_\emptyset(G) \leq k$. Since $k < \mathcal{K}$, $\lambda_\emptyset(G) = \lambda_\emptyset(C) \leq k$, so $C \cup H$ has a k-edge-cut as well.

Conversely, suppose that γ' is a minimum k-edge-cut of $C \cup H$ that divides the graph into S_1 and S_2, and that $k < \mathcal{K}$. If both S_1 and S_2 contain vertices in H, then the fact that C is a local certificate of k-edge-connectivity for X in G shows that there is a k-edge-cut of $G \cup H$. Alternately, if S_1 (say) contains no vertices from H, then γ' contains only edges from C. This means that $\lambda_\emptyset(C) \leq k$. Since $k < \mathcal{K}$, $\lambda_\emptyset(C) = \lambda_\emptyset(G) \leq k$, so $G \cup H$ has a k-edge-cut as well. Therefore, C is a global, as well as local, certificate of k-edge-connectivity provided that $k \leq \mathcal{K}$. □

Putting these three lemmas together, we have Theorem 4.1.

THEOREM 4.1. *Let $\mathcal{K}$ be a given integer. Let G be a given graph, with interesting vertices $X \subseteq V(G)$. C is a full certificate of k-edge-connectivity for X in G, for every $k \leq \mathcal{K}$, if and only if for any subset R of interesting vertices, $\emptyset \subseteq R \subseteq X$:*

$$\min\{\lambda_R(G), \mathcal{K}\} = \min\{\lambda_R(C), \mathcal{K}\}.$$

4.1. Split graphs and certificates. We now prove some properties about splitting 2-edge-connected graphs and computing their certificates for edge connectivity. Let G be a 2-edge-connected graph, and let $\{e_1, e_2\}$ be a 2-edge-cut in G. Let G_1 and G_2 be the two graphs obtained from G after the deletion of $\{e_1, e_2\}$, and let $e_1 = (u_1, u_2)$ and $e_2 = (v_1, v_2)$ be such that u_1, v_1 are in G_1 and u_2, v_2 are in G_2. We call this a *split* and call $G_1 \cup \{(u_1, v_1)\}$ and $G_2 \cup \{(u_2, v_2)\}$ the two *split graphs of G with respect to the 2-edge-cut $\{e_1, e_2\}$*. Note that $\{(u_1, v_1)\}$ and $\{(u_2, v_2)\}$ are not originally edges of G: they are called the *virtual edges* associated with the split. The operation inverse to a split is called a *merge*: it takes two split graphs with respect to the same 2-edge-cut and merges them back together, yielding the original graph. We observe that splits and merges preserve planarity. Indeed, the split graphs $G_1 \cup \{(u_1, v_1)\}$ and $G_2 \cup \{(u_2, v_2)\}$ can be obtained from G by means of edge contractions (for instance, $G_1 \cup \{(u_1, v_1)\}$ can be obtained after contracting e_1 and all the edges in G_2). Hence, if G is planar, the split graphs $G_1 \cup \{(u_1, v_1)\}$ and $G_2 \cup \{(u_2, v_2)\}$ obtained after a split are planar. Conversely, if the split graphs are planar, the graph obtained after a merge will be planar. Figure 3 illustrates splits and merges.

LEMMA 4.5. *Let G be a 2-edge-connected graph, and let $k \geq 2$ be an integer. Let $\{e_1, e_2\}$ be any 2-edge-cut of G. For $i = 1, 2$ let us denote by $G_i \cup \{(u_i, v_i)\}$ the split graphs of G with respect to the 2-edge-cut $\{e_1, e_2\}$. Let x and y be any two vertices in the same split graph $G_i \cup \{(u_i, v_i)\}$. Then x and y are k-edge-connected in $G_i \cup \{(u_i, v_i)\}$ if and only if they are k-edge-connected in G.*

Proof. Without loss of generality, let x and y be any two vertices of $G_1 \cup \{(u_1, v_1)\}$. Since G is 2-edge-connected, any two edges of G are contained in a cycle. In particular,

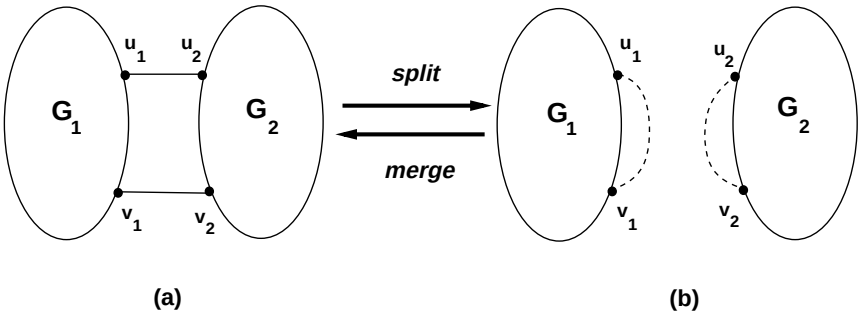


FIG. 3. *Splitting and merging a 2-edge-connected graph and its split graphs. Virtual edges are dashed.*

there is a cycle in G containing edges e_1 and e_2. This implies that there exists a path π_1 between vertices u_1 and v_1 that is entirely contained in G_1, and a path π_2 between u_2 and v_2 that is entirely contained in G_2.

Assume x and y are k-edge-connected in $G_1 \cup \{(u_1, v_1)\}$. Then there are k edge-disjoint paths in $G_1 \cup \{(u_1, v_1)\}$ between x and y, and at most one of them can use the edge (u_1, v_1). If none of these paths use (u_1, v_1), there will be k edge-disjoint paths between x and y in G. If one of these paths use (u_1, v_1), then the path in G containing e_1, π_2, and e_2 in place of (u_1, v_1) is a path between x and y that is edge-disjoint from the other $(k-1)$ paths in G_1. In both cases, x and y are k-edge-connected in G.

Conversely, assume that x and y are k-edge-connected in G. Again, at most one of the k edge-disjoint paths between x and y can go through G_2. If all the k-edge-disjoint paths between x and y are contained in G_1, x and y are k-edge-connected in $G_1 \cup \{(u_1, v_1)\}$ too. If one of these paths go through G_2, replacing this portion of the path with the edge (u_1, v_1) gives a path in $G_1 \cup \{(u_1, v_1)\}$ which is edge-disjoint from the other $(k-1)$ paths. □

The following corollary is an easy consequence of Lemma 4.5.

COROLLARY 4.1. *Let G be a 2-edge-connected graph, and let $\{e_1, e_2\}$ be any 2-edge-cut of G. The split graphs of G with respect to $\{e_1, e_2\}$ are 2-edge-connected.*

Since the split graphs of G are 2-edge-connected, the same splitting can be applied recursively to the split graphs of G, and to their split graphs, and so on. When no further splits are possible, each split graph left is 3-edge-connected and corresponds to exactly one 3-edge-connected class of the original graph. This gives another way of defining the cactus tree of a graph with edge connectivity 2. We call these final split graphs the 3-*edge-connected components* of G. We point out here a difference between 2-edge-connectivity and 3-edge-connectivity. For 2-edge-connectivity, the subgraph induced by a 2-edge-connected class is 2-edge-connected and coincides with a 2-edge-connected component. On the contrary, the subgraph induced by a 3-edge-connected class may differ from a 3-edge-connected component. Indeed, the subgraph induced by a 3-edge-connected class is not necessarily even connected, while the addition of the virtual edges makes a 3-edge-connected component 3-edge-connected. We remark that this decomposition of a 2-edge-connected graph is similar to the decomposition of a biconnected graph into its triconnected components [25], and it is implicit in the work of Dinitz [5].

Let G be a 2-edge-connected graph, and consider the following operation: split G and compute a full certificate of k-edge-connectivity, $k \geq 2$, for one of its split graphs; then merge this certificate with the other split graph. We will prove that by doing

so we obtain a full certificate of k-edge-connectivity, $k \geq 2$, for the original graph G. This property will be useful later on. We now summarize the notation used.

- G: a 2-edge-connected graph;
- $X \subseteq V(G)$: the interesting vertices of G;
- $\{e_1, e_2\}$: a 2-edge-cut in G, with $e_1 = (u_1, u_2)$ and $e_2 = (v_1, v_2)$;
- $\widehat{G}_1 = G_1 \cup \{(u_1, v_1)\}$, $\widehat{G}_2 = G_2 \cup \{(u_2, v_2)\}$: the split graphs of G with respect to $\{e_1, e_2\}$; $u_1, v_1 \in V(G_1)$ and $u_2, v_2 \in V(G_2)$;
- $X_1 = (X \cap V(G_1)) \cup \{u_1, v_1\}$: the interesting vertices in G_1 augmented with u_1 and v_1;
- $\widehat{C}_1 = C_1 \cup \{(u_1, v_1)\}$: a full certificate of k-edge-connectivity for X_1 in $\widehat{G}_1$ (note that this certificate keeps the virtual edge (u_1, v_1) of $\widehat{G}_1$);
- H: any graph such that $V(H) \cap V(G) \subseteq X$;
- $F = G_2 \cup H \cup \{e_1, e_2\}$. Namely, the vertices of F are u_1, v_1 plus the vertices of G_2 and H; the edges of F are e_1, e_2 plus the edges of G_2 and H.

Note that $G = G_1 \cup \{e_1, e_2\} \cup G_2$. We further define $C = C_1 \cup \{e_1, e_2\} \cup G_2$. The point of all of this is that C is a full certificate of k-edge-connectivity for X in G. To prove this, we first prove that it is a local certificate and then prove that it is a global certificate.

LEMMA 4.6. *Let $\mathcal{K}$ be a given integer. Let $\widehat{C}_1 = C_1 \cup \{(u_1, v_1)\}$ be a local certificate of k-edge-connectivity for X_1 in $\widehat{G}_1 = G_1 \cup \{(u_1, v_1)\}$ for every $k \leq \mathcal{K}$. Then $C = C_1 \cup \{e_1, e_2\} \cup G_2$ is a local certificate of k-edge-connectivity for X in G for every $k \leq \mathcal{K}$.*

Proof. Again, the idea behind the proof is to show that the appropriate edge-cuts exist. Specifically, we need to show that for any H such that $V(G) \cap V(H) \subseteq X$, $V(C) \cap V(H) \subseteq X$, and for any $x, y \in V(H)$:

1. if $\lambda_{x,y}(G \cup H) < \mathcal{K}$ then $\lambda_{x,y}(C \cup H) \leq \lambda_{x,y}(G \cup H)$, and
2. if $\lambda_{x,y}(C \cup H) < \mathcal{K}$ then $\lambda_{x,y}(G \cup H) \leq \lambda_{x,y}(C \cup H)$.

Suppose that γ is a minimum edge-cut that disconnects x from y in $G \cup H$ and that $|\gamma| < \mathcal{K}$. Since γ is a minimum edge-cut, it separates $G \cup H$ into two pieces $S^{(x)}$ and $S^{(y)}$, with $x \in S^{(x)}$ and $y \in S^{(y)}$. Let $G_1^{(x)} = G_1 \cap S^{(x)}$, $G_1^{(y)} = G_1 \cap S^{(y)}$, $F^{(x)} = F \cap S^{(x)}$, $F^{(y)} = F \cap S^{(y)}$, $X_1^{(x)} = X_1 \cap S^{(x)}$, and $X_1^{(y)} = X_1 \cap S^{(y)}$. Note that $G_1 = G_1^{(x)} \cup G_1^{(y)}$, $X_1 = X_1^{(x)} \cup X_1^{(y)}$, and $F = F^{(x)} \cup F^{(y)}$. Since $x \in (S^{(x)} \cap H)$ and $y \in (S^{(y)} \cap H)$, we must have $x \in F^{(x)}$ and $y \in F^{(y)}$. Also let $\gamma_{G_1} = \gamma \cap G_1$ and let $\gamma_F = \gamma - \gamma_{G_1}$. Note that γ_F disconnects $X_1^{(x)} \cap V(F)$ from $X_1^{(y)} \cap V(F)$ in F.

If γ_{G_1} is empty, then $\gamma_F = \gamma$. G_1 is connected by the assumption that G is biconnected, so one of $G_1^{(x)}$ or $G_1^{(y)}$ is empty and the other one contains all of G_1. Without loss of generality $G_1^{(x)} = \emptyset$. C_1 is not reachable from x in $C \cup H - \gamma$, since any path connecting x to C_1 in $C \cup H$ would connect x to G_1 in $G \cup H$, and so must be cut by γ. Then x is separated from y in $C \cup H - \gamma$, since no path from x can go through C_1, and since any path avoiding C_1 would also exist in $G \cup H - \gamma$.

Alternately, assume that γ_{G_1} is not empty. We claim that in this case neither $X_1^{(x)}$ nor $X_1^{(y)}$ can be empty. Indeed, if either $X_1^{(x)}$ or $X_1^{(y)}$ were empty, then the removal of all the edges of γ from $G \cup H$ would leave all the vertices of X_1 still connected. Since $V(H) \cap V(G_1) \subseteq X_1$, any path of $G \cup H$ between $x \in V(H)$ and $y \in V(H)$ that contains an edge of G_1 must also contain a vertex of X_1 (recall that, by definition, $u_1, v_1 \in X_1$). Thus, in this case, γ_F would be an edge-cut of $G \cup H$ that disconnects x from y. Since γ is by assumption a minimum edge-cut, this implies by contradiction that neither $X_1^{(x)}$ nor $X_1^{(y)}$ can be empty.

Without loss of generality, assume that $u_1 \in G_1^{(x)}$. We have two cases depending on whether $v_1 \in G_1^{(x)}$ or $v_1 \in G_1^{(y)}$. Suppose first that $v_1 \in G_1^{(x)}$. In this case, γ_{G_1} is an edge-cut of $\widehat{G_1}$ that disconnects $X_1^{(x)}$ from $X_1^{(y)}$. Consequently, $\lambda_{X_1^{(x)}}(\widehat{G_1}) \leq |\gamma_{G_1}|$. Since $\widehat{C_1}$ is a local certificate of k-edge-connectivity for X_1 in $\widehat{G_1}$, and $|\gamma_{G_1}| \leq |\gamma| < \mathcal{K}$, by Theorem 4.1 we have that $\lambda_{X_1^{(x)}}(\widehat{C_1}) = \lambda_{X_1^{(x)}}(\widehat{G_1}) \leq |\gamma_{G_1}|$. This is equivalent to saying that there is an edge-cut $\gamma_{\widehat{C_1}}$ of $\widehat{C_1}$ that disconnects $X_1^{(x)}$ from $X_1^{(y)}$ and such that $|\gamma_{\widehat{C_1}}| = \lambda_{X_1^{(x)}}(\widehat{C_1}) \leq |\gamma_{G_1}|$. Since we can take $\gamma_{\widehat{C_1}}$ to be a minimal edge-cut, it does not contain the edge (u_1, v_1), so it is an edge-cut of C_1. Thus $\gamma' = \gamma_{\widehat{C_1}} \cup \gamma_F$ is a set of edges of $C \cup H$ with $|\gamma'| \leq |\gamma|$. Moreover, $\gamma_{\widehat{C_1}}$ divides C_1 into $C_1^{(x)}$ and $C_1^{(y)}$. Since $X_1^{(x)} \subseteq V(C_1^{(x)})$ and $X_1^{(y)} \subseteq V(C_1^{(y)})$, we have that $C_1^{(x)} \cap F^{(y)} = \emptyset$ and $C_1^{(y)} \cap F^{(x)} = \emptyset$, so Lemma 4.1 shows that $\gamma' = \gamma_{\widehat{C_1}} \cup \gamma_F$ is an edge-cut of $C_1 \cup F = C \cup H$ that disconnects x from y.

Alternately, it could be the case that $v_1 \in G_1^{(y)}$. In this case, γ_{G_1} is not an edge-cut of $\widehat{G_1}$, but $\gamma_{G_1} \cup \{(u_1, v_1)\}$ is such an edge-cut. Again, Theorem 4.1 says that there is an edge-cut $\widehat{\gamma_{\widehat{C_1}}}$ that disconnects $X_1^{(x)}$ from $X_1^{(y)}$ in $\widehat{C_1}$ such that $|\widehat{\gamma_{\widehat{C_1}}}| \leq |\gamma_{\widehat{C_1}}| + 1$. Moreover, since $u_1 \in X_1^{(x)}$ and $v_1 \in X_1^{(y)}$, the edge-cut $\widehat{\gamma_{\widehat{C_1}}}$ contains the edge (u_1, v_1). Let $\gamma_{\widehat{C_1}} = \widehat{\gamma_{\widehat{C_1}}} - \{(u_1, v_1)\}$. Then $\gamma' = \gamma_{\widehat{C_1}} \cup \gamma_F$ is a set of edges in $C \cup H$ with $|\gamma'| \leq |\gamma|$. Moreover, $\gamma_{\widehat{C_1}}$ disconnects $X_1^{(x)}$ from $X_1^{(y)}$, so again, by Lemma 4.1, γ' is an edge-cut that disconnects x from y in $C \cup H$.

Therefore, in either case, γ' is the desired edge-cut, and the first property holds. Since $\widehat{C_1}$ is a certificate of k-edge-connectivity for X_1 in $\widehat{G_1}$, it is the case that $\widehat{G_1}$ is a certificate for k-edge-connectivity of X_1 in $\widehat{C_1}$. Therefore, the same proof can be used to prove property 2, and C is a global certificate of k-edge-connectivity for X in G. □

To prove that C is a full certificate, we also need to prove that it is a global certificate. The proof of this fact is annoyingly similar to the proof that C is a local certificate, but there seems to be no clear way to combine the proofs.

LEMMA 4.7. *Let $\mathcal{K}$ be a given integer. Let $\widehat{C}_1 = C_1 \cup \{(u_1, v_1)\}$ be a full certificate of k-edge-connectivity for X_1 in $\widehat{G}_1 = G_1 \cup \{(u_1, v_1)\}$ for every $k \leq \mathcal{K}$. Then $C = C_1 \cup \{e_1, e_2\} \cup G_2$ is a global certificate of k-edge-connectivity for X in G for every $k \leq \mathcal{K}$.*

Proof. Again it suffices to show that for any H with $V(H) \cap V(G) \subseteq X$, the following two properties hold:

1. if $\lambda(C \cup H) \leq \mathcal{K}$ then $\lambda(G \cup H) \leq \lambda(C \cup H)$, and
2. if $\lambda(G \cup H) \leq \mathcal{K}$ then $\lambda(C \cup H) \leq \lambda(G \cup H)$.

To prove property 1, let γ be a minimum edge-cut of $C \cup H$ that divides $C \cup H$ into S^* and S^{**}. Since γ is a minimum edge-cut in $C \cup H$, it must be of cardinality $\lambda(C \cup H)$: assume that $\lambda(C \cup H) \leq \mathcal{K}$. To prove that $\lambda(G \cup H) \leq \lambda(C \cup H)$, we distinguish several cases according to S^*, S^{**}, and H.

Assume first that both S^* and S^{**} contain vertices of H: thus, there exist two vertices, say x and y, in $V(H)$ that are separated by γ. By Lemma 4.6, we know that C is a local certificate of k-edge-connectivity for X in G, for every $k \leq \mathcal{K}$. Then, if $\lambda(C \cup H) \leq \mathcal{K}$, by Definition 3.2 there is an edge-cut γ' separating x and y in $G \cup H$ of cardinality exactly $\lambda(C \cup H)$. As a result, the minimum edge-cut of $G \cup H$ must have cardinality less than or equal to the cardinality of γ' and hence $\lambda(G \cup H) \leq \lambda(C \cup H)$.

Alternately, it may be the case that $V(H) \subseteq V(S^*)$. In this case, γ contains only edges in C. There are three subcases, depending on how many of the vertices v_1 and u_1 are in $V(S^{**})$. If neither vertex is in $V(S^{**})$, then either $\gamma \subseteq G_2 \cup \{e_1, e_2\}$, in which case γ is also an edge-cut of $G \cup H$, or $\gamma \subseteq C_1$. In the latter case, γ is also an edge-cut of $\widehat{C_1}$, so there is an edge-cut γ' of $\widehat{G_1}$ that is also an edge-cut of $G \cup H$ such that $|\gamma'| \leq |\gamma|$, since $\widehat{C_1}$ is a global certificate of k-edge-connectivity for X_1 in $\widehat{G_1}$.

Another possibility is that both u_1 and v_1 are in S^{**}. Here γ contains edges from both C_1 and G_2, since it is minimal. Let $\gamma_{C_1} = \gamma \cap C_1$ and $\gamma_F = \gamma - \gamma_{C_1}$. Then γ_{C_1} disconnects $\{u_1, v_1\}$ from the rest of X_1 in $\widehat{C_1}$. By Theorem 4.1, there is an edge-cut γ_{G_1} that disconnects $\{u_1, v_1\}$ from the rest of X_1 in $\widehat{G_1}$. Moreover, $|\gamma_{G_1}| \leq |\gamma_{C_1}|$, so if $\gamma' = \gamma_{G_1} \cup \gamma_F$, then $|\gamma'| \leq |\gamma|$. Moreover, since γ_F disconnects $\{u_1, v_1\}$ from X_2 in F, Lemma 4.1 says that γ' is an edge-cut of $G \cup H$.

The final possibility is that one vertex is in S^* and the other is in S^{**}. Without loss of generality, assume that $u_1 \in S^{**}$ but $v_1 \in S^*$. Again let $\gamma_{C_1} = \gamma \cap C_1$ and $\gamma_F = \gamma - \gamma_{C_1}$. Here, let $\widehat{\gamma_{C_1}} = \gamma_{c_1} \cup \{(u_1, v_1)\}$. Then $\widehat{\gamma_{C_1}}$ disconnects u_1 from the rest of X_1 in $\widehat{C_1}$. Therefore, by Theorem 4.1, there is an edge-cut $\widehat{\gamma_{G_1}}$ that disconnects u_1 from the rest of X_1 in $\widehat{G_1}$ such that $|\widehat{\gamma_{G_1}}| \leq |\widehat{\gamma_{C_1}}|$. Moreover, since $\widehat{\gamma_{G_1}}$ disconnects u_1 from v_1, $(u_1, v_1) \in \widehat{\gamma_{G_1}}$, so let $\gamma_{G_1} = \widehat{\gamma_{G_1}} - \{(u_1, v_1)\}$. Since $(u_1, v_1) \notin G_1$, the edge-cut γ_{G_1} disconnects u_1 from the rest of X_1 in G_1. Moreover, γ_F disconnects u_1 from X_2 in F, so, by applying Lemma 4.1 yet again, we discover that $\gamma' = \gamma_{G_1} \cup \gamma_F$ is an edge-cut of $G \cup H$. It is also the case that $|\gamma'| \leq |\gamma|$. Therefore, we have seen that in all cases $\lambda(G \cup H) \leq \lambda(C \cup H)$, and property 1 holds.

Since $\widehat{C_1}$ is a certificate of k-edge-connectivity for X_1 in $\widehat{G_1}$, it is the case that $\widehat{G_1}$ is a certificate for k-edge-connectivity of X_1 in $\widehat{C_1}$. Therefore, the same proof can be used to prove property 2, and C is a global certificate of k-edge-connectivity for X in G. □

THEOREM 4.2. *Let $\mathcal{K}$ be a given integer, let $X \subseteq V(G)$ be the interesting vertices of G, and let $X_1 = (X \cap V(G_1)) \cup \{u_1, v_1\}$: the interesting vertices in G_1 augmented with u_1 and v_1. Let $\widehat{C}_1 = C_1 \cup \{(u_1, v_1)\}$ be a full certificate of k-edge-connectivity for X_1 in $\widehat{G}_1 = G_1 \cup \{(u_1, v_1)\}$ for every $k \leq \mathcal{K}$. Then $C = C_1 \cup \{e_1, e_2\} \cup G_2$ is a full certificate of k-edge-connectivity for X in G for every $k \leq \mathcal{K}$.*

5. Edge connectivity. In this section we present compressed full certificates for 3- and 4-edge-connectivity. Using these certificates in Theorem 3.4 yields fast, fully dynamic algorithms for maintaining information about 3- and 4-edge-connectivity in a planar graph. Our certificates for edge connectivity will be constructed using the linear time algorithm of Lemma 3.4, so they will be required to preserve planarity as well as edge connectivity. To use this construction, we merely need to show that such certificates exist, by describing an algorithm for finding them. However, we need not analyze the time bounds of this algorithm, since Lemma 3.4 will then provide an alternate algorithm with linear complexity.

Let X be the set of interesting vertices in G. Our certificates are based upon a repeated compression of G during different phases. In the first phase we shrink some edges of G so as to reduce the number of 2-edge-connected classes (and thus components) to $O(|X|)$. After this phase we could easily get compressed full certificates for 2-edge-connectivity; however, as we mentioned earlier, these certificates for 2-edge-connectivity would yield time bounds that are worse than the polylogarithmic bounds we obtain in the companion paper [11], and so we will not describe them. In the second phase, we compress each 2-edge-connected component left so as to reduce

the total number of 3-edge-connected classes (and thus components) to $O(|X|)$: compressed full certificates for 3-edge-connectivity can be computed after this phase. In the third phase, we similarly reduce the number of 4-edge-connected classes. Finally, we compress each 4-edge-connected class left so as to reduce the overall size of the graph to $O(|X|)$.

To carry out these compressions, we use the decomposition of a k-edge-connected graph into its $(k+1)$-edge-connected classes described by the cactus tree (see section 2). In each phase we use a compression that follows many of the same ideas used in the companion paper [11] to compute the minimum spanning forest certificates of size $O(|X|)$. We recall here that for the minimum spanning forest certificate we started with a tree T and then repeatedly applied the following two rules until no more rule could be applied.

(1) If $v \in T - X$ touches a single edge (u, v) in T, remove both v and edge (u, v).
(2) If $v \in T - X$ touches two edges (u, v) and (v, w) in T, remove v and replace the two edges by a single edge (u, w).

Rule (1) cuts uninteresting branches (parts of the graph not containing interesting vertices) and rule (2) shortcuts uninteresting paths (paths not containing interesting vertices). If neither rule can be applied, the resulting tree has size $O(|X|)$.

A high-level description of our computation of certificates for 3- and 4-edge-connectivity follows. We proceed one level at the time, and at each level we compress the cactus tree. Namely, at level k, $1 \le k \le 3$, we have a tree or something like a tree describing the k-edge-cuts and $(k+1)$-edge-connected classes. To reduce the number of k-edge-cuts and $(k+1)$-edge-connected classes, we compress this tree by using rules which are the analog of rules (1) and (2) above. That is, we will cut uninteresting branches (parts of the graph not containing interesting vertices and separated by a k-edge-cut) and shortcut uninteresting paths (parts not containing interesting vertices and separated by two k-edge-cuts). There are only $O(1)$ ways a branch or path may be used to connect the rest of the graph, so we will replace each branch or path by the smallest possible graph having the same edge connectivity properties, and that graph will have size $O(1)$. Then we go on to the next level.

Our description of these compressions will be given in a top-down fashion. We will first abstract three different compression problems that we need to solve. Next, we will show how to solve these problems. Finally, we will combine the solutions to these problems to achieve our certificates. The three different problems we solve are the following.

PROBLEM 5.1. *Given a planar connected graph G_0 and a set $X_1 \subseteq V(G_0)$ of vertices in G_0, find a graph G_1 that satisfies the following properties:*

(a) *$X_1 \subseteq V(G_1)$;*
(b) *G_1 has $O(|X_1|)$ 2-edge-connected components;*
(c) *For every $k \ge 2$, G_1 is a full certificate of k-edge-connectivity for X_1 in G_0;*
(d) *G_1 preserves planarity and is connected.*

PROBLEM 5.2. *Given a planar 2-edge-connected graph G_0 and a set $X_2 \subseteq V(G_0)$ of vertices in G_0, find a graph G_2 that satisfies the following properties:*

(a) *$X_2 \subseteq V(G_2)$;*
(b) *G_2 has $O(|X_2|)$ 3-edge-connected components;*
(c) *For every $k \ge 2$, G_2 is a full certificate of k-edge-connectivity for X_2 in G_0;*
(d) *G_2 preserves planarity and is 2-edge-connected.*

PROBLEM 5.3. *Given a planar 3-edge-connected graph G_0, a set $X_3 \subseteq V(G_0)$ of vertices, and a set $Y_3 \subseteq E(G_0)$ of edges in G_0, denote by $Z_3 \subseteq V(G_0)$ the set of*

endpoints of edges in Y_3. *Find a graph* G_3 *that satisfies the following properties:*

(a) $X_3 \subseteq V(G_3)$ *and* $Y_3 \subseteq E(G_3)$*;*

(b) G_3 *is a compressed full certificate of* 3- *and* 4*-edge-connectivity for* $(X_3 \cup Z_3)$ *in* G_0*;*

(c) G_3 *preserves planarity and is* 3*-edge-connected.*

Note that Problems 5.1, 5.2, and 5.3 admit the trivial solutions $G_1 = G_0$, $G_2 = G_0$, $G_3 = G_0$ whenever, respectively, $X_1 = V(G_0)$, $X_2 = V(G_0)$, $X_3 \cup Z_3 = V(G_0)$. Hence, we look for nontrivial solutions when $X_1 \subset V(G_0)$, $X_2 \subset V(G_0)$, and $X_3 \cup Z_3 \subset V(G_0)$.

Let G be a graph and let X be a set of interesting vertices. We now give a very high-level description of our algorithm that computes a compressed full certificate for 3- and 4-edge-connectivity of X in G. We will first solve Problem 5.1 with $G = G_0$ and $X = X_1$. This will give us a graph G_1 that has only $O(|X|)$ 2-edge-connected components and bridges but is still a planarity-preserving full certificate for X in G. Next, we will solve Problem 5.2 for each 2-edge-connected component of G_1, so as to reduce the overall number of 3-edge-connected components to $O(|X|)$. Finally, we will obtain our planarity-preserving compressed full certificates by solving Problem 5.3 for each 3-edge-connected component left in the graph at this point.

In the next sections, we will fill in the low-level details of our approach. In section 5.1 we show how to solve Problem 5.1, in section 5.2 we deal with Problem 5.2, and in section 5.3 with Problem 5.3. The solutions to these three problems will then be combined in section 5.4, yielding our compressed full certificates for 3- and 4-edge-connectivity.

5.1. Compressing a connected graph. In this section we present our solution to Problem 5.1. Let G_0 be a planar connected graph, and let $X_1 \subseteq V(G_0)$. We start by computing the 2-edge-connected components of G_0 [34]. As said before, the 2-edge-connected components of a graph have a tree-like structure: indeed, shrinking each 2-edge-connected class of G_0 into a super-vertex yields a tree whose edges are all and only the bridges of G_0. This is called the *bridge-block tree* of G_0. To compute the graph G_1 we work on the bridge-block tree of G_0: namely, we will apply to the bridge-block tree rules (1) and (2) of section 5. This will reduce the total number of vertices and edges in the bridge-block tree to $O(|X_1|)$. In what follows, we will often interchange the term 2-edge-connected component of G_0 with the corresponding 2-edge-connected class and with the corresponding node of its bridge-block tree, and the term bridge of G_0 with the corresponding edge of its bridge-block tree.

We now give the details of the compression and prove that it yields a solution to Problem 5.1. Let (S, T) be a minimum edge-cut separating X_1 in G_0: without loss of generality, assume that $X_1 \subseteq S$, and pick arbitrarily a vertex $t_1 \in T$. Color red the vertices of $X_1 \cup \{t_1\}$. Note that the total number of red vertices of G_0 is $|X_1| + 1$. Define a 2-edge-connected component to be *red* if it contains at least one red vertex, and define it to be *black* otherwise. Clearly, there are $O(|X_1|)$ red 2-edge-connected components. Define the *degree of a* 2*-edge-connected component* to be the number of bridges incident to it (i.e., the degree of its corresponding node in the bridge-block tree). Black 2-edge-connected components of degree one are uninteresting leaves in the bridge-block tree, and adjacent black 2-edge-connected components of degree two yield uninteresting chains in the bridge-block tree. We compress the black leaves and black chains of the bridge-block tree by applying the following two rules (analogs of rules (1) and (2)).

(B1) Let B be a black 2-edge-connected component of degree one, and let $e = (u, v)$

be the bridge incident to B. Contract e. This corresponds to deleting a black leaf from the bridge-block tree.

(B2) Let B_1 and B_2 be two adjacent black 2-edge-connected components of degree two. Let $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ be the two bridges incident to B_1, and let e_2 and $e_3 = (u_3, v_3)$ be the two bridges adjacent to B_2. Contract e_2 identifying u_2 and v_2. This corresponds to merging two adjacent black nodes of degree two in the bridge-block tree.

Note that both rules (B1) and (B2) delete one bridge and keep the graph connected. After rule (B2) is applied, the subgraph $B_1 \cup B_2$ from the contraction of bridge e_2 is 2-edge-connected. Similarly, let B' be the 2-edge-connected component adjacent to B in rule (B1): after contracting e, $B \cup B'$ is 2-edge-connected.

Let G_1 be the graph obtained from G_0 after all the rules (B1) and (B2) have been applied. Let H be any graph with $V(H) \cap V(G_0) \subseteq X_1$: since $G_1 \cup H$ is obtained from $G_0 \cup H$ by means of edge contractions, $G_1 \cup H$ is planar whenever $G_0 \cup H$ is planar. Thus, G_1 preserves planarity according to Definition 3.5. Furthermore, red vertices of G are never contracted by (B1) or (B2), and therefore $X_1 \cup \{t_1\} \subseteq V(G_1)$. Consider the bridge-block tree of G_1: because of (B1) there are no black leaves, and because of (B2) there are no two adjacent degree-two black nodes. This implies that there are at most $O(|X_1|)$ nodes in the bridge-block tree of G_1, and therefore $O(|X_1|)$ 2-edge-connected components and $O(|X_1|)$ bridges in G_1. This shows that G_1 satisfies properties (a), (b), and (d) of Problem 5.1. The following two lemmas show also that property (c) is satisfied, and therefore G_1 is a solution to Problem 5.1.

LEMMA 5.1. *Let $\widehat{G}$ be obtained from G_0 by applying either rule* (B1) *or rule* (B2). *Let H be given with $V(G_0) \cap V(H) \subseteq X_1$, and let x and y be any two vertices of $X_1 \cup \{t_1\} \cup V(H)$. For every $k \geq 2$, x and y are k-edge-connected in $\widehat{G} \cup H$ if and only if they are k-edge-connected in $G_0 \cup H$.*

Proof. We observe that neither (B1) nor (B2) can contract edges incident to vertices of $X_1 \cup \{t_1\}$. Since $V(G_0) \cap (V(H) \cup X_1 \cup \{t_1\}) \subseteq X_1 \cup \{t_1\}$, this implies that $\widehat{G} \cup H$ is obtained from $G_0 \cup H$ by contracting an edge that is not incident to either x or y. So if x and y are k-edge-connected in $G_0 \cup H$, they are k-edge-connected in $\widehat{G} \cup H$. Assume now that x and y are k-edge-connected in $\widehat{G} \cup H$. Then there are k edge-disjoint paths between x and y in $\widehat{G} \cup H$.

If (B1) was applied, $e = (u, v)$ is a bridge and B is a black 2-edge-connected component of degree one. Since B is black, it contains no vertex of $X_1 \cup \{t_1\}$. Furthermore, since $V(G_0) \cap V(H) \subseteq X_1$, no edge of H is incident to a vertex in B. This implies that neither x nor y is in B. After applying rule (B1), e is contracted, identifying vertices u and v into a new vertex w. Conversely, $G_0 \cup H$ can be obtained from $\widehat{G} \cup H$ by the inverse operation of splitting vertex w so as to reconstruct $e = (u, v)$. Since e is a bridge of G_0 separating B from the rest of the graph, and no edge of H is incident to B, e is a bridge of $G_0 \cup H$ too. This implies that after contracting edge e, w will be an articulation point of $\widehat{G} \cup H$, again separating B from the rest of the graph. Since neither x nor y is in B, there are k edge-disjoint simple paths in $\widehat{G} \cup H$ that do not contain edges of B. Thus, after splitting vertex w, the same k paths are edge-disjoint in $G_0 \cup H$, and therefore x and y are k-edge-connected in $G_0 \cup H$ too.

If (B2) was applied, B_1 and B_2 are black 2-edge-connected components of degree two in G_0. Since they are black, neither B_1 nor B_2 contains vertices of $X_1 \cup \{t_1\}$. Furthermore, since $V(G_0) \cap V(H) \subseteq X_1$, no edge of H is incident to a vertex in either B_1 or B_2. This implies that x and y are outside both B_1 and B_2. After applying rule (B2), e_2 is contracted and u_2 and v_2 are identified into a new vertex, say w_2.

Conversely, $G_0 \cup H$ can be obtained from $\widehat{G} \cup H$ by the inverse operation of splitting vertex w_2 into vertices u_2 and v_2 joined by edge e_2. Since no edge of H is incident to either B_1 or B_2, $\{e_1, e_3\}$ is a 2-edge-cut in $\widehat{G} \cup H$. Since neither x nor y is in $B_1 \cup B_2$, at most one of the k edge-disjoint paths in $\widehat{G} \cup H$ contains e_1 and e_3 and goes through B_1 and B_2. If w is split into vertices u_2 and v_2 joined by edge e_2, there is a new path containing edges e_1, e_2, and e_3 and going through B_1 and B_2 that is still edge-disjoint from the other $k-1$ paths: there are still k edge-disjoint paths between x and y in $G_0 \cup H$, and therefore x and y are k-edge-connected. □

The following corollary follows from Lemma 5.1.

COROLLARY 5.1. *Let $\widehat{G}$ be obtained from G_0 by applying either rule* (B1) *or rule* (B2). *For every $k \geq 2$, $\widehat{G}$ is a local certificate of k-edge-connectivity for X_1 in G_0.*

LEMMA 5.2. *Let $\widehat{G}$ be obtained from G_0 by applying either rule* (B1) *or rule* (B2). *Then for every $k \geq 2$, $\widehat{G}$ is a global certificate of k-edge-connectivity for X_1 in G_0.*

Proof. Let H be given with $V(G_0) \cap V(H) \subseteq X_1$. Rules (B1) and (B2) can only increase the edge connectivity of $G_0 \cup H$, and thus $\lambda(\widehat{G} \cup H) \geq \lambda(G_0 \cup H)$. To prove the lemma, it remains to show that if there is a k-edge-cut in $G_0 \cup H$, then in $\widehat{G} \cup H$ there must be an edge-cut of cardinality $k' \leq k$.

Let (S, T) be a k-edge-cut in $G_0 \cup H$. If both S and T contain vertices of $V(H) \cup X_1 \cup \{t_1\}$, a k-edge-cut in $\widehat{G} \cup H$ exists by Lemma 5.1. Otherwise, assume without loss of generality that $V(H) \cup X_1 \cup \{t_1\} \subseteq S$. This implies that T contains only black vertices of G_0 and that all the edges of (S, T) are in G_0: thus, (S, T) is an edge-cut separating X_1 in G_0. Let (S', T') be a minimum edge-cut separating X_1 from t_1 in G_0, with $X_1 \subseteq S'$ and $t_1 \in T'$, and let k' denote the cardinality of (S', T'). Recall that by the definition of t_1, (S', T') has the smallest cardinality among all of the edge-cuts separating X_1 in G_0: thus $k' \leq k$. Since $V(G_0) \cap V(H) \subseteq X_1 \subseteq S'$, $(S' \cup V(H), T')$ is itself a k'-edge-cut in $G_0 \cup H$. Since t_1 was colored red, a k'-edge-cut, $k' \leq k$, must exist in $\widehat{G} \cup H$ because of Lemma 5.1. □

It is clear from the proof of Lemma 5.2 why we needed to choose vertex t_1. Indeed, if G_0 has a bridge that leaves all the vertices of X_1 on one side, then $\lambda(G_0 \cup H) = 1$ for every H such that $V(H) \cap V(G_0) \subseteq X_1$. Picking vertex t_1 and coloring it red guarantees that there will be a bridge that leaves all the vertices of X_1 on one side in $\widehat{G} \cup H$ too.

Corollary 5.1 and Lemma 5.2 imply that G_1 is a full certificate for G. However, we remark that G_1 is not necessarily a *compressed* certificate, since it can have more than $O(|X_1|)$ vertices and edges: indeed, each 2-edge-connected component of G_1 might contain many vertices and edges. In the next section we show how to compress a 2-edge-connected graph, solving Problem 5.2.

5.2. Compressing a 2-edge-connected graph. We now turn to Problem 5.2. Let G_0 be a 2-edge-connected graph, and let $X_2 \subseteq V(G_0)$ be a set of vertices of G_0. As said before, if each 3-edge-connected class of G_0 is shrunk into one super-vertex, the resulting graph consists of a collection of simple cycles such that no two cycles share more than one super-vertex. This graph is the *cactus tree* of G_0: each node in the cactus tree corresponds to a 3-edge-connected class (and thus component) of G. Let p be the number of 3-edge-connected classes of G_0: the cactus tree has the property of having only $O(p)$ edges, and the edges in the cactus tree define all the possible 2-edge-cuts of G_0. Indeed, any two edges in the same cycle of the cactus tree define a 2-edge-cut of G_0. Consequently, even though the number of 2-edge-cuts can be $\Omega(p^2)$ (for instance, if G_0 is a simple cycle of p edges), the cactus tree of G_0 always

has size $O(p)$.

Let (S,T) be a minimum edge-cut separating X_2 in G_0, and without loss of generality, assume that $X_2 \subseteq S$. Pick arbitrarily one vertex $t_2 \in T$ and color red all the vertices of $X_2 \cup \{t_2\}$: note that the total number of red vertices of G_0 is $|X_2|+1$. Define a 3-edge-connected class of G_0 to be *red* if it contains at least one red vertex, and define it to be *black* otherwise. Clearly, there are $O(|X_2|)$ red 3-edge-connected classes. Our compression follows many of the same ideas used in section 5.1, but this time applies rules that are the analogs of rules (1) and (2) to the cactus tree of a 2-edge-connected graph rather than to the bridge-block tree of a connected graph. However, there are more technicalities involved, since the cactus tree here is not really a tree, but rather a tree of cycles.

Define the *degree of a 3-edge-connected class* to be the number of 2-edge-cuts incident to it (i.e., the number of cycles in the cactus tree that the corresponding node belongs to). If C is a 3-edge-connected class of G_0, there is a node in the cactus tree corresponding to C. Since there is no danger of ambiguity, we call this cactus tree node C also, and we color it with the same color we used for the 3-edge-connected class C.

Given a set S of vertices in G_0, define the graph $\Gamma_G(S)$ having as vertices S and all neighbors of S, and having as edges all edges in G_0 with one or both endpoints in S.

To compress "uninteresting leaves" of the cactus tree, we delete black nodes that appear in only one cycle. To compress "uninteresting chains," we take a chain of black nodes of the cactus tree that appear in the same 2-cycle and merge them into a smaller certificate for that chain. The two rules (analogs of rules (1) and (2)) that we use are the following.

(T1) Let C be a black 3-edge-connected class of degree one, and let $\{e_1, e_2\}$ be the 2-edge-cut incident to C. Let $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$, with u_1 and u_2 in C. Replace e_1, e_2, and C with an edge $e = (v_1, v_2)$.

(T2) Let $C_1, C_2, \ldots, C_\ell$ be a set of one or more black 3-edge-connected classes of degree two, such that each adjacent pair C_i, C_{i+1} shares an edge-cut $\{e_i, e_i'\}$. Let $\{e_0, e_0'\}$ and $\{e_\ell, e_\ell'\}$ be the two 2-edge-cuts separating $\bigcup C_i$ from the rest of the graph, and let $Y = \{v_0, v_0', v_\ell, v_\ell'\}$ be the endpoints of these cut edges in the rest of the graph. Let $B = \Gamma_G(C_1 \cup C_2 \cup \cdots \cup C_\ell)$; i.e., B consists of the edges induced by each C_i together with all cut edges e_i, e_i', $0 \le i \le \ell$. Then replace B by the minimum size planarity-preserving certificate B' of k-edge-connectivity (for all k) for B with the vertices of Y denoted as interesting.

The definition of rule (T2) may seem circular, as we are invoking the existence of certificates in the definition of an algorithm we are trying to use to prove that certificates exist. However, we know that there will always exist some planarity-preserving certificate: B itself is such a graph. Therefore, (T2) is well defined, although it may not be clear how to implement it efficiently. The reason for stating (T2) in this form is threefold: it is very general, and applies equally well to other forms of connectivity in other cactus trees; it motivates the seemingly more complicated rule (T2′) below, and it is immediately apparent that the result is a certificate. A general argument based on the possible connectivity requirements through vertices in Y, and on the different ways these vertices can be placed in an embedding of B, shows that only $O(1)$ different replacement certificates are needed in rule (T2), and hence B' has size $O(1)$. We do not elaborate, as this argument does not give good bounds on $|B'|$. Instead, we replace (T2) by rule (T2′) below. With rule (T2′), the size of B' is clearly

$O(1)$, but it is less clear that the resulting graph is a certificate. We prove this by showing that (T2) and (T2′) are equivalent.

Our description is based on *flows* in G. If we assign orientations and nonnegative *flow amounts* to the edges of G, the *excess* at any vertex is the sum of the flow amounts on incoming edges minus the sum of flow amounts on outgoing edges. A *flow* is an assignment of orientations and flow amounts such that, except at certain designated *terminals*, the excess at each vertex is zero. We will assume *integer flows* and *unit capacity edges*; i.e., all flow amounts should be zero or one. A *flow requirement* is the designation of values on certain terminal vertices; we say that a flow requirement is *satisfied* if there exists a flow such that, at each terminal, the excess equals the designated value. The *total flow* is the sum of positive flow requirement values; we call a flow with total flow k a *k-unit flow*. Any k-unit flow can be partitioned into k single-unit flows; a single-unit flow is just a path connecting its two terminals, possibly together with some cycles. This partitioning of a flow into paths forms one-half of the *max-flow min-cut theorem* [3], that x and y are k-connected if and only if there exists a flow (with integer edge capacities, and which can be restricted to an integer flow without loss of generality) with x and y as terminals having designated values $-k$ and k; for a proof of this connection between flows and connectivity, see any graph theory text.

First, we define the following term. Suppose that $\ell = 1$, so that B consists of the subgraph induced by a single black 3-edge-connected class C_1 together with the four vertices Y and the edges connecting Y to C_1. We say that B is *well connected* if, for any way of partitioning Y into two pairs of vertices, there is a two-unit flow in B with the first pair as the two sources and the second pair as the two sinks. For instance, if B consists of a star $K_{1,4}$, it is well connected. We are now ready to define rule (T2′).

(T2′) Let C_1, C_2, …, C_ℓ be a set of one or more black 3-edge-connected classes of degree two, such that each adjacent pair C_i, C_{i+1} shares an edge-cut $\{e_i, e_i'\}$. Let $\{e_0, e_0'\}$ and $\{e_\ell, e_\ell'\}$ be the two 2-edge-cuts separating $\bigcup C_i$ from the rest of the graph, and let $Y = \{v_0, v_0', v_\ell, v_\ell'\}$ be the endpoints of these cut edges in the rest of the graph. Let $B = \Gamma_G(C_1 \cup C_2 \cup \cdots \cup C_\ell)$.

- (i) If $\ell > 1$, form B' from B by contracting $C_1 \cup C_2 \cup \cdots \cup C_\ell$ to a single vertex.
- (ii) If $\ell = 1$, and C_1 is well connected, form B' from B by contracting C_1 to a single vertex.
- (iii) If $\ell = 1$, and C_1 is not well connected, let T be a spanning tree of B. (We show below that B is connected, so T exists.) Contract any edge in T adjacent to a vertex in C_1 of degree one or two, until no more contractions are possible; let B' be the resulting contracted tree.

In all cases replace B by B'.

LEMMA 5.3. *The graph B specified in rules* (T2) *and* (T2′) *is connected; moreover, the subgraph induced by each 3-edge-connected class C_i is connected.*

Proof. The connectedness of the subgraph induced by C_i follows because each pair of vertices in C_i is connected by three edge-disjoint paths, and only two such paths can pass through the four edges separating C_i from the rest of G. For each pair of classes C_i, C_{i+1} there is an edge connecting that pair, from which the overall connectedness of B follows. □

LEMMA 5.4. *Rule* (T2′) *preserves planarity.*

Proof. In all three cases of rule (T2′), the resulting graph B' is a minor of B and hence preserves planarity. □

We next show that the graph B' resulting from rule (T2′) is a certificate of k-edge-connectivity for B. A pair of vertices is k-edge-connected if and only if there is a k-unit flow with one vertex as source and the other as sink. If we consider the orientations and flow amounts of this flow, restricted to the edges of B, we can think of the excesses at each vertex of Y as giving flow requirements satisfied by this flow. (The excesses at vertices of $B - Y$ must be zero since all incident edges of such vertices are in B.) Replacing this flow on B by any other flow satisfying the same requirements will produce a k-unit flow in B, so it is enough to show that, whenever a flow requirement F with terminals in Y can be satisfied in B, it can also be satisfied in B'.

Note that we need only consider flow requirements of at most one unit at each vertex of Y, since each vertex of Y is adjacent to a single edge in B. Thus larger flow requirements could not be satisfied in B. So, overall, there are at most two single-unit sources and two single-unit sinks. If there is one source or sink only, the satisfiability of the flow requirements follows from the connectedness of B (Lemma 5.3) and from the connectedness of B' (obvious from the definition of rule (T2′)).

Let the remaining two-unit flow requirements be denoted F_1, having sources $\{v_0, v_0'\}$ and sinks $\{v_\ell, v_\ell'\}$; F_2, having sources $\{v_0, v_\ell\}$ and sinks $\{v_0', v_\ell'\}$; and F_3, having sources $\{v_0, v_\ell'\}$ and sinks $\{v_0', v_\ell\}$. (Any other collection of two sources and two sinks can be found by switching sources and sinks in some F_i.)

LEMMA 5.5. *Flow requirement F_1 is satisfiable in B.*

Proof. By assumption, G is 2-edge-connected, so there exists a pair of edge-disjoint paths in G from v_0 to v_ℓ. Each path must cross B, and the union of these two paths intersected with B forms a two-unit flow satisfying F_1. □

LEMMA 5.6. *At least one of F_2 or F_3 is satisfiable in B.*

Proof. Let F_1 be satisfied by a 2-unit flow. Then the flow can be partitioned into two edge-disjoint paths, each connecting a source with a sink. If the paths connect v_0 with v_ℓ and v_0' with v_ℓ', then F_3 is also satisfiable. If the paths connect v_0 with v_ℓ' and v_0' with v_ℓ, then F_2 is also satisfiable. □

LEMMA 5.7. *If some flow requirement F_i is not satisfiable by a flow in B, then B consists of the subgraph induced by a single class C_1.*

Proof. By Lemma 5.6, we can without loss of generality let the unsatisfiable flow requirement be F_2. If $\ell > 1$, this flow requirement would be satisfied by finding one path in the subgraph induced by C_1 connecting v_0 and v_0', and a path in the subgraph induced by C_ℓ connecting v_ℓ and v_ℓ', both of which exist by Lemma 5.3. □

LEMMA 5.8. *When part* (iii) *of rule* (T2′) *is applied to a graph B in which the flow requirement F_2 is not satisfiable by any flow, the resulting graph B' consists of Y together with two vertices u and u', with edges v_0u, uv_1, uu', $v_0'u'$, and $u'v_1'$.*

Proof. Since B' is a contraction of a tree, it is itself a tree. Since F_2 is unsatisfiable, B contains a bridge edge e between $\{v_0, v_1\}$ and $\{v_0', v_1'\}$ by the max-flow min-cut theorem [3]. This edge e must be in T. Construct a tree T' by contracting out all degree-one black vertices in T. Then B' remains a contraction of T'. Since e is on a path between two vertices of Y, it is not contracted in forming T'. On either side of e, T' consists of paths connecting v_0 with v_1, and v_0' with v_1'. Further, note that all vertices in Y must be leaves in T' since they have degree one in B and their degree does not increase in forming T'. The only compressed tree possible with four leaves and a bridge separating the two pairs of leaves is the one described in the lemma. □

LEMMA 5.9. *The graph B' resulting from rule* (T2′) *is a certificate of k-edge-*

connectivity for B.

Proof. We show that for every flow requirement F on terminals in Y, F is satisfiable by a flow in B if and only if it is satisfiable in B'. Requirements with more than two units of flow are never satisfiable in either graph. One-unit requirements are satisfiable in B by Lemma 5.3, and in B' since B' is clearly connected. Thus we need only worry about requirements F_1, F_2, and F_3.

If all three of the requirements F_i are satisfiable by flows in B, then either $\ell > 1$ or B consists of a single well-connected component C_1. In either case parts (i) or (ii) of rule (T2′) contract B to a star $K_{1,4}$ in which all three collections remain satisfiable.

If only two of the three requirements are satisfiable, then by Lemma 5.7, B consists of the subgraph induced by the single component C_1 which must not be well connected. By Lemma 5.6, we can assume without loss of generality that requirement F_2 is not satisfiable by a flow in B. Then by Lemma 5.8, B' consists of a tree with four leaves and a bridge, containing edge-disjoint paths from v_0 to v_1 and v_0' to v_1'; in this graph as in B, F_1 and F_3 are satisfiable and F_2 is not satisfiable. □

LEMMA 5.10. *Rule* (T2) *is equivalent to rule* (T2′).

Proof. The fact that B' is a certificate is shown in Lemma 5.9. It can be shown that it is the minimum certificate for B by noting that any certificate must contain a spanning tree with at least as many vertices as are in B'; we omit the details, as they are not necessary for the overall correctness of our connectivity algorithm. □

Next, we analyze the size of the cactus tree for G' and show that the graph obtained at the end of this compression is a solution to Problem 5.2. Recall that the meaning of our rules in the cactus tree is the following. Rule (T1) takes as input a black node that is contained in only one cycle and deletes it, while rule (T2) compresses a chain of black nodes in the cactus tree. The following lemmas show that the graph obtained at the end of this compression is a solution to Problem 5.2.

LEMMA 5.11. *Let G_2 be the graph obtained from a 2-edge-connected graph G_0 by repeated applications of rules* (T1) *and* (T2)*, until no further rule can be applied. Then there are $O(|X_2|)$ 3-edge-connected components in G_2, and its cactus tree contains $O(|X_2|)$ nodes and edges.*

Proof. Since rules (T1) and (T2) can be described in terms of the cactus tree, and each node of the cactus tree corresponds to a 3-edge-connected class (and thus component), we refer to nodes of the cactus tree as components, and show that when no rule can be applied any more, we are left with a cactus tree that contains $O(|X_2|)$ nodes and edges.

We observe that the cactus tree can be represented by a tree. We form a black or red node for every component in the cactus tree, and a blue node for every cycle in the cactus tree. A node representing a component is adjacent to a node representing a cycle if and only if the cycle contains that component.

Because of rule (T1), a black component must be shared by more than one cycle in the cactus tree. Therefore, all leaves of the tree are red, and there are at most $|X_2| + 1$ leaves.

Next, we examine the degree-two nodes in the tree. Form maximal chains of black and blue degree-two nodes. Each such chain is terminated either by a high-degree node or by a red node; therefore, as in any tree, there can be at most $2|X_2| - 1$ chains. Each chain can consist of at most one black node and two blue nodes by rule (T2), so there are at most $2|X_2| - 1$ black nodes of degree two and at most $4|X_2| - 2$ blue nodes of degree two.

Finally, as in any tree, there are at most two fewer nodes of degrees higher than

two than there are leaves, so there are at most $|X_2| - 1$ such nodes.

In total we find at most $3|X_2| - 2$ black nodes representing black components of the cactus tree, and at most $5|X_2| - 3$ blue nodes representing cycles in the cactus tree. Any cactus tree with k components has at most $2k - 2$ edges, so there are at most $8|X_2| - 4$ edges in the cactus tree. ∎

LEMMA 5.12. *Let $\widehat{G}$ be obtained from G_0 by repeatedly applying rules* (T1) *and* (T2). *Let H be given with $V(G_0) \cap V(H) \subseteq X_2$. Any two vertices of $X_2 \cup V(H) \cup \{t_2\}$ are k-edge-connected in $\widehat{G} \cup H$ if and only if they are k-edge-connected in $G_0 \cup H$, $k \geq 2$.*

Proof. By transitivity of certificates, we need only show that each step produces a certificate for the previous graph. Steps involving rule (T1) yield a split graph with respect to the 2-edge-cut $\{e_1, e_2\}$, and Lemma 4.5 shows that this produces a certificate of k-edge-connectivity. Steps involving rule (T2) yield a certificate by definition. ∎

Lemma 5.12 implies the following corollary.

COROLLARY 5.2. *Let $\widehat{G}$ be obtained from G_0 by repeatedly applying rules* (T1) *and* (T2). *For every $k \geq 2$, $\widehat{G}$ is a local certificate of k-edge-connectivity for X_2 in G_0.*

LEMMA 5.13. *Let $\widehat{G}$ be obtained from G_0 by applying rule* (T1) *or* (T2). *Then $\widehat{G}$ is a global certificate of k-edge-connectivity for X_2 in G_0, $k \geq 2$.*

Proof. Let H be given with $V(G_0) \cap V(H) \subseteq X_2$. Rules (T1) and (T2) can only increase the connectivity of $G_0 \cup H$, and thus $\lambda(\widehat{G} \cup H) \geq \lambda(G_0 \cup H)$. Hence, it remains to show that any k-edge-cut (S, T) in $G_0 \cup H$ corresponds to an edge-cut in $\widehat{G} \cup H$ of cardinality at most k.

If both S and T contain vertices in $V(H) \cup X_2 \cup \{t_2\}$, a corresponding edge-cut can be found by Lemma 5.12. Otherwise assume without loss of generality that T contains only black vertices of G_0. Then all edges of cut (S, T) are in G_0. Let (S', T') be the minimum edge-cut separating X_2 from t_2 in G_0. Then $(S' \cup V(H), T')$ has edge cardinality at most that of (S, T), and since t_2 was colored red, a corresponding edge-cut exists in $\widehat{G} \cup H$ by Lemma 5.12. ∎

5.3. Compressing a 3-edge-connected graph. In this section we describe our solution to Problem 5.3. Since it relies heavily on the notion of cactus tree, we first list some properties of a cactus tree of a 3-edge-connected graph, referring the interested reader to [4, 28] for the full details. It might be helpful to refer to Figure 2 while we discuss these properties.

Let G_0 be a 3-edge-connected graph, and let $\mathcal{T}(G_0)$ denote its cactus tree. Since 3 is odd, $\mathcal{T}(G_0)$ is an actual tree (and not a tree of edges and cycles). Edges of $\mathcal{T}(G_0)$ correspond to 3-edge-cuts of G_0. Each 4-edge-connected class of G_0 corresponds to a node in $\mathcal{T}(G_0)$, while the converse is not necessarily true. Indeed, there can be nodes that do not correspond to any 4-edge-connected class: the reason for having these nodes is to represent the tree structure of the 3-edge-cuts (see Figure 2). We call these nodes *empty* since they do not contain any vertex of G_0. In what follows, since there is no danger of ambiguity, we will use interchangeably the terms 3-edge-cut of G_0 and edge of $\mathcal{T}(G_0)$, and the terms 4-edge-connected class of G_0 and nonempty node of $\mathcal{T}(G_0)$. Note that an edge $e = (u, v)$ of G_0 might take part in different 3-edge-cuts of G_0. However, all these 3-edge-cuts must form a path in $\mathcal{T}(G_0)$: endpoints of this path are the two nodes corresponding to the 4-edge-connected classes containing u and v. We label each cactus edge with the three edges of the corresponding 3-edge-cut.

We now describe our solution to Problem 5.3. Let G_0 be a 3-edge-connected graph, and let $X_3 \subseteq V(G_0)$ and $Y_3 \subseteq E(G_0)$. Let Z_3 be the set of endpoints of edges in Y_3. We color red the vertices of $X_3 \cup Z_3$ and the edges of Y_3. Note that both the endpoints of a red edge are colored red. We define a 3-edge-cut of G_0 to be an *inter* 3-*edge-cut* if it separates two different red vertices of G_0, and define it to be an *extra* 3-*edge-cut* otherwise. If G_0 has at least one *extra* 3-edge-cut, we pick one such 3-edge-cut and we choose arbitrarily one vertex that is separated from all the red vertices by this 3-edge-cut. We call this vertex the *chosen extra vertex*, and we call the 3-edge-cut the *chosen extra* 3-*edge-cut*. We color the chosen extra vertex red: the total number of red vertices of G_0 is therefore $(|X_3 \cup Z_3| + 1)$. Our computation of a solution for Problem 5.3 consists of the following three phases.

`Phase 1:` Compute the cactus tree $\mathcal{T}(G_0)$ of G_0 (see Figure 4(b)).

`Phase 2:` Compute a new cactus tree $\mathcal{T}'$ by compressing $\mathcal{T}(G_0)$, so that $\mathcal{T}'$ will have size linear in the number of red vertices (see Figure 4(c)).

`Phase 3:` Compute G_3 from $\mathcal{T}'$.

We now give the low-level details of these phases. Phase 1 is simply accomplished by computing the cactus tree of G_0. Let R be the set of red vertices in G_0, and let R_1 and R_2 be any two nonempty sets of red vertices in G_0, $R_1 \cap R_2 = \emptyset$. We say that an edge of $\mathcal{T}(G_0)$ separates R_1 and R_2 if it corresponds to a 3-edge-cut separating R_1 and R_2 in G_0. The following lemma is an immediate consequence of the notion of cactus tree [4].

LEMMA 5.14. *Let R_1 and R_2 be any two nonempty sets of red vertices in G_0, $R_1 \cap R_2 = \emptyset$. The* 3-*edge-cut $\{e_1, e_2, e_3\}$ of G_0 separates R_1 and R_2 if and only if the edge (α, β) labeled with $\{e_1, e_2, e_3\}$ in $\mathcal{T}(G_0)$ separates R_1 and R_2.*

In Phase 2 we use a compression similar to the one used for the MSF (minimum spanning forest) certificate. Given the graph G_0 and its cactus tree $\mathcal{T}(G_0)$, we define the *degree of a* 4-*edge-connected class* of G_0 to be the tree degree of the corresponding (nonempty) node in $\mathcal{T}(G_0)$. Intuitively, the degree of a 4-edge-connected class denotes the number of different 3-edge-cuts that are "incident" to the class. Again, we define a 4-edge-connected class to be *red* if it contains at least one red vertex, and define it to be *black* otherwise. We color red the (nonempty) nodes of $\mathcal{T}(G_0)$ that correspond to red 4-edge-connected classes of G_0 (see Figure 4(b)). Black 4-edge-connected classes of degree one are uninteresting leaves in the cactus tree, and adjacent black nodes of degree two give rise to uninteresting chains in the cactus tree. Our rules to compress $\mathcal{T}(G_0)$ are the following.

(Q1) Let Q be a black cactus node of degree 1. Delete Q and its incident edge from the cactus tree.

(Q2) Let Q be a black cactus node of degree 2. Let e_1 and e_2 be the cactus edges incident to Q, and let Q_1 and Q_2 be the cactus nodes adjacent to Q. Delete Q and replace e_1 and e_2 with an edge e between Q_1 and Q_2. Edge e gets the same label as e_1.

We call $\mathcal{T}'$ the cactus tree obtained after all the rules (Q1) and (Q2) have been applied. Note that $\mathcal{T}'$ contains all the red nodes of $\mathcal{T}(G_0)$ and may contain some black nodes of $\mathcal{T}(G_0)$. However, because of (Q1) and (Q2), the total size of $\mathcal{T}'$ is linear in the number of red nodes. Note that edges of $\mathcal{T}'$ are also edges of $\mathcal{T}(G_0)$, and therefore correspond to 3-edge-cuts of G_0. Again, given two nonempty disjoint sets of red vertices R_1 and R_2, we say that an edge of $\mathcal{T}'$ separates R_1 and R_2 if it corresponds to a 3-edge-cut separating R_1 and R_2 in G_0. As the following lemma shows, $\mathcal{T}'$ has the property of preserving 3-edge-cuts separating any two sets of red

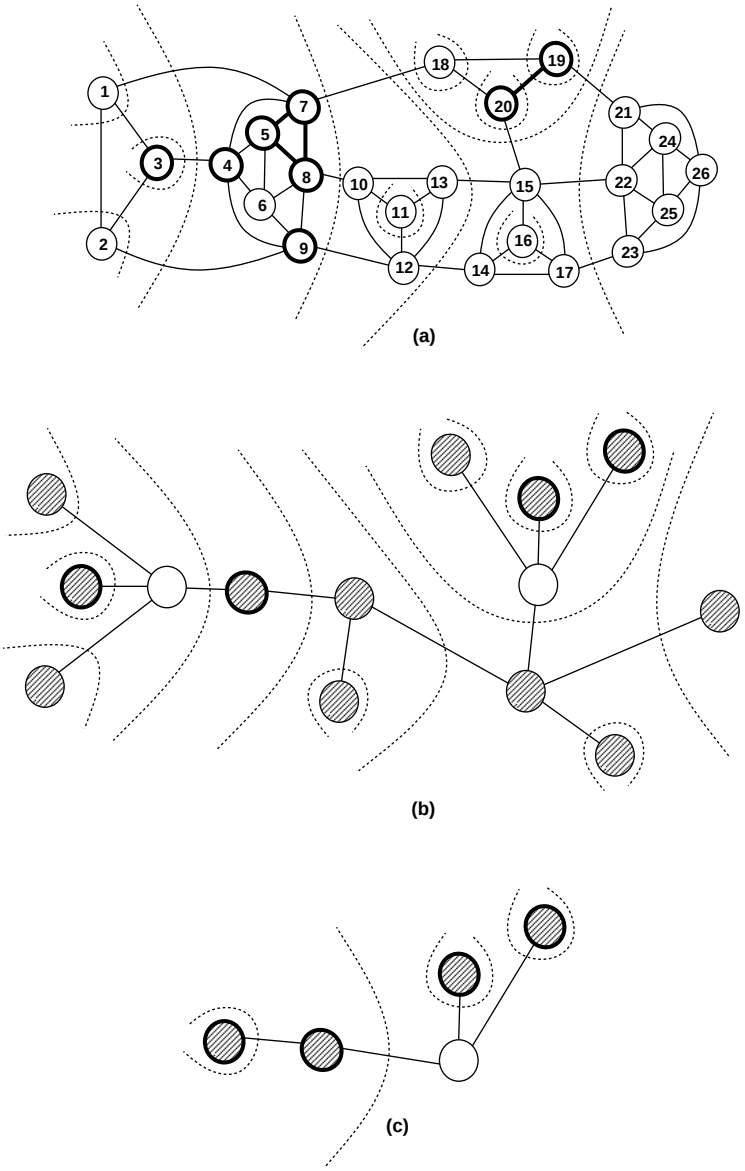


FIG. 4. *Compressing a 3-edge-connected graph G_0.* (a) *The original graph G_0: red vertices ($\{3,4,5,7,8,9,19,20\}$) and red edges ($\{(5,7),(5,8),(7,8),(19,20)\}$) are shown in bold, 3-edge-cuts are dashed.* (b) *The cactus tree $\mathcal{T}(G_0)$: red 4-edge-connected classes are shown in bold.* (c) *The compressed cactus tree $\mathcal{T}'$ obtained from $\mathcal{T}(G_0)$.*

vertices in G_0.

LEMMA 5.15. *Let R_1 and R_2 be any two nonempty sets of red vertices in G_0, $R_1 \cap R_2 \neq \emptyset$. There is an edge (α, β) in $\mathcal{T}(G_0)$ separating R_1 and R_2 if and only if there is an edge (α', β') in $\mathcal{T}'$ separating R_1 and R_2.*

Proof. To prove the lemma, we show that rules (Q1) and (Q2) maintain the following invariant. There is an edge (α, β) separating R_1 and R_2 before the rule is

applied if and only if there is an edge (α', β') separating R_1 and R_2 after the rule is applied.

If rule (Q1) is applied, Q is a black node. Since the edge incident to Q cannot separate red vertices, the invariant must hold. If rule (Q2) is applied, Q is a black node. Consequently, e_1 and e_2 separate exactly the same set of red vertices. But then the edge e inserted by rule (Q2) still separates the same set of red vertices previously separated by e_1 and e_2. Hence, the invariant still holds. □

Now, we describe our implementation of Phase 3, namely, how to compute from $\mathcal{T}'$ the graph G_3 that is a solution to Problem 5.3. By Lemmas 5.14 and 5.15, we know that, given any two sets R_1 and R_2 in G_0, they can be separated by a 3-edge-cut of G_0 if and only if they can be separated by an edge of $\mathcal{T}'$. In Phase 3, we build a graph G_3 that contains the red vertices and red edges of G_0, and has all the 3-edge-cuts corresponding to edges of $\mathcal{T}'$. We will use this property to prove that any two sets of red vertices R_1 and R_2 can be separated by a 3-edge-cut in G_0 if and only if they can be separated by a 3-edge-cut in G_3.

We show how to obtain G_3. For each edge (α', β') of $\mathcal{T}'$, we do the following. Let $\{e_1, e_2, e_3\}$ be the label of (α', β') (corresponding to the 3-edge-cut $\{e_1, e_2, e_3\}$ of G_0): we mark the edges e_1, e_2, and e_3 and their endpoints in G_0. In other words, all the edges that are in a 3-edge-cut corresponding to an edge of $\mathcal{T}'$ are *marked*. Next, we delete all the marked edges from G_0, and denote by G_0^1, G_0^2, ..., G_0^p, $p \geq 1$, the connected components left (see Figure 5(a)). The edges of G_0^i, $1 \leq i \leq p$, are referred to as *unmarked*.

Our graph G_3 will keep all the marked edges and replace each G_0^i with a smaller graph that preserves 3- and 4-edge-connectivity between the red vertices. We compute this smaller graph as follows. We denote the red and marked vertices of G_0^i *interesting*. We force 4-edge-connectedness between any two interesting vertices in G_0^i by replacing G_0^i with a smaller graph that contains all the interesting vertices and the red edges originally in G_0^i and that is 4-edge-connected. We call this graph $\mathcal{C}(G_0^i)$. Note that this might change the edge connectivity inside G_0^i, since it may change the number of edge-disjoint paths inside G_0^i. Namely, two vertices x and y of G_0^i may be ℓ-edge-connected, $\ell \geq 4$ in G_0, and ℓ'-edge-connected, $\ell' \geq 4$, $\ell' \neq \ell$, in the new graph obtained after replacing G_0^i with $\mathcal{C}(G_0^i)$. However, as we will show, this is not a problem, since we are interested in certificates for 3- and 4-edge-connectivity only, and not for higher edge connectivity. We make all the interesting vertices of G_0^i 4-edge-connected in the new graph by compressing the unmarked edges as follows. We first compute the minimal subgraph S_0^i of G_0^i that is connected and contains all the red edges and the interesting vertices in G_0^i (see Figure 5(b)). Note that S_0^i is not necessarily a tree, since it might contain cycles: however, because of the minimality of S_0^i, the possible cycles of S_0^i can have only red edges. We collapse each red cycle in a single node, and then compress the tree obtained, using as interesting vertices the collapsed cycles, the red vertices and the marked vertices. The compression of the unmarked edges left in S_0^i is the same as used in the MSF algorithm: we compress uninteresting branches and uninteresting chains of degree-two vertices (see Figure 5(c)). Next, we expand the red cycles back, and finally, we quadruplicate each unmarked edge left in the compressed graph, to force 4-edge-connectivity. We call the graph obtained $\mathcal{C}(G_0^i)$, and we call G_3 the graph obtained after replacing G_0^i with $\mathcal{C}(G_0^i)$, $1 \leq i \leq p$ (see Figure 5(d)).

Note that G_3 is obtained from G_0 by means of a suitable sequence of the following operations:

(1) delete unmarked edges inside a component G_0^i; and

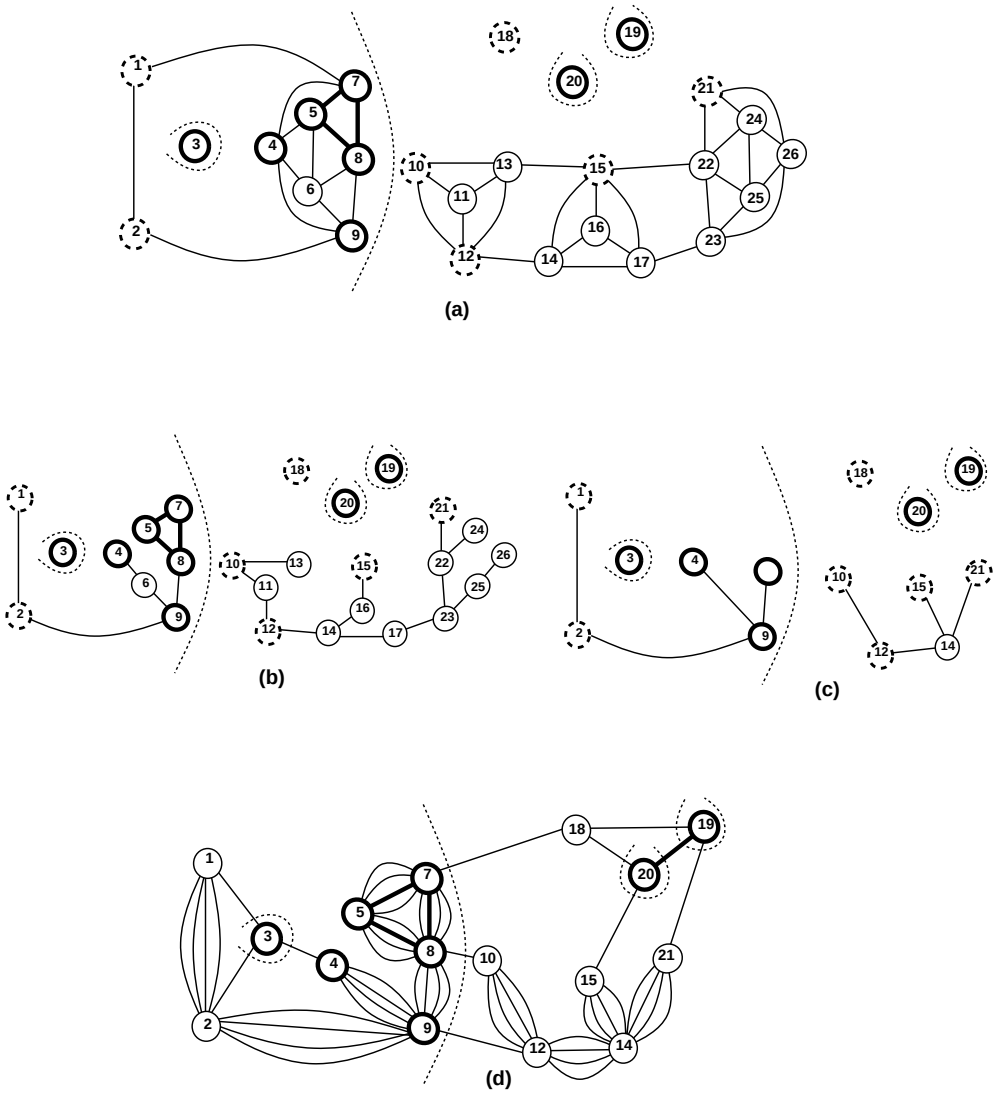


FIG. 5. *Compressing G_0; 3-edge-cuts corresponding to edges of $\mathcal{T}'$ are always shown dashed.* (a) *The connected components G_0^i, $1 \le i \le p$, obtained after the removal of 3-edge-cuts in $\mathcal{T}'$. Marked (non-red) vertices are shown dashed.* (b) *The minimal subgraphs S_0^i, $1 \le i \le p$.* (c) *The graphs obtained after compressing S_0^i, $1 \le i \le p$: the red cycle containing vertices* 5, 7, *and* 8 *has been collapsed into a single node.* (d) *The graph G_3 obtained from G_0 by replacing G_0^i with $\mathcal{C}(G_0^i)$, $1 \le i \le p$. The red cycle has been expanded.*

(2) contract unmarked edges; and
(3) quadruplicate unmarked edges.

Conversely, G_0 can be recomputed from G_3 by means of a suitable sequence of the following operations:

(4) delete multiple unmarked edges;
(5) expand unmarked edges;

(6) insert new unmarked edges inside a component $\mathcal{C}(G_0^i)$.

We now prove that G_3 is a solution to Problem 5.3.

LEMMA 5.16. *G_3 is 3-edge-connected, and $X_3 \subseteq V(G_3)$ and $Y_3 \subseteq E(G_3)$.*

Proof. We first prove that G_3 is 3-edge-connected. Assume by contradiction that the minimum edge-cut γ of G_3 is of cardinality ℓ, $\ell \leq 2$. Recall that G_3 consists of marked edges and unmarked edges. Since each unmarked edge is quadruplicated in G_3, its endpoints are 4-edge-connected. This implies that no unmarked edge can be in γ, and therefore, γ consists of marked edges only. Due to the minimality of γ, deleting the edges of γ disconnects G_3 into two graphs: say G_3' and G_3''. Since γ contains only marked edges, and each $\mathcal{C}(G_0^i)$ is connected, each $\mathcal{C}(G_0^i)$ is contained in either G_3' or G_3''. Recall that G_0 can be obtained from G_3 by means of operations of type (4), (5), and (6) above. Each such operation modifies only the graphs $\mathcal{C}(G_0^i)$ by deleting multiple edges, expanding unmarked edges, and inserting new unmarked edges. Marked edges will be unaffected by these operations. This implies that all these operations will be local to the graphs $\mathcal{C}(G_0^i)$, and therefore will still keep the vertices in G_3' and in G_3'' disconnected. As a result, γ is an ℓ-edge-cut in G_0. This is clearly a contradiction, since $\ell \leq 2$ and G_0 is 3-edge-connected. Thus, there cannot be an ℓ-edge-cut, $\ell \leq 2$, in G_3, and therefore G_3 must be 3-edge-connected.

We now prove that G_3 contains all the red vertices and edges of G_0. Since red vertices are kept in each $\mathcal{C}(G_0^i)$, we have that $X_3 \subseteq V(G_3)$. As for red edges, recall that the edges of G_0 are partitioned into marked and unmarked. Each $\mathcal{C}(G_0^i)$ preserves all the red edges in G_0^i: this shows that all the unmarked red edges are in G_3. Since G_3 preserves all the marked edges, it will preserve also the marked red edges. This shows that $Y_3 \subseteq E(G_3)$. ∎

Before proving the other properties that make G_3 a solution to Problem 5.3, we need some technical lemmas.

LEMMA 5.17. *Let R_1 and R_2 be any two nonempty sets of red vertices in G_0, $R_1 \cap R_2 = \emptyset$. There is a 3-edge-cut in G_0 separating R_1 and R_2 if and only if there is a 3-edge-cut in G_3 separating R_1 and R_2.*

Proof. Assume that there is a 3-edge-cut $\{e_1, e_2, e_3\}$ separating R_1 and R_2 in G_0. By Lemmas 5.14 and 5.15 there must be an edge (α', β') in $\mathcal{T}'$ separating R_1 and R_2. Let $\{e_1', e_2', e_3'\}$ be the label of (α', β'). We claim that $\{e_1', e_2', e_3'\}$ is a 3-edge-cut of G_3 separating R_1 and R_2. Note that, as a consequence of our computation of $\mathcal{T}'$, $\{e_1', e_2', e_3'\}$ must have been the label of one edge of $\mathcal{T}(G_0)$. By definition of cactus tree, this implies that $\{e_1', e_2', e_3'\}$ is a 3-edge-cut of G_0. Note that it is not necessarily the case that $\{e_1, e_2, e_3\} = \{e_1', e_2', e_3'\}$. Both $\{e_1, e_2, e_3\}$ and $\{e_1', e_2', e_3'\}$ are 3-edge-cuts separating R_1 and R_2 in G_0, and $\{e_1', e_2', e_3'\}$ is kept in $\mathcal{T}'$ (and therefore in G_3).

Denote by G_0' and G_0'' the graphs obtained from G_0 after removing e_1', e_2', and e_3', and without loss of generality, assume that $R_1 \subseteq V(G_0')$ and $R_2 \subseteq V(G_0'')$. Note that $\{e_1', e_2', e_3'\}$ is a 3-edge-cut corresponding to an edge of $\mathcal{T}'$. As a result, the edges e_1', e_2', and e_3' are marked edges, and each G_0^i is contained in either G_0' or G_0''. Recall that G_3 is obtained from G_0 by means of operations (1)–(3) above, which are local to the graphs G_0^i and therefore will still keep G_0' and G_0'' disconnected. Since $R_1 \subseteq V(G_0')$ and $R_2 \subseteq V(G_0'')$, this implies that $\{e_1', e_2', e_3'\}$ is a 3-edge-cut separating R_1 and R_2 in G_3.

Assume now that there is a 3-edge-cut $\{e_1, e_2, e_3\}$ separating R_1 and R_2 in G_3. Note that no unmarked edge can be in a 3-edge-cut, since each unmarked edge left after the compression of S_0^i is quadruplicated, therefore making its endpoints 4-edge-connected. As a result, e_1, e_2, and e_3 must all be marked edges. This implies that

e_1, e_2 and e_3 are all edges of G_0.

Since by Lemma 5.16 G_3 is 3-edge-connected, $\{e_1, e_2, e_3\}$ is a minimum edge-cut in G_3, and therefore the removal of e_1, e_2, and e_3 splits G_3 into two graphs, say G_3' and G_3''. Without loss of generality assume that $R_1 \subseteq V(G_3')$ and $R_2 \subseteq V(G_3'')$. Recall that G_0 can be obtained from G_3 by means of operations (4)–(6) above, which still keep G_3' and G_3'' disconnected. Since $R_1 \subseteq V(G_3')$ and $R_2 \subseteq V(G_3'')$, this implies that $\{e_1, e_2, e_3\}$ is a 3-edge-cut separating R_1 and R_2 in G_0. □

LEMMA 5.18. *Let γ be a 3-edge-cut of G_3, separating V' and V'', with $V', V'' \subseteq V(G_3)$. Then γ is a 3-edge-cut of G_0, separating V' and V'' in G_0.*

Proof. Let G_3' and G_3'' be the two graphs obtained after the deletion of γ from G_3, with $V' = V(G_3')$ and $V'' = V(G_3'')$. Recall that G_0 can be obtained from G_3 by means of operations of types (4), (5), and (6). None of these operations adds a path between V' and V'', and therefore γ is a 3-edge-cut of G_0, separating V' and V''. □

LEMMA 5.19. *G_3 has $O(|X_3 \cup Z_3|)$ vertices and edges.*

Proof. As said before, there are exactly $(|X_3 \cup Z_3| + 1)$ red vertices in G_0. Recall that a 4-edge-connected class Q_j of G_0 is red if it contains at least one red vertex. Let $r_j \geq 1$ be the number of red vertices contained in a red 4-edge-connected class Q_j, and let ρ be the total number of red 4-edge-connected classes of G_0. Since any two 4-edge-connected classes are disjoint, we have

$$\sum_{j=1}^{\rho} r_j = |X_3 \cup Z_3| + 1$$

and

$$\rho \leq |X_3 \cup Z_3| + 1.$$

Let $\mathcal{T}(G_0)$ be the cactus tree of G_0. We color red a node of $\mathcal{T}(G_0)$ only if it corresponds to a red 4-edge-connected class of G_0. Hence, there are exactly ρ red nodes in $\mathcal{T}(G_0)$. Let $\mathcal{T}'$ be the compressed cactus tree. Because of rules (Q1) and (Q2), $\mathcal{T}'$ has no black leaves and no black chains of degree-two nodes. Consequently, $\mathcal{T}'$ has $O(\rho)$ nodes and edges. Since there are at most three marked edges in G_0 for each edge in $\mathcal{T}'$, the total number of marked edges and vertices in G_0 will be $O(\rho)$. For $1 \leq i \leq p$, let ρ_i be the total number of marked vertices in G_0^i, and let σ_i be the total number of red vertices in G_0^i. Note that

$$\sum_{i=1}^{p} (\rho_i + \sigma_i) = O(\rho + |X_3 \cup Z_3|) = O(|X_3 \cup Z_3|).$$

Since each $\mathcal{C}(G_i)$ has size $O(\rho_i + \sigma_i)$, and there are $O(\rho)$ marked edges in G_3, the lemma follows. □

LEMMA 5.20. *Let H be a graph such that $V(H) \cap V(G_0) \subseteq X_3 \cup Z_3$. If $\lambda(G_0 \cup H) = \ell$, $\ell \leq 3$, then $\lambda(G_3 \cup H) \leq \ell$.*

Proof. Assume that $\lambda(G_0 \cup H) = \ell \leq 3$, and let γ be a minimum edge-cut of $G_0 \cup H$, with $|\gamma| = \ell$. Let $\gamma_{G_0} = \gamma \cap E(G_0)$ and $\gamma_H = \gamma \cap E(H)$ be, respectively, the edges of γ in G_0 and in H. Due to the minimality of γ, if $\gamma_{G_0} \neq \emptyset$, γ_{G_0} must be an edge-cut in G_0. Similarly, if $\gamma_H \neq \emptyset$, γ_H must be an edge-cut in H. In other words, denote by $G_0' \cup H'$ and $G_0'' \cup H''$ the two graphs obtained from $G_0 \cup H$ after deleting the edges of γ, with $G_0', G_0'' \subseteq G_0$, and $H', H'' \subseteq H$: γ_{G_0} splits G_0 into G_0'

and G_0'', while γ_H splits H into H' and H''. If $\gamma_{G_0} = \emptyset$, then γ uses no edges of G_0, and therefore it does not split G_0: in this case we assume that $G_0' = G_0$ and $G_0'' = \emptyset$. Since γ_{G_0} is an edge-cut of G_0, and G_0 is 3-edge-connected, either $\gamma_{G_0} = \emptyset$ or $|\gamma_{G_0}| \geq 3$. Since $|\gamma| \leq 3$, we are left with only two possibilities: either

(a) $\gamma_{G_0} = \emptyset$ (and therefore $\gamma = \gamma_H$), or
(b) $|\gamma_{G_0}| = 3$ and $\gamma_H = \emptyset$ (and therefore $\gamma = \gamma_{G_0}$ and $\ell = 3$).

In case (a), since $\gamma_{G_0} = \emptyset$, $G_0 = G_0'$. Since $\gamma = \gamma_H$, no path between H' and H'' can go through G_0. Since G_0 is 3-edge-connected, and therefore connected, this implies that either H' or H'' is not connected to G_0: without loss of generality, assume that $V(H') \cap V(G_0) = \emptyset$. Then $\gamma = \gamma_H$ splits $G_0 \cup H$ into $G_0 \cup H''$ and H'. Since $V(H) \cap V(G_0) = V(H) \cap V(G_3)$, we also have that $V(H') \cap V(G_3) = \emptyset$, and therefore $\gamma = \gamma_H$ splits $G_3 \cup H$ into $G_3 \cup H''$ and H'. Thus, $\lambda(G_3 \cup H) \leq \ell$, and the lemma holds in case (a).

Assume now that we are in case (b) and $\gamma = \gamma_{G_0}$ is a 3-edge-cut in $G_0 \cup H$. Since $\ell = 3$, we have to prove that $\lambda(G_3 \cup H) \leq 3$. To do this, we distinguish two subcases.

(b1) γ_{G_0} is separating two nonempty sets of red vertices, say R_1 and R_2, in G_0, or
(b2) no two red vertices are separated by γ_{G_0}.

Assume we are in case (b1). Since $\gamma_H = \emptyset$, and γ_{G_0} is a 3-edge-cut of $G_0 \cup H$ separating R_1 and R_2, there is no path in H between R_1 and R_2. As a consequence of $R_1, R_2 \neq \emptyset$, by Lemma 5.17 there is a 3-edge-cut γ' separating R_1 and R_2 in G_3. Since γ' separates R_1 and R_2 in G_3, with $R_1, R_2 \neq \emptyset$, and there is no path in H between R_1 and R_2, γ' is a 3-edge-cut of $G_3 \cup H$. If we are in case (b2), $\gamma = \gamma_{G_0}$ is a 3-edge-cut of G_0 that does not separate red vertices. Since all the vertices in $X_3 \cup Z_3$ are red, this implies that γ is an *extra* 3-edge-cut. But then there is a chosen extra 3-edge-cut $\widehat{\gamma}$ in G_0. Let v_0 be the chosen extra vertex: $\widehat{\gamma}$ separates $\widehat{R_1} = X_3 \cup Z_3$ from $\widehat{R_2} = \{v_0\}$, and v_0 is colored red. By Lemma 5.17, there is a 3-edge-cut $\widehat{\gamma}'$ separating $X_3 \cup Z_3$ and $\{v_0\}$ in G_3. Since $V(H) \cap V(G_0) \subseteq X_3 \cup Z_3$, and $v_0 \notin (X_3 \cup Z_3)$, there cannot be a path in H between $\widehat{R_1} = X_3 \cup Z_3$ and $\widehat{R_2} = \{v_0\}$. This implies that $\widehat{\gamma}'$ separates $X_3 \cup Z_3$ and v_0 in $G_3 \cup H$, and therefore $\widehat{\gamma}'$ is a 3-edge-cut of $G_3 \cup H$. In summary, in both cases (b1) and (b2), there is a 3-edge-cut in $G_3 \cup H$. Then, $\lambda(G_3 \cup H) \leq 3$, and the lemma also holds in case (b), since $\ell = 3$. □

LEMMA 5.21. *Let H be a graph such that $V(H) \cap V(G_0) \subseteq X_3 \cup Z_3$. If $\lambda(G_3 \cup H) = \ell \leq 3$, then $\lambda(G_0 \cup H) \leq \ell$.*

Proof. We use an argument similar to the one used in the proof of Lemma 5.20. Assume that $\lambda(G_3 \cup H) = \ell \leq 3$, and let δ be a minimum edge-cut of $G_3 \cup H$, with $|\delta| = \ell$. Let $\delta_{G_3} = \delta \cap E(G_3)$ and $\delta_H = \delta \cap E(H)$ be the edges of δ in G_3 and in H. Due to the minimality of δ, if $\delta_{G_3} \neq \emptyset$, δ_{G_3} must be an edge-cut in G_3. Similarly, if $\delta_H \neq \emptyset$, δ_H must be an edge-cut in H. Since by Lemma 5.16, G_3 is 3-edge-connected, either $\delta_{G_3} = \emptyset$ or $|\delta_{G_3}| \geq 3$. As in the proof of Lemma 5.20, $|\delta| \leq 3$ leaves only two possibilities: either (a) $\delta_{G_3} = \emptyset$ or (b) $|\delta_{G_3}| = 3$, $\delta_H = \emptyset$, and $\ell = 3$. In case (a), the same proof previously used for case (a) of Lemma 5.20 shows that $\delta = \delta_H$ is an ℓ-edge-cut in $G_0 \cup H$ too. If we are in case (b), $\delta_H = \emptyset$, and $\delta = \delta_{G_3}$. Let V_1 and V_2 be the two sets of vertices separated by δ in G_3. Because of Lemma 5.18, δ is a 3-edge-cut separating V_1 and V_2 in G_0, which implies that all the paths between V_1 and V_2 in G_0 go through δ. Since $\delta_H = \emptyset$, there is no path in H between V_1 and V_2. The latter two statements imply that all the paths between V_1 and V_2 in $G_0 \cup H$ go through δ. Thus, δ is a 3-edge-cut of $G_0 \cup H$ too. □

LEMMA 5.22. *For $2 \leq k \leq 4$, G_3 is a compressed global certificate of k-edge-connectivity for $(X_3 \cup Z_3)$ in G_0.*

Proof. Let H be any graph with $V(H) \cap V(G_0) \subseteq X_3 \cup Z_3$. By Lemmas 5.20 and 5.21, $\lambda(G_0 \cup H) = \ell \leq 3$ if and only if $\lambda(G_3 \cup H) = \ell \leq 3$. This implies that $\lambda(G_0 \cup H) > 3$ if and only if $\lambda(G_3 \cup H) > 3$. Hence, G_3 is a global certificate of k-edge-connectivity for $(X_3 \cup Z_3)$ in G_0, $2 \leq k \leq 4$. This certificate is compressed because of Lemma 5.19. ∎

LEMMA 5.23. *Let H be given with $V(G_0) \cap V(H) \subseteq X_3 \cup Z_3$, and let $u \in X_3 \cup Z_3 \cup V(H)$. For any vertex $v \in V(G_0)$, all the paths between u and v in $G_0 \cup H$ must contain at least one red vertex. Similarly, for any vertex $w \in V(G_3)$, all the paths between u and w in $G_3 \cup H$ must contain at least one red vertex.*

Proof. Recall that all the vertices in $X_3 \cup Z_3$ are colored red. If $u \in X_3 \cup Z_3$, u is itself a red vertex, and therefore the lemma is trivially true. Assume now that $u \in (V(H) - (X_3 \cup Z_3))$. Since $V(G_0) \cap V(H) \subseteq X_3 \cup Z_3$, G_0 and H share only red vertices, and therefore each path between a vertex in H and a vertex in G_0 must contain at least one red vertex. The same argument applies to G_3. ∎

LEMMA 5.24. *For $2 \leq k \leq 4$, G_3 is a local certificate of k-edge-connectivity for $X_3 \cup Z_3$ in G_0.*

Proof. Let H be given with $V(G_0) \cap V(H) \subseteq X_3 \cup Z_3$, and let x and y be any two vertices of $X_3 \cup Z_3 \cup V(H)$. To prove the lemma, we basically follow the same ideas used for proving Lemma 5.22, but this time we consider edge-cuts that separate x and y instead. Note that since $x, y \in X_3 \cup Z_3 \cup V(H)$, x and y are in both $G_0 \cup H$ and $G_3 \cup H$. We prove the following two properties, which are the analogs of Lemmas 5.20 and 5.21.

(i) If $\lambda_{x,y}(G_0 \cup H) = \ell \leq 3$, then $\lambda_{x,y}(G_3 \cup H) \leq \ell$.
(ii) If $\lambda_{x,y}(G_3 \cup H) = \ell \leq 3$, then $\lambda_{x,y}(G_0 \cup H) \leq \ell$.

Properties (i) and (ii) imply that $\lambda_{x,y}(G_0 \cup H) = \ell \leq 3$ if and only if $\lambda_{x,y}(G_3 \cup H) = \ell \leq 3$. Note that any two vertices whose edge connectivity is not ℓ, $\ell \leq 3$, are at least 4-edge-connected. Hence, the lemma will follow from (i) and (ii).

We first prove (i). If $\lambda_{x,y}(G_0 \cup H) = \ell \leq 3$, then there is a minimum edge-cut $\gamma(x, y)$ separating x and y in $G_0 \cup H$ such that $|\gamma(x, y)| = \ell \leq 3$. Denote by $G_0' \cup H'$ and $G_0'' \cup H''$ the two graphs obtained from $G_0 \cup H$ after deleting the edges of $\gamma(x, y)$, with $G_0', G_0'' \subseteq G_0$, and $H', H'' \subseteq H$. Without loss of generality, assume that x is in $G_0' \cup H'$ and y is in $G_0'' \cup H''$. Let $\gamma_{G_0}(x, y) = \gamma(x, y) \cap E(G_0)$ and $\gamma_H(x, y) = \gamma(x, y) \cap E(H)$ be, respectively, the edges of $\gamma(x, y)$ in G_0 and in H. Due to the minimality of $\gamma(x, y)$, if $\gamma_{G_0}(x, y) \neq \emptyset$, $\gamma_{G_0}(x, y)$ must be an edge-cut of G_0 (separating G_0' and G_0''). If $\gamma_{G_0}(x, y) = \emptyset$, then $\gamma(x, y)$ uses no edges of G_0, and therefore it does not split G_0: in this case we assume that $G_0' = G_0$ and $G_0'' = \emptyset$. Similarly, if $\gamma_H \neq \emptyset$, $\gamma_H(x, y)$ must be an edge-cut of H (separating H' and H'').

Since $\gamma_{G_0}(x, y)$ is an edge-cut of G_0, and G_0 is 3-edge-connected, either $\gamma_{G_0}(x, y) = \emptyset$ or $|\gamma_{G_0}(x, y)| \geq 3$. Since $|\gamma(x, y)| \leq 3$, we are left with only two possibilities:

(a) $\gamma_{G_0}(x, y) = \emptyset$ (and therefore $\gamma(x, y) = \gamma_H(x, y)$);
(b) $|\gamma_{G_0}(x, y)| = 3$ and $\gamma_H(x, y) = \emptyset$ (and therefore $\gamma(x, y) = \gamma_{G_0}(x, y)$).

Assume we are in case (a). Since $\gamma_{G_0}(x, y) = \emptyset$, $G_0 = G_0'$ and $\gamma_H(x, y)$ splits $G_0 \cup H$ into $G_0 \cup H''$ and H'. Due to the fact that $\gamma_H(x, y)$ separates x and y, one of these vertices must be in H', and the other must be in $G_0 \cup H''$. Since $V(G_3) \subseteq V(G_0)$, we have that $V(H') \cap V(G_3) = \emptyset$. Thus, $\gamma_H(x, y)$ splits $G_3 \cup H$ into $G_3 \cup H''$ and H', and therefore it separates x and y in $G_3 \cup H$ too. This yields (i) for this case.

Assume now that we are in case (b). Since $\gamma(x, y) = \gamma_{G_0}(x, y)$ is a 3-edge-cut separating x and y in $G_0 \cup H$, it splits G_0 into G_0' and G_0'', which are both non-empty. Assume that x is in the same side as G_0' and y is in the same side as G_0''.

Let R' and R'' be the sets of red vertices, respectively, in G'_0 and in G''_0. Recall that $V(H) \cap V(G_3) \subseteq X_3 \cup Z_3$, and vertices of $X_3 \cup Z_3$ are colored red. Because of Lemma 5.23, there is a red vertex (of R') in each path between x and a vertex in G'_0, and there is a red vertex (of R'') in each path between y and a vertex in G''_0. This shows that $R', R'' \neq \emptyset$, and therefore by Lemma 5.17, there is a 3-edge-cut $\widehat{\gamma}$ separating R' and R'' in G_3. Again, because of Lemma 5.23, any path between x and y that goes through G_3 must contain at least one vertex in R' and one vertex in R''. Since R' and R'' are separated by $\widehat{\gamma}$ in G_3, any such path between x and y must contain at least one edge of $\widehat{\gamma}$. As a consequence of $\gamma_H(x, y) = \emptyset$, no path between x and y can go through H. Then, every path between x and y in $G_3 \cup H$ must go through $\widehat{\gamma}$, thus giving (i).

We now turn to (ii). Assume that the minimum edge-cut $\delta(x, y)$ separating x and y in $G_3 \cup H$ is of cardinality $|\delta(x, y)| = \ell \leq 3$. Let $\delta_{G_3}(x, y) = \delta(x, y) \cap E(G_3)$ and $\delta_H(x, y) = \delta(x, y) \cap E(H)$ be the edges of $\delta(x, y)$ in G_3 and in H. Due to the minimality of $\delta(x, y)$, if $\delta_{G_3}(x, y) \neq \emptyset$, $\delta_{G_3}(x, y)$ must be an edge-cut in $G_3(x, y)$. Similarly, if $\delta_H(x, y) \neq \emptyset$, $\delta_H(x, y)$ must be an edge-cut in H. Since G_3 is 3-edge-connected, either $\delta_{G_3}(x, y) = \emptyset$ or $|\delta_{G_3}(x, y)| \geq 3$. Once again, we have two possibilities: either (a) $\delta_{G_3}(x, y) = \emptyset$ or (b) $|\delta_{G_3}(x, y)| = 3$ and $\delta_H(x, y) = \emptyset$. In case (a), the same argument used for case (a) of (i) shows that $\delta(x, y) = \delta_H(x, y)$ is an ℓ-edge-cut separating x and y in $G_0 \cup H$ too, and therefore (ii) holds. If we are in case (b), $\delta_H(x, y) = \emptyset$, and $\delta(x, y) = \delta_{G_3}(x, y)$, $|\delta_{G_3}(x, y)| = 3$. Let R' and R'' be the red vertices separated by $\delta(x, y)$ in G_3. Exactly as in (i), any path between x and y that goes through G_3 must contain at least one (red) vertex in R' and one (red) vertex in R''. As before, this implies two things. First, by Lemma 5.17, there is a 3-edge-cut $\widehat{\delta}$ separating R' and R'' in G_0. Second, any path between x and y in G_0 must contain at least one edge of $\widehat{\delta}$. As a consequence of $\gamma_H(x, y) = \emptyset$, no path between x and y can go through H. Then, every path between x and y in $G_0 \cup H$ must go through $\widehat{\delta}$. Thus, $\widehat{\delta}$ is a 3-edge-cut separating x and y in $G_0 \cup H$. Since $\ell = 3$ in case (b), this proves (ii). □

LEMMA 5.25. *G_3 preserves planarity.*

Proof. Let H be any graph such that $V(H) \cap V(G_0) \subseteq X_3 \cup Z_3$. $G_3 \cup H$ can be obtained from $G_0 \cup H$ by means of operations (1), (2), and (3), which consist of deletion of edges, contraction of edges, and insertion of parallel edges. Since all these operations preserve planarity, if $G_0 \cup H$ is planar, then $G_3 \cup H$ is planar. Thus, G_3 preserves planarity according to Definition 3.5. □

Lemmas 5.16, 5.22, 5.24, and 5.25 prove that G_3 is a solution to Problem 5.3.

5.4. Compressed certificates for 3- and 4-edge-connectivity. In this section, we combine the results obtained in sections 5.1, 5.2, and 5.3 to derive compressed full certificates for 3- and 4-edge-connectivity. Let $G = (V, E)$ be a planar connected graph, and let X be a set of interesting vertices in G. Let H be given, with $V(G) \cap V(H) \subseteq X$. Our algorithm for finding compressed full certificates consists of the following steps.

Step 1 (*decreasing the number of* 2*-edge-connected components*). Set $X_1 = X$, $G_0 = G$, and compute a solution G_1 to Problem 5.1 as shown in section 5.1.

Step 2 (*decreasing the number of* 3*-edge-connected components*). Let N_1 be the set of endpoints of bridges left in G_1 at the end of Step 1. Because of the previous step, there are $O(|X|)$ bridges in G_1 and therefore $|N_1| = O(|X|)$. For each 2-edge-connected component B left in G_1 do the following: set $X_2(B) = (X_1 \cup N_1) \cap V(B)$, $G_0(B) = B$, and compute a solution $G_2(B)$ to Problem 5.2 as shown in section 5.2.

Let $X_2 = \sum_B X_2(B)$. Note that $|X_2| = \sum_B |X_2(B)| = |X_1 \cup N_1| = O(|X|)$, so this compression uses $O(|X|)$ interesting vertices overall. Denote by G_2 the graph obtained from G_1 by replacing each 2-edge-connected component B with $G_2(B)$. Since each $G_2(B)$ is a solution to Problem 5.2, $G_2(B)$ has $O(|X_2(B)|)$ 3-edge-connected components. Hence, the overall number of 3-edge-connected components left in G_2 will be $\sum_B O(|X_2(B)|) = O(|X|)$.

Step 3 (*final compression*). Let N_2 be the set of endpoints of 2-edge-cuts left in G_2 at the end of Step 2. Since the total number of endpoints of 2-edge-cuts is linear in the number of 3-edge-connected components, because of the previous step, $|N_2| = O(|X|)$. Let T be a 3-edge-connected component of G_2: recall that T is obtained by means of splits (as defined in section 4.1), and it contains virtual edges corresponding to 2-edge-cuts. Let $M_2(T)$ be the set of virtual edges in T. For each 3-edge-connected component T left in G_2 do the following: set $X_3(T) = (X_2 \cup N_2) \cap V(T)$, $G_0(T) = T$, $Y_3(T) = M_2(T)$, and compute a solution $G_3(T)$ to Problem 5.3 as shown in section 5.3. Let $X_3 = \sum_T X_3(T)$ and $M_2 = \sum_T M_2(T)$ be, respectively, the total number of red vertices and red edges. Note that $|X_3| = \sum_T |X_3(T)| = |X_2 \cup N_2| = O(|X|)$ and $|M_2| = \sum_T |M_2(T)| \leq \sum_T |N_2 \cap V(T)| \leq \sum_T |X_3(T)| = O(|X|)$, so again the compression uses $O(|X|)$ interesting vertices overall. Then merge back the compressed 3-edge-connected components along their original 2-edge-cuts. This is well defined, since all the virtual edges, and hence the endpoints of the original 2-edge-cuts, are colored red and therefore preserved in the graphs $G_3(T)$. Denote by G_3 the graph obtained from G_2 in this way.

We now prove that G_3 is a planarity-preserving compressed full certificate of 3- and 4-edge-connectivity for X in G.

LEMMA 5.26. *Let $G = (V, E)$ be an undirected planar graph with a set $X \subseteq V(G)$. Let G_3 be the graph defined at the end of Step* 3 *above. Then G_3 is a planarity-preserving compressed full certificate for X in G with respect to* 3- *and* 4-*edge-connectivity.*

Proof. Let G_1 be the graph obtained after Step 1. Since G_1 is a solution to Problem 5.1, then for every $k \geq 2$, G_1 is a planarity-preserving full certificate of k-edge-connectivity for X in G and has $O(|X|)$ 2-edge-connected components.

Let G_2 be the graph obtained from G_1 by replacing each 2-edge-connected component B with $G_2(B)$, where each $G_2(B)$ is a solution to Problem 5.2 with interesting vertices $X_2(B)$ as described in Step 2. We claim that for every $k \geq 2$, G_2 is a planarity-preserving compressed full certificate for k-edge-connectivity of X in G. Since each $G_2(B)$ is planarity-preserving, as shown in section 5.2, and G_2 is obtained by only adding bridges between the graphs $G_2(B)$, G_2 will be planarity-preserving too. The fact that G_2 is compressed follows easily from repeated applications of Lemma 5.11. It remains to show that G_2 is indeed a certificate for X in G. To prove this, fix a particular 2-edge-connected component B of G_1. Let $X_2(B)$ be the interesting vertices of B as defined in Step 2, and let $G_2^{(1)}$ be the graph obtained after replacing B with $G_2(B)$. By Lemma 5.13 and Corollary 5.2, for every $k \geq 2$, $G_2(B)$ is a full certificate for k-edge-connectivity of $X_2(B)$ in B. By Lemma 3.2 applied with $C' = G_2(B)$, $X' = X_2(B)$, $G' = B$, $C'' = G'' = G_1 - B$, and $X'' = V(G_1 - B)$, we have that for every $k \geq 2$, $G_2^{(1)}$ is a full certificate for k-edge-connectivity of $X_2(B) \cup V(G_1 - B)$ in G_1. Since $X \cup X_2(B) \subseteq X_2(B) \cup V(G_1 - B)$, by Lemma 3.3 we have that for every $k \geq 2$, $G_2^{(1)}$ is a full certificate for k-edge-connectivity of $X \cup X_2(B)$ in G_1. Repeating this argument for each 2-edge-connected component B of G_1 yields that for every $k \geq 2$, G_2 is a full certificate for k-edge-connectivity of X_2 in G_1. Since G_1 is a full

certificate of k-edge-connectivity for X in G, and $X \subseteq X_2$, by Lemma 3.1 we have that G_2 is a full certificate of k-edge-connectivity for X in G.

Let G_3 be the graph obtained at the end of Step 3. We claim that G_3 is a planarity-preserving compressed full certificate for 3- and 4-edge-connectivity of X in G. By Lemma 5.25, each $G_3(T)$, which is a solution to Problem 5.3, is planar. Recall that G_3 is obtained by splitting G_2, replacing each 3-edge-connected component T with $G_3(T)$, and by merging back the compressed 3-edge-connected components $G_3(T)$. Since splits and merges preserve planarity, G_3 will be planarity-preserving. The fact that G_3 is compressed follows from repeated applications of Lemma 5.19. To show that G_3 is a certificate of 3- and 4-edge-connectivity for X in G, we proceed as follows. Fix a particular 3-edge-connected component T of G_2, and let $B(T)$ be the 2-edge-connected component containing T, whose interesting vertices (defined in step 2) are in $X_2(B)$. Let $X_3(T)$ be the interesting vertices of T as defined in Step 3, and let $G_3^{(1)}$ be the graph obtained from G_2 after replacing T with $G_3(T)$. By Lemmas 5.22 and 5.24, for $2 \le k \le 4$, $G_3(T)$ is a full certificate for k-edge-connectivity of $X_3(T)$ in T. Let $B(T)^{(1)}$ be the 2-edge-component of $G_3^{(1)}$ containing $G_3(T)$. By Theorem 4.2 applied to the 2-edge-connected graph $B(T)$, we have that for $2 \le k \le 4$, $B(T)^{(1)}$ is a full certificate for k-edge-connectivity of $X_2(B)$ in $B(T)$. Applying Lemma 3.2 as before yields that for $2 \le k \le 4$, $G_3^{(1)}$ is a full certificate for k-edge-connectivity of $X_2(B) \cup V(G_2 - B(T))$ in G_2. Again, by Lemma 3.3, we have that for $2 \le k \le 4$, $G_3^{(1)}$ is a full certificate for k-edge-connectivity of $X \cup X_2(B)$ in G_2. Repeating this argument for each 3-edge-connected component T of G_2 yields that for $2 \le k \le 4$, G_3 is a full certificate for k-edge-connectivity of X in G_2. Since we have already proved that, for any $k \ge 2$, G_2 is a full certificate of k-edge-connectivity for X in G, by Lemma 3.1 we have that G_3 is a full certificate of k-edge-connectivity for X in G, $2 \le k \le 4$. □

THEOREM 5.1. *We can maintain a planar graph subject to insertions and deletions that preserve planarity, and allow testing the* 3- *and* 4*-edge-connectivity of the graph in* $O(n^{1/2})$ *time per update, or test whether two vertices are* 3- *or* 4*-edge-connected, in* $O(n^{1/2})$ *time per update or query.*

Proof. Lemma 5.26 shows that planarity-preserving compressed full certificates exist, so we can use Lemma 3.4 to find such certificates in linear time. We then apply Theorem 3.4 with $T(n) = Q(n) = O(n)$. □

6. Vertex connectivity. As another application of our basic sparsification technique, we describe an algorithm for 2- and 3-vertex-connectivity.

THEOREM 6.1. *We can maintain a planar graph, subject to insertions and deletions that preserve planarity, and allow queries that test the* 2- *or* 3*-vertex-connectivity of the graph, or test whether two vertices belong to the same* 2- *or* 3*-vertex-connected component, in* $O(n^{1/2})$ *amortized time per update or query.*

Proof. Galil, Italiano, and Sarnak [21] show that compressed certificates for 2- and 3-vertex-connectivity can be found in linear time. It can be verified that their certificates comply with our Definition 3.3. Using their compressed certificates in our separator tree gives $O(n^{1/2})$ amortized time per update by Theorem 3.2. □

We remark that the same $O(n^{1/2})$ bound can be achieved for the problem of maintaining minimum spanning forests and for local connectivity. This does not improve the $O(n^{1/2})$ bounds that can be achieved using the algorithms for general graphs [9], however. In the companion paper [11], we achieve better $O(\log^2 n)$ bounds for these problems.

7. Conclusions and open problems. We have introduced a new and general technique for designing dynamic planar graph algorithms. This technique is based upon sparsification, compressed certificates, and balanced separator trees, and improves many known bounds. In the companion paper [11], we have showed how this technique produces fast, fully dynamic algorithms for minimum spanning forests, for 2-edge-connectivity on planar graphs, and for dynamic planarity testing. In this paper we have applied this technique to 2-vertex-, 3-vertex-, 3-edge-, and 4-edge-connectivity. For edge connectivity, we have developed certificates which may be interesting on their own. There are a number of related and perhaps interesting questions.

The algorithms described in [11] exploit a stability property in their certificates to support polylogarithmic time updates in planar graphs. For the problems tackled in this paper, we have not been able to apply stable sparsification, and our bounds are $O(n^{1/2})$. Is it possible to exploit stability and to achieve polylogarithmic bounds for these problems too?

Furthermore, are there compressed certificates for other problems? For higher edge connectivity, for instance, the cactus tree gets more complicated (see, for instance, [6]) and makes our way of computing compressed certificates for edge connectivity difficult to generalize. Are there compressed certificates for edge connectivity which are not based on cactus trees?

Finally, can we exploit known partially dynamic algorithms to speed up the insertion times in our bounds? This would be particularly appealing, since there are very fast, partially dynamic algorithms already available in the literature for edge and vertex connectivity [6, 20, 27, 29, 30, 36].

Acknowledgments. We are deeply indebted to the anonymous referees, whose thorough reading of the paper lead to a substantial improvement in its presentation.

REFERENCES

[1] J. Cheriyan and S. N. Maheshwari, *Finding nonseparating induced cycles and independent spanning trees in* 3*-connected graphs*, J. Algorithms, 9 (1988), pp. 507–537.

[2] J. Cheriyan, M. Y. Kao, and R. Thurimella, *Scan-first search and sparse certificates: An improved parallel algorithm for k-vertex connectivity*, SIAM J. Comput., 22 (1993), pp. 157–174.

[3] G. B. Dantzig and D. R. Fulkerson, *On the max-flow min-cut theorem of networks*, in Linear Inequalities and Related Systems, Ann. Math. Study 38, Princeton University Press, Princeton, NJ, 1956, pp. 215–221.

[4] E. A. Dinitz, A. V. Karzanov, and M. V. Lomonosov, *A structure of the system of all minimal cuts of a graph*, in Studies in Discrete Optimization, A. A. Fridman, ed., Nauka, Moscow, 1976, pp. 290–306. (In Russian.)

[5] E. A. Dinitz, *The* 3*-edge-components and a structural description of all* 3*-edge-cuts in a graph*, in Proc. 18th Int. Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science 657, Springer-Verlag, New York, 1992, pp. 145–157.

[6] E. A. Dinitz, *Maintaining the* 4*-edge-connected components of a graph on-line*, in Proc. 2nd Israel Symp. on Theory of Computing and Systems, Nethania, Israel, IEEE Computer Society Press, Piscataway, NJ, 1993, pp. 88–97.

[7] D. Eppstein, *Subgraph isomorphism for planar graphs and related problems*, in Proc. 6th ACM-SIAM Symp. on Discrete Algorithms, 1995, pp. 189–196; J. Graph Algorithms Appl., to appear.

[8] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, *Sparsification—A technique for speeding up dynamic graph algorithms*, in Proc. 33rd IEEE Symp. on Foundations of Computer Science, 1992, pp. 60–69.

[9] D. Eppstein, Z. Galil, and G. F. Italiano, *Improved Sparsification*, Tech. Report 93-20, Department of Information and Computer Science, University of California, Irvine, 1993.
[10] D. Eppstein, Z. Galil, G. F. Italiano, and T. H. Spencer, *Separator based sparsification for dynamic planar graph algorithms*, in Proc. 25th Annual ACM Symp. on Theory of Computing, 1993, pp. 208–217.
[11] D. Eppstein, Z. Galil, G. F. Italiano, and T. H. Spencer, *Separator based sparsification* I*: Planarity testing and minimum spanning trees*, J. Comput. System Sci., Special issue on STOC 93, 52 (1996), pp. 3–27.
[12] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung, *Maintenance of a minimum spanning forest in a dynamic plane graph*, J. Algorithms, 13 (1992), pp. 33–54.
[13] A. Frank, T. Ibaraki, and H. Nagamochi, *On sparse subgraphs preserving connectivity properties*, J. Graph Theory, 17 (1993), pp. 275–281.
[14] G. N. Frederickson, *Data structures for on-line updating of minimum spanning trees, with applications*, SIAM J. Comput., 14 (1985), pp. 781–798.
[15] G. N. Frederickson, *Ambivalent data structures for dynamic* 2*-edge-connectivity and k smallest spanning trees*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, 1991, pp. 632–641.
[16] H. N. Gabow, *A matroid approach to finding edge connectivity and packing arborescences*, in Proc. 23rd ACM Symp. on Theory of Computing, 1991, pp. 112–122.
[17] H. N. Gabow, *Applications of a poset representation to edge connectivity and graph rigidity*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, 1991, pp. 812–821.
[18] H. N. Gabow and M. Stallman, *Efficient algorithms for graphic matroid intersection and parity*, in Proc. 12th Int. Coll. Automata, Languages, and Programming, Lecture Notes in Computer Science 194, Springer-Verlag, New York, 1985, pp. 210–220.
[19] Z. Galil and G. F. Italiano, *Maintaining biconnected components of dynamic planar graphs*, in Proc. 18th Int. Colloq. Automata, Languages, and Programming, Lecture Notes in Computer Science 510, Springer-Verlag, New York, 1991, pp. 339–350.
[20] Z. Galil and G. F. Italiano, *Maintaining the* 3*-edge-connected components of a graph on-line*, SIAM J. Comput., 22 (1993), pp. 11–28.
[21] Z. Galil, G. F. Italiano, and N. Sarnak, *Fully dynamic planarity testing*, in Proc. 24th ACM Symp. on Theory of Computing, 1992, pp. 495–506.
[22] D. Giammarresi and G. F. Italiano, *Decremental* 2*- and* 3*-connectivity on planar graphs*, Algorithmica, 16 (1996), pp. 263–287.
[23] M. T. Goodrich, *Planar separators and parallel polygon triangulation*, in Proc. 24th ACM Symp. on Theory of Computing, 1992, pp. 507–516.
[24] J. Hershberger, M. Rauch, and S. Suri, *Data structures for two-edge connectivity on planar graphs*, Theoret. Comput. Sci., 130 (1994), pp. 139–161.
[25] J. Hopcroft and R. E. Tarjan, *Dividing a graph into triconnected components*, SIAM J. Comput., 2 (1973), pp. 135–158.
[26] A. Itai and M. Rodeh, *The multi-tree approach to reliability in distributed networks*, Inform. and Comput., 79 (1988), pp. 3–59.
[27] A. Kanevsky, R. Tamassia, G. Di Battista, and J. Chen, *On-line maintenance of the four-connected components of a graph*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, 1991, pp. 793–801.
[28] A. V. Karzanov and E. A. Timofeev, *Efficient algorithm for finding all minimal edge-cuts of a nonoriented graph*, Cybernetics, 1986, pp. 156–162. (Trans. from Kibernetica, 2(1986), pp. 8–12.)
[29] J. A. La Poutré, *Dynamic Graph Algorithms and Data Structures*, Ph.D. Thesis, Utrecht University, the Netherlands, September 1991.
[30] J. A. La Poutré, *Maintenance of triconnected components of graphs*, in Proc. 19th Int. Coll. Automata, Languages, and Programming, Lecture Notes in Computer Science 623, Springer-Verlag, New York, 1992, pp. 354–365.
[31] H. Nagamochi and T. Ibaraki, *A linear-time algorithm for finding a sparse* k*-connected spanning subgraph of a* k*-connected graph*, Algorithmica, 7 (1992), pp. 583–596.
[32] M. Rauch, *Fully dynamic biconnectivity in graphs*, in Proc. 33rd IEEE Symp. on Foundations of Computer Science, 1992, pp. 50–59.
[33] R. Tamassia, *A dynamic data structure for planar graph embedding*, in Proc. 15th Int. Colloq. Automata, Languages, and Programming, Lecture Notes in Computer Science 317, Springer-Verlag, New York, 1988, pp. 576–590.
[34] R. E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

[35] R. Thurimella, *Techniques for the Design of Parallel Graph Algorithms*, Ph.D. Thesis, Univ. of Texas, Austin, 1989.
[36] J. Westbrook and R. E. Tarjan, *Maintaining bridge-connected and biconnected components on-line*, Algorithmica, 7 (1992), pp. 433–464.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 383–393

A DOWNWARD COLLAPSE WITHIN THE POLYNOMIAL HIERARCHY*

EDITH HEMASPAANDRA†, LANE A. HEMASPAANDRA‡, AND HARALD HEMPEL§

Abstract. Downward collapse (also known as upward separation) refers to cases where the equality of two larger classes implies the equality of two smaller classes. We provide an unqualified downward collapse result completely within the polynomial hierarchy. In particular, we prove that, for $k > 2$, if $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{P}^{\Sigma_k^p[2]}$ then $\Sigma_k^p = \Pi_k^p = \mathrm{PH}$. We extend this to obtain a more general downward collapse result.

Key words. computational complexity theory, easy-hard arguments, downward collapse, polynomial hierarchy

AMS subject classifications. 68Q15, 68Q10, 03D15, 03D10

PII. S0097539796306474

1. Introduction.

The theory of NP-completeness does not resolve the issue of whether P and NP are equal. However, it does unify the issues of whether thousands of natural problems—the NP-complete problems—have deterministic polynomial-time algorithms. The study of downward collapse is similar in spirit. By proving downward collapses, we seek to tie together central open issues regarding the computing power of complexity classes. For example, the main result of this paper shows that (for $k > 2$) the issue of whether the kth level of the polynomial hierarchy is closed under complementation is identical to the issue of whether two queries to this level give more power than one query to this level.

Informally, downward collapse (equivalent terms are "downward translation of equality" and "upward separation") refers to cases in which the collapse of larger classes implies the collapse of smaller classes (for background, see, e.g., [All91, AW90]). For example, $\mathrm{NP}^{\mathrm{NP}} = \mathrm{coNP}^{\mathrm{NP}} \Rightarrow \mathrm{NP} = \mathrm{coNP}$ would be a (shocking and inherently nonrelativizing [Ko89]) downward collapse, the "downward" part referring to the well-known fact that $\mathrm{NP} \cup \mathrm{coNP} \subseteq \mathrm{NP}^{\mathrm{NP}} \cap \mathrm{coNP}^{\mathrm{NP}}$.

Downward collapse results are extremely rare, but there are some results in the literature that do have the general flavor of downward collapse. Cases where the collapse of larger classes forces sparse sets (but perhaps not nonsparse sets) to fall out of smaller classes were found by Hartmanis, Immerman, and Sewelson ([HIS85], see also [Boo74]) and by others (e.g., Rao, Rothe, and Watanabe [RRW94], but in contrast see also [HJ95]). Existential cases have long been implicitly known (i.e., theorems such as "if PH = PSPACE then $(\exists k)\,[\mathrm{PH} = \Sigma_k^p]$"—note that here one can prove nothing about what value k might have). Regarding probabilistic classes, Ko [Ko82] proved that "if NP $\subseteq$ BPP then NP = R," and Babai, Fortnow, Nisan, and Wigderson [BFNW93] proved the striking result that "if EH = E then P = BPP."

* Received by the editors July 2, 1996; accepted for publication (in revised form) January 24, 1997; published electronically July 7, 1998. This research was supported in part by grants NSF-INT-9513368/DAAD-315-PRO-fo-ab and NSF-CCR-9322513.

http://www.siam.org/journals/sicomp/28-2/30647.html

† Department of Mathematics, Le Moyne College, Syracuse, NY 13214 (edith@bamboo.lemoyne.edu). This work was done in part while visiting Friedrich-Schiller-Universität Jena.

‡ Department of Computer Science, University of Rochester, Rochester, NY 14627 (lane@cs.rochester.edu). This work was done in part while visiting Friedrich-Schiller-Universität Jena.

§ Institut für Informatik, Friedrich-Schiller-Universität Jena, 07743 Jena, Germany (hempel@informatik.uni-jena.de). This work was done in part while visiting Le Moyne College.

Hemaspaandra, Rothe, and Wechsung have given an example involving degenerate certificate schemes [HRW], and examples due to Allender [All86, section 5] and Hartmanis and Yesha [HY84, section 4] are known regarding circuit-related classes.[1]

We provide an unqualified downward collapse result that is not restricted to sparse or tally sets, whose conclusion does not contain a variable that is not specified in its hypothesis, and that deals with classes whose *ex ante* containments[2] are clear (and plausibly strict). Namely, as is standard, let $\mathrm{P}^{\mathcal{C}[j]}$ denote the class of languages computable by P machines making at most j queries to some set from $\mathcal{C}$. We prove that, for each $k > 2$, it holds that

$$\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{P}^{\Sigma_k^p[2]} \Rightarrow \Sigma_k^p = \Pi_k^p = \mathrm{PH}.$$

(As just mentioned in footnote 2, the classes in the hypothesis clearly have the property that they contain both Σ_k^p and Π_k^p.) The best previously known results from the assumption $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{P}^{\Sigma_k^p[2]}$ collapse the polynomial hierarchy only to a level that contains Σ_{k+1}^p and Π_{k+1}^p [CK96, BCO93].

Our proof actually establishes a $\Sigma_k^p = \Pi_k^p$ collapse from a hypothesis that is even weaker than $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{P}^{\Sigma_k^p[2]}$. Namely, we prove that, for $i < j < k$ and $i < k-2$, if one query each (in parallel) to the ith and kth levels of the polynomial hierarchy equals one query each (in parallel) to the jth and kth levels of the polynomial hierarchy, then $\Sigma_k^p = \Pi_k^p = \mathrm{PH}$.

In the final section of the paper, we generalize from 1-versus-2 queries to m-versus-$(m+1)$ queries. In particular, we show that our main result is in fact a reflection of an even more general downward collapse: if the truth-table hierarchy over Σ_k^p collapses to its mth level, then the boolean hierarchy over Σ_k^p collapses one level further than one would expect.

2. Simple case. Our proof works by extracting advice internally and algorithmically, while holding down the number of quantifiers needed, within the framework of a so-called "easy-hard" argument. Easy-hard arguments were introduced by Kadin [Kad88] and were further used by Chang and Kadin ([CK96]; see also [Cha91]) and Beigel, Chang, and Ogiwara [BCO93] (we follow the approach of Beigel, Chang, and Ogiwara).

THEOREM 2.1. *For each $k > 2$ it holds that*

$$\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{P}^{\Sigma_k^p[2]} \Rightarrow \Sigma_k^p = \Pi_k^p = \mathrm{PH}.$$

Theorem 2.1 follows immediately[3] from Theorem 2.4 below, which states that, for $i < j < k$ and $i < k-2$, if one query each to the ith and kth levels of the polynomial

[1] Note that we are not claiming that all the above examples from the literature are totally unqualified downward collapse results, but rather we are merely stating that they have the general flavor of downward collapse. In some cases, the results mentioned above do not fully witness what one might hope for from the notion of "downward." Ideally, downward collapse results would be truly "downward" in the sense that they would be of the form "if $\mathcal{A} = \mathcal{B}$ then $\mathcal{C} = \mathcal{D}$," where the classes are such that (a) $\mathcal{A} \cap \mathcal{B} \supseteq \mathcal{C} \cup \mathcal{D}$ is a well-known result and (b) it is not currently known that $\mathcal{A} \cap \mathcal{B} = \mathcal{C} \cup \mathcal{D}$. The downward collapses proven in this paper do have this strong "downward" form.

[2] For example, in the case of Theorem 2.1, $\Sigma_k^p \cup \Pi_k^p \subseteq \mathrm{P}^{\Sigma_k^p[1]} \cap \mathrm{P}^{\Sigma_k^p[2]}$ is well known to be true (and most researchers suspect that the inclusion is strict).

[3] In particular, taking $i = 0$ and $j = k-1$ in Theorem 2.4 yields a statement that itself clearly implies Theorem 2.1.

hierarchy equals one query each to the jth and kth levels of the polynomial hierarchy, then $\Sigma_k^p = \Pi_k^p = \mathrm{PH}$.

DPTM will refer to deterministic polynomial-time oracle Turing machines, whose polynomial time upper bounds are clearly clocked and are independent of their oracles. We will also use the following definitions.

DEFINITION 2.2.

1. *Let $M^{(A,B)}$ denote DPTM M making, simultaneously (i.e., in a truth-table fashion), at most one query to oracle A and at most one query to oracle B, and let*

$$\mathrm{P}^{(\mathcal{C},\mathcal{D})} = \{L \subseteq \Sigma^* \mid (\exists C \in \mathcal{C})(\exists D \in \mathcal{D})(\exists \text{DPTM } M)[L = L(M^{(C,D)})]\}.$$

2. $A\tilde{\Delta}B = \{\langle x, y\rangle \mid x \in A \Leftrightarrow y \notin B\}$ (*see* [BCO93]).

LEMMA 2.3. *Let $0 \le i < k$, let $L_{\mathrm{P}^{\Sigma_i^p[1]}}$ be any set $\le_{\mathrm{m}}^p$-complete for $\mathrm{P}^{\Sigma_i^p[1]}$, and let $L_{\Sigma_k^p}$ be any language $\le_{\mathrm{m}}^p$-complete for Σ_k^p. Then $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\Sigma_k^p}$ is $\le_{\mathrm{m}}^p$-complete for $\mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)}$.*

Proof. Clearly $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\Sigma_k^p}$ is in $\mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)}$. Regarding $\le_{\mathrm{m}}^p$-hardness for $\mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)}$, let $L \in \mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)}$ via transducer M, Σ_i^p set A, and Σ_k^p set B. Without loss of generality, on each input x, M asks exactly one question a_x to A and one question b_x to B. Define sets D and E as follows:

$D = \{x \mid M^{(A,B)}$ accepts x if a_x is answered correctly and b_x is answered "no"$\}$.

$E = \{x \mid b_x \in B$ and the (one-variable) truth-table with respect to b_x of $M^{(A,B)}$ on input x induced by the correct answer to a_x is neither "always accept" nor "always reject"$\}$.

Note that $D \in \mathrm{P}^{\Sigma_i^p[1]}$, and that $E \in \Sigma_k^p$, since $i < k$. But $L \le_{\mathrm{m}}^p D\tilde{\Delta}E$ via the reduction $f(x) = \langle x, x\rangle$. So clearly $L \le_{\mathrm{m}}^p L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\Sigma_k^p}$, via the reduction $\widehat{f}(x) = \langle f'(x), f''(x)\rangle$, where f' and f'' are, respectively, reductions from D to $L_{\mathrm{P}^{\Sigma_i^p[1]}}$ and from E to $L_{\Sigma_k^p}$. □

Theorem 2.4 contains the following two technical advances. First, it internally extracts information in a way that saves a quantifier. (In contrast, the earliest easy-hard arguments in the literature merely ensure that $\Sigma_k^p \subseteq \Pi_k^p/\mathrm{poly}$ and from that infer a weak polynomial hierarchy collapse. Even the interesting recent strengthenings of the argument [BCO93] still, under the hypothesis of Theorem 2.4, conclude only a collapse of the polynomial hierarchy to a level a bit worse than Σ_{k+1}^p.) The second advance is that previous easy-hard arguments seek to determine whether there exists a hard string for a length or not. Then they use the fact that if there is not a hard string, all strings (at the length) are easy. In contrast, we *never* search for a hard string; rather, we use the fact that the input itself (which we do not have to search for as, after all, it is our input) is either easy or hard. So we check whether the input is easy, and if so we can use it as an easy string, and if not, it must be a hard string so we can use it that way. This innovation is important in that it allows Theorem 2.1 to apply for all $k > 2$—as opposed to merely applying for all $k > 3$, which is what we would get without this innovation. (Following a referee's suggestion, we mention that during a first traversal the reader may wish to consider just the $i = 0$ and $j = 1$ special case of Theorem 2.4 and its proof, as this provides a restricted version that is easier to read.)

THEOREM 2.4. *Let $0 \le i < j < k$ and $i < k - 2$. If $\mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)} = \mathrm{P}^{(\Sigma_j^p,\Sigma_k^p)}$ then $\Sigma_k^p = \Pi_k^p = \mathrm{PH}$.*

Proof. Suppose $\mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)} = \mathrm{P}^{(\Sigma_j^p,\Sigma_k^p)}$. Let $L_{\mathrm{P}^{\Sigma_i^p[1]}}$, $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}$, and $L_{\Sigma_k^p}$ be $\leq_{\mathrm{m}}^p$-complete for $\mathrm{P}^{\Sigma_i^p[1]}$, $\mathrm{P}^{\Sigma_{i+1}^p[1]}$, and Σ_k^p, respectively; such sets exist. From Lemma 2.3 it follows that $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\Sigma_k^p}$ is $\leq_{\mathrm{m}}^p$-complete for $\mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)}$. Since (as $i<j$) $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\tilde{\Delta}L_{\Sigma_k^p} \in \mathrm{P}^{(\Sigma_j^p,\Sigma_k^p)}$, and by assumption $\mathrm{P}^{(\Sigma_j^p,\Sigma_k^p)} = \mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)}$, there exists a polynomial-time many-one reduction h from $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\tilde{\Delta}L_{\Sigma_k^p}$ to $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\Sigma_k^p}$. So, for all $x_1, x_2 \in \Sigma^*$: if $h(\langle x_1, x_2\rangle) = \langle y_1, y_2\rangle$, then $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow x_2 \notin L_{\Sigma_k^p})$ if and only if $(y_1 \in L_{\mathrm{P}^{\Sigma_i^p[1]}} \Leftrightarrow y_2 \notin L_{\Sigma_k^p})$. Equivalently, for all $x_1, x_2 \in \Sigma^*$, the following fact holds.

Fact 1.
If $h(\langle x_1, x_2\rangle) = \langle y_1, y_2\rangle$,
then

$(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow x_2 \in L_{\Sigma_k^p})$ if and only if $(y_1 \in L_{\mathrm{P}^{\Sigma_i^p[1]}} \Leftrightarrow y_2 \in L_{\Sigma_k^p})$.

We can use h to recognize some of $\overline{L_{\Sigma_k^p}}$ by a Σ_k^p algorithm. The definitions of easy and hard used in this paper follow the easy and hard concepts used by Kadin [Kad88], Chang and Kadin ([CK96]; see also [Cha91]), and Beigel, Chang, and Ogiwara [BCO93], modified as needed for our goals. In particular, we say that a string x is *easy for length* n if there exists a string x_1 such that $|x_1| \leq n$ and $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \notin L_{\mathrm{P}^{\Sigma_i^p[1]}})$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$.

Let p be a fixed polynomial, which will be exactly specified later in the proof. We have the following Σ_k^p algorithm to test whether $x \in \overline{L_{\Sigma_k^p}}$ in the case that (our input) x is an easy string for $p(|x|)$. On input x, guess x_1 with $|x_1| \leq p(|x|)$, let $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$, and accept if and only if $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \notin L_{\mathrm{P}^{\Sigma_i^p[1]}})$ and $y_2 \in L_{\Sigma_k^p}$. In light of Fact 1 above, it is clear that this is correct.

We say that x is *hard for length* n if $|x| \leq n$ and x is not easy for length n, i.e., if $|x| \leq n$ and for all x_1 with $|x_1| \leq n$ it holds that $x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \in L_{\mathrm{P}^{\Sigma_i^p[1]}}$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$.

If x is a hard string for length n, then x induces a many-one reduction from $(L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}})^{\leq n}$ to $L_{\mathrm{P}^{\Sigma_i^p[1]}}$; namely, $f(x_1) = y_1$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$. Note that f is computable in time polynomial in $\max(n, |x_1|)$.

We can use hard strings to obtain a Σ_k^p algorithm for $\overline{L_{\Sigma_k^p}}$. Let M be a Π_{k-i-1}^p machine such that M with oracle $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}$ recognizes $\overline{L_{\Sigma_k^p}}$. Let the run-time of M be bounded by polynomial p, which without loss of generality satisfies $(\forall \widehat{m} \geq 0)[p(\widehat{m}+1) > p(\widehat{m}) > 0]$ (as promised above, we have now specified p). Then

$$\left(\overline{L_{\Sigma_k^p}}\right)^{=n} = (L\left(M^{\left(L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\right)^{\leq p(n)}}\right))^{=n}.$$

If there exists a hard string for length $p(n)$, then this hard string induces a reduction from $(L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}})^{\leq p(n)}$ to $L_{\mathrm{P}^{\Sigma_i^p[1]}}$. Thus, with any hard string for length $p(n)$ in hand, call it w_n, $\widehat{M}$ with oracle $L_{\mathrm{P}^{\Sigma_i^p[1]}}$ recognizes $\overline{L_{\Sigma_k^p}}$ for strings of length n, where $\widehat{M}$ is the machine that simulates M but replaces each query q by the first component of $h(\langle q, w_n\rangle)$. It follows that if there exists a hard string for length $p(n)$, then this string induces a Π_{k-1}^p algorithm for $(\overline{L_{\Sigma_k^p}})^{=n}$ and therefore certainly a Σ_k^p algorithm for $(\overline{L_{\Sigma_k^p}})^{=n}$.

However, now we have an $\mathrm{NP}^{\Sigma_{k-1}^p} = \Sigma_k^p$ algorithm for $\overline{L_{\Sigma_k^p}}$: on input x, the NP base machine of $\mathrm{NP}^{\Sigma_{k-1}^p}$ executes the following algorithm.

1. Using its Σ_{k-1}^p oracle, it deterministically determines whether the input x is an easy string for length $p(|x|)$. This can be done, as checking whether the input is an easy string for length $p(|x|)$ can be done by one query to Σ_{i+2}^p, and $i+2 \leq k-1$ by our $i < k-2$ hypothesis.
2. If the previous step determined that the input is not an easy string, then the input must be a hard string for length $p(|x|)$. So simulate the Σ_k^p algorithm induced by this hard string (i.e., the input x itself) on input x (via our NP machine itself simulating the base level of the Σ_k^p algorithm and using the NP machine's oracle to simulate the oracle queries made by the base level NP machine of the Σ_k^p algorithm being simulated).
3. If the first step determined that the input x is easy for length $p(|x|)$, then our NP machine simulates (using itself and its oracle) the Σ_k^p algorithm for easy strings on input x.

We need one brief technical comment. The Σ_{k-1}^p oracle in the above algorithm is being used for a number of different sets. However, as Σ_{k-1}^p is closed under disjoint union, this presents no problem as we can use the disjoint union of the sets, while modifying the queries so they address the appropriate part of the disjoint union.

Since $\overline{L_{\Sigma_k^p}}$ is complete for Π_k^p, it follows that $\Sigma_k^p = \Pi_k^p = \mathrm{PH}$. □

We conclude this section with three remarks. First, if one is fond of the truth-table version of bounded query hierarchies, one can certainly replace the hypothesis of Theorem 2.1 with $\mathrm{P}_{1\text{-tt}}^{\Sigma_k^p} = \mathrm{P}_{2\text{-tt}}^{\Sigma_k^p}$ (both as this is an equivalent hypothesis and as it in any case clearly follows from Theorem 2.4). Indeed, one can equally well replace the hypothesis of Theorem 2.1 with the even weaker-looking hypothesis[4] $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p)$ (as this hypothesis is also in fact equivalent to the hypothesis of Theorem 2.1—just note that if $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p)$ then $\mathrm{DIFF}_2(\Sigma_k^p)$ is closed under complementation and thus equals the boolean hierarchy over Σ_k^p, see [CGH+88], and so in particular we then have $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p) = \mathrm{P}^{\Sigma_k^p[2]}$).

Of course, the two equivalences just mentioned—$\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{P}^{\Sigma_k^p[2]} \Leftrightarrow \mathrm{P}_{1\text{-tt}}^{\Sigma_k^p} = \mathrm{P}_{2\text{-tt}}^{\Sigma_k^p} \Leftrightarrow \mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p)$—are well known. However, Theorem 2.4 is sufficiently strong that it creates an equivalence that is quite new and somewhat surprising. We state it below as Corollary 2.6.

THEOREM 2.5. *For each $k > 2$ it holds that*

$$\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p) \cap \mathrm{coDIFF}_2(\Sigma_k^p) \Rightarrow \Sigma_k^p = \Pi_k^p = \mathrm{PH}.$$

Proof. Let $A \triangle B =_{\mathrm{def}} (A - B) \cup (B - A)$. Recalling that $k > 2$, it is not hard to see that $\mathrm{P}^{(\mathrm{NP},\Sigma_k^p)} \subseteq \mathrm{DIFF}_2(\Sigma_k^p)$. In particular, this holds due to Lemma 2.3, in light of the facts that (i) $\mathrm{DIFF}_2(\Sigma_k^p) = \{L \mid (\exists L_1 \in \Sigma_k^p)(\exists L_2 \in \Sigma_k^p)[L = L_1 \triangle L_2]\}$ (due to Köbler, Schöning, and Wagner [KSW87]—see the discussion just before Theorem 3.7) and (ii) $A \tilde{\Delta} B = \{\langle x, y\rangle \mid x \in A\} \triangle \{\langle x, y\rangle \mid y \in B\}$. So, since $\mathrm{P}^{(\mathrm{NP},\Sigma_k^p)}$ is closed under complementation, we have $\mathrm{P}^{\Sigma_k^p[1]} \subseteq \mathrm{P}^{(\mathrm{NP},\Sigma_k^p)} \subseteq \mathrm{DIFF}_2(\Sigma_k^p) \cap \mathrm{coDIFF}_2(\Sigma_k^p)$. However, this says, under the hypothesis of the theorem, that $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{P}^{(\mathrm{NP},\Sigma_k^p)}$, which itself, by Theorem 2.4, implies that $\Sigma_k^p = \Pi_k^p = \mathrm{PH}$. □

[4] Where $\mathrm{DIFF}_2(\mathcal{C}) =_{\mathrm{def}} \{L \mid (\exists L_1 \in \mathcal{C})(\exists L_2 \in \mathcal{C})[L = L_1 - L_2]\}$, and $\mathrm{co}\mathcal{C} =_{\mathrm{def}} \{L \mid \overline{L} \in \mathcal{C}\}$ (see Definition 3.1 for background).

COROLLARY 2.6. *For each $k > 2$ it holds that*

$$\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p) \cap \mathrm{coDIFF}_2(\Sigma_k^p) \Leftrightarrow \mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p).$$

Our second remark is that Theorem 2.1 implies that, for $k > 2$, if the bounded query hierarchy over Σ_k^p collapses to its $\mathrm{P}^{\Sigma_k^p[1]}$ level, then the bounded query hierarchy over Σ_k^p equals the polynomial hierarchy (this provides a partial affirmative answer to the issue of whether, when a bounded query hierarchy collapses, the polynomial hierarchy necessarily collapses to it; see [HRZ95, Problem 4]).

Third, in Lemma 2.3 and Theorem 2.4 we speak of classes of the form $\mathrm{P}^{(\Sigma_i^p, \Sigma_j^p)}$, $i \neq j$. It would be very natural to reason as follows: "$\mathrm{P}^{(\Sigma_i^p, \Sigma_j^p)}$, $i \neq j$, must equal $\mathrm{P}^{\Sigma^p_{\max(i,j)}[1]}$, as $\Sigma^p_{\max(i,j)}$ can easily solve any $\Sigma^p_{\min(i,j)}$ query 'strongly' using the $\Sigma^p_{\max(i,j)-1}$ oracle of its base NP machine and thus the hypothesis of Theorem 2.4 is trivially satisfied and so you in fact are claiming to prove, unconditionally, that $\mathrm{PH} = \Sigma_3^p$." This reasoning, though tempting, is wrong for the following somewhat subtle reason. Though it is true that, for example, $\mathrm{NP}^{\Sigma_q^p}$ can solve any Σ_q^p query and then can tackle any Σ^p_{q+1} query, it does not follow that $\mathrm{P}^{(\Sigma^p_{q+1}, \Sigma^p_q)} = \mathrm{P}^{\Sigma^p_{q+1}[1]}$. The problem is that the answer to the Σ_q^p query may *change the truth-table* the P transducer uses to evaluate the answer of the Σ^p_{q+1} query.

We mention that Buhrman and Fortnow [BF96], building on and extending our proof technique, have very recently obtained the $k = 2$ analogue of Theorem 2.1. They also prove that there are relativized worlds in which the $k = 1$ analogue of Theorem 2.1 fails. On the other hand, if one changes Theorem 2.1's left-hand-side classes to function classes, then the $k = 1$ analogue of the resulting claim does hold due to Krentel (see [Kre88, Theorem 4.2]): $\mathrm{FP}^{\mathrm{NP}[1]} = \mathrm{FP}^{\mathrm{NP}[2]} \Rightarrow \mathrm{P} = \mathrm{PH}$. Also, very recent work of Hemaspaandra, Hemaspaandra, and Hempel [HHH97], building on and extending the techniques of the present paper and those of Buhrman and Fortnow [BF96], has established the $k = 2$ analogue of Theorem 3.2.

3. General case. We now generalize the results of section 2 to the case of m-truth-table reductions. Though the results of this section are stronger than those of section 2, the proofs are somewhat more involved, and thus we suggest the reader first read section 2.

For clarity, we now describe the two key differences between the proofs in this section and those of section 2. (1) The completeness claims of section 2 were simpler. Here, we now need Lemma 3.5, which extends [BCO93, Lemma 8] with the trick of splitting a truth-table along a simple query's dimension in such a way that the induced one-dimension-lower truth-tables cause no problems. (2) The proof of Theorem 3.6 is quite analogous to the proof of Theorem 2.4, except (i) it is a bit harder to understand as one continuously has to parse the deeply nested set differences caused by the fact that we are now working in the difference hierarchy and (ii) the "input is an easy string" simulation is changed to account for a new problem, namely, that in the boolean hierarchy one models each language by a *collection* of machines (mimicking the nested difference structure of boolean hierarchy languages) and thus it is hard to ensure that these machines, when guessing an object, necessarily guess the same object (we solve this coordination problem by forcing them to each guess a lexicographically extreme object, and we argue that this can be accomplished within the computational power available).

The difference hierarchy was introduced by Cai et al. [CGH$^+$88, CGH$^+$89] and is defined below. Cai et al. studied the case $\mathcal{C} = \mathrm{NP}$, but a number of other cases have since been studied [BJY90, BCO93, HR97].

DEFINITION 3.1. *Let $\mathcal{C}$ be any complexity class.*

1. $\mathrm{DIFF}_1(\mathcal{C}) = \mathcal{C}$.
2. *For any $k \geq 1$,* $\mathrm{DIFF}_{k+1}(\mathcal{C}) = \{L \mid (\exists L_1 \in \mathcal{C})(\exists L_2 \in \mathrm{DIFF}_k(\mathcal{C}))[L = L_1 - L_2]\}$.
3. *For any $k \geq 1$,* $\mathrm{coDIFF}_k(\mathcal{C}) = \{L \mid \overline{L} \in \mathrm{DIFF}_k(\mathcal{C})\}$.

Note in particular that

$$\mathrm{DIFF}_m(\Sigma_k^p) \cup \mathrm{coDIFF}_m(\Sigma_k^p) \subseteq \mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} \subseteq \mathrm{DIFF}_{m+1}(\Sigma_k^p) \cap \mathrm{coDIFF}_{m+1}(\Sigma_k^p).$$

THEOREM 3.2. *For each $m > 0$ and each $k > 2$ it holds that*

$$\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{P}_{m+1\text{-tt}}^{\Sigma_k^p} \Rightarrow \mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p).$$

Theorem 2.1 is the $m = 1$ case of Theorem 3.2 (except the former is stated in terms of Turing access). Theorem 3.2 follows immediately from Theorem 3.6 below, which states that, for $i < j < k$ and $i < k-2$, if one query to the ith and m queries to the kth levels of the polynomial hierarchy equals one query to the jth and m queries to the kth levels of the polynomial hierarchy, then $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$. Note, of course, that the conclusion of Theorem 3.2 implies a collapse of the polynomial hierarchy. In particular, via [BCO93, Theorem 10], Theorem 3.2 implies that, for each $m \geq 0$ and each $k > 2$, it holds that if $\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{P}_{m+1\text{-tt}}^{\Sigma_k^p}$ then the polynomial hierarchy can be solved by a P machine that makes $m-1$ truth-table queries to Σ_{k+1}^p and that in addition is allowed unbounded queries to Σ_k^p. This polynomial hierarchy collapse is about one level lower in the difference hierarchy over Σ_{k+1}^p than one could conclude from previous papers, in particular, from Beigel, Chang, and Ogiwara. That is, in light of Theorem 3.2, we have the following corollary, given here in a form that corrects a misstatement in an earlier version of this paper. (In fact, one can claim a slightly stronger collapse; see [HHH] for full details.)

COROLLARY 3.3. *For each $m \geq 0$ and each $k > 2$ it holds that* $\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{P}_{m+1\text{-tt}}^{\Sigma_k^p} \Rightarrow \mathrm{PH} = \mathrm{P}_{1,m-1\text{-tt}}^{(\Delta_{k+1}^p, \Sigma_{k+1}^p)}$.

The following definition will be useful.

DEFINITION 3.4. *Let $M_{a,b\text{-tt}}^{(A,B)}$ denote DPTM M making, simultaneously (i.e., all $a+b$ queries are made at the same time, in the standard truth-table fashion), at most a queries to oracle A and at most b queries to oracle B, and let*

$$\mathrm{P}_{a,b\text{-tt}}^{(\mathcal{C},\mathcal{D})} = \{L \subseteq \Sigma^* \mid (\exists C \in \mathcal{C})(\exists D \in \mathcal{D})(\exists \text{DPTM } M)[L = L(M_{a,b\text{-tt}}^{(C,D)})]\}.$$

LEMMA 3.5. *Let $m > 0$, let $0 \leq i < k$, let $L_{\mathrm{P}^{\Sigma_i^p[1]}}$ be any set $\leq_{\mathrm{m}}^p$-complete for $\mathrm{P}^{\Sigma_i^p[1]}$, and let $L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ be any language $\leq_{\mathrm{m}}^p$-complete for $\mathrm{DIFF}_m(\Sigma_k^p)$. Then $L_{\mathrm{P}^{\Sigma_i^p[1]}} \tilde{\Delta} L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ is $\leq_{\mathrm{m}}^p$-complete for $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p, \Sigma_k^p)}$.*

Lemma 3.5 does not require proof, as it is a use of the standard mind-change technique, and is analogous to [BCO93, Lemma 8], with one key twist that we now discuss. Assume, without loss of generality, that we focus on $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p, \Sigma_k^p)}$ machines that always make exactly $m+1$ queries. Regarding any such machine accepting a set complete for the class $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p, \Sigma_k^p)}$ of Lemma 3.5, we have on each input a truth-table with $m+1$ variables. Note that if one knows the answer to the one Σ_i^p query, then this induces a truth-table on m variables; however, note also that the two m-variable truth-tables (one corresponding to a "yes" answer to the Σ_i^p query and the other to

a "no" answer) may differ sharply. Regarding $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\mathrm{DIFF}_m(\Sigma_k^p)}$, we use $L_{\mathrm{P}^{\Sigma_i^p[1]}}$ to determine whether the m-variable truth-table induced by the true answer to the one Σ_i^p query accepts or not when all the Σ_k^p queries get the answer no. This use is analogous to [BCO93, Lemma 8]. The new twist is the action of the $L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ part of $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\mathrm{DIFF}_m(\Sigma_k^p)}$. We use this, just as in [BCO93, Lemma 8], to find whether or not we are in an odd mind-change region *but now with respect to the m-variable truth-table induced by the true answer to the one Σ_i^p query*. Crucially, this still is a $\mathrm{DIFF}_m(\Sigma_k^p)$ issue as, since $i < k$, a Σ_k^p machine can first on its own (by its base NP machine making one deterministic query to its Σ_{k-1}^p oracle) determine the true answer to the one Σ_i^p query, and thus the machine can easily determine which of the two m-variable truth-table cases it is in, and thus it plays its standard part in determining if the mind-change region of the m true answers to the Σ_k^p queries falls in an odd mind-change region *with respect to the correct m-variable truth-table*.

THEOREM 3.6. *Let $m > 0$, $0 \le i < j < k$, and $i < k-2$. If $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p,\Sigma_k^p)} = \mathrm{P}_{1,m\text{-tt}}^{(\Sigma_j^p,\Sigma_k^p)}$ then $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$.*

Proof. Suppose $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p,\Sigma_k^p)} = \mathrm{P}_{1,m\text{-tt}}^{(\Sigma_j^p,\Sigma_k^p)}$. Let $L_{\mathrm{P}^{\Sigma_i^p[1]}}$, $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}$, and $L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ be $\le_{\mathrm{m}}^p$-complete for $\mathrm{P}^{\Sigma_i^p[1]}$, $\mathrm{P}^{\Sigma_{i+1}^p[1]}$, and $\mathrm{DIFF}_m(\Sigma_k^p)$, respectively; such languages exist, e.g., via the standard canonical complete set constructions using enumerations of clocked machines. From Lemma 3.5 it follows that $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ is $\le_{\mathrm{m}}^p$-complete for $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p,\Sigma_k^p)}$. Since (as $i < j$) $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\tilde{\Delta}L_{\mathrm{DIFF}_m(\Sigma_k^p)} \in \mathrm{P}_{1,m\text{-tt}}^{(\Sigma_j^p,\Sigma_k^p)}$, and by assumption $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_j^p,\Sigma_k^p)} = \mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p,\Sigma_k^p)}$, there exists a polynomial-time many-one reduction h from $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\tilde{\Delta}L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ to $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\mathrm{DIFF}_m(\Sigma_k^p)}$. So, for all $x_1, x_2 \in \Sigma^*$,

if $h(\langle x_1, x_2\rangle) = \langle y_1, y_2\rangle$,
then

$(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow x_2 \in L_{\mathrm{DIFF}_m(\Sigma_k^p)})$ if and only if $(y_1 \in L_{\mathrm{P}^{\Sigma_i^p[1]}} \Leftrightarrow y_2 \in L_{\mathrm{DIFF}_m(\Sigma_k^p)})$.

We can use h to recognize some of $\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ by a $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm. In particular, we say that a string x is *easy for length n* if there exists a string x_1 such that $|x_1| \le n$ and $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \notin L_{\mathrm{P}^{\Sigma_i^p[1]}})$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$.

Let p be a fixed polynomial, which will be exactly specified later in the proof. We have the following algorithm to test whether $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ in the case that (our input) x is an easy string for $p(|x|)$. On input x, guess x_1 with $|x_1| \le p(|x|)$, let $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$, and accept if and only if $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \notin L_{\mathrm{P}^{\Sigma_i^p[1]}})$ and $y_2 \in L_{\mathrm{DIFF}_m(\Sigma_k^p)}$. This algorithm is not necessarily a $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm, but it does inspire the following $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm to test whether $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ in the case that x is an easy string for $p(|x|)$. Let $L_1, L_2, \ldots, L_m$ be languages in Σ_k^p such that $L_{\mathrm{DIFF}_m(\Sigma_k^p)} = L_1 - (L_2 - (L_3 - \cdots (L_{m-1} - L_m)\cdots))$. Then $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ if and only if $x \in L_1' - (L_2' - (L_3' - \cdots (L_{m-1}' - L_m')\cdots))$, where L_r' is computed as follows: on input x, guess x_1 with $|x_1| \le p(|x|)$, let $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$, and accept if and only if (a) $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \notin L_{\mathrm{P}^{\Sigma_i^p[1]}})$, (b) $(\forall z <_{lex} x_1)[z \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow w_1 \in L_{\mathrm{P}^{\Sigma_i^p[1]}}]$, where $h(\langle z, x\rangle) = \langle w_1, w_2\rangle$, and (c) $y_2 \in L_r$.

Since $i + 2 < k$, $L_r' \in \Sigma_k^p$, and thus our algorithm is in $\mathrm{DIFF}_m(\Sigma_k^p)$. Note that condition (b) has no analogue in the proof of Theorem 2.4. We need this extra condition here as otherwise the different L_r' might latch onto *different* strings x_1 and this would cause unpredictable behavior (as different x_1's would create different y_2's).

We say that x is *hard for length* n if $|x| \leq n$ and x is not easy for length n, i.e., if $|x| \leq n$ and for all x_1 with $|x_1| \leq n$ it holds that $x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \in L_{\mathrm{P}^{\Sigma_i^p[1]}}$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$.

If x is a hard string for length n, then x induces a many-one reduction from $(L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}})^{\leq n}$ to $L_{\mathrm{P}^{\Sigma_i^p[1]}}$, namely, $f(x_1) = y_1$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$. Note that f is computable in time polynomial in $\max(n, |x_1|)$.

We can use hard strings to obtain a $\mathrm{DIFF}_m(\Sigma_{k-1}^p)$ algorithm for $L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ and thus (since $\mathrm{DIFF}_m(\Sigma_{k-1}^p) \subseteq \mathrm{P}^{\Sigma_{k-1}^p} \subseteq \Sigma_k^p \cap \Pi_k^p$) certainly a $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm for $\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$. Again, let $L_1, L_2, \ldots, L_m$ be languages in Σ_k^p such that $L_{\mathrm{DIFF}_m(\Sigma_k^p)} = L_1 - (L_2 - (L_3 - \cdots (L_{m-1} - L_m)\cdots))$. For all $1 \leq r \leq m$, let M_r be a Σ_{k-i-1}^p machine such that $L_r = L(M_r^Y)$, where $Y = L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}$. Let the run-time of all M_r's be bounded by polynomial p, which without loss of generality satisfies $(\forall \widehat{m} \geq 0)[p(\widehat{m}+1) > p(\widehat{m}) > 0]$ (as promised above, we have now specified p). Then for all $1 \leq r \leq m$,

$$(L_r)^{=n} = (L(M_r^{(Y^{\leq p(n)})}))^{=n},$$

where $Y = L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}$. If there exists a hard string for length $p(n)$, then this hard string induces a reduction from $(L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}})^{\leq p(n)}$ to $L_{\mathrm{P}^{\Sigma_i^p[1]}}$. Thus, with any hard string for length $p(n)$ in hand, call it w_n, $\widehat{M_r}$ with oracle $L_{\mathrm{P}^{\Sigma_i^p[1]}}$ recognizes L_r for strings of length n, where $\widehat{M_r}$ is the machine that simulates M_r but replaces each query q by the first component of $h(\langle q, w_n\rangle)$. It follows that if there exists a hard string for length $p(n)$, then this string induces a $\mathrm{DIFF}_m(\Sigma_{k-1}^p)$ algorithm for $(L_{\mathrm{DIFF}_m(\Sigma_k^p)})^{=n}$, and therefore certainly a $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm for $(\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}})^{=n}$. It follows that there exist m Σ_k^p sets, say, $\widehat{L_r}$ for $1 \leq r \leq m$, such that the following holds: for all x, if x (functioning as $w_{|x|}$ above) is a hard string for length $p(|x|)$, then $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ if and only if $x \in \widehat{L_1} - (\widehat{L_2} - (\widehat{L_3} - \cdots (\widehat{L_{m-1}} - \widehat{L_m})\cdots))$.

However, now we have an outright $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm for $\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$: for $1 \leq r \leq m$ define an $\mathrm{NP}^{\Sigma_{k-1}^p}$ machine N_r as follows: on input x, the NP base machine of N_r executes the following algorithm.

1. Using its Σ_{k-1}^p oracle, it deterministically determines whether the input x is an easy string for length $p(|x|)$. This can be done, as checking whether the input is an easy string for length $p(|x|)$ can be done by one query to Σ_{i+2}^p, and $i+2 \leq k-1$ by our $i < k-2$ hypothesis.
2. If the previous step determined that the input is not an easy string, then the input must be a hard string for length $p(|x|)$. So simulate the Σ_k^p algorithm for $\widehat{L_r}$ induced by this hard string (i.e., the input x itself) on input x (via our NP machine itself simulating the base level of the Σ_k^p algorithm and using the NP machine's oracle to simulate the oracle queries made by the base level NP machine of the Σ_k^p algorithm being simulated).
3. If the first step determined that the input x is easy for length $p(|x|)$, then our NP machine simulates (using itself and its oracle) the Σ_k^p algorithm for L_r' on input x.

It follows that for all x, $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ if and only if $x \in L(N_1) - (L(N_2) - (L(N_3) - \cdots (L(N_{m-1}) - L(N_m))\cdots))$. Since $\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ is complete for $\mathrm{coDIFF}_m(\Sigma_k^p)$, it follows that $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$. □

Finally, we note that we have analogues of Theorem 2.5 and Corollary 2.6. The proof is analogous to that of Theorem 2.5; one just uses $\mathrm{P}_{1,m\text{-tt}}^{(\mathrm{NP},\Sigma_k^p)}$ in the way $\mathrm{P}^{(\mathrm{NP},\Sigma_k^p)}$ was used in that proof and again invokes the relation between the difference and symmetric difference hierarchies (namely, that $\mathrm{DIFF}_j(\Sigma_k^p)$ is exactly the class of sets L that for some $L_1, \ldots, L_j \in \Sigma_k^p$ satisfy $L = L_1 \triangle \cdots \triangle L_j$; this well-known equality is due to [KSW87, section 3] in light of the standard equalities regarding boolean hierarchies (see [CGH+88, section 2.1]); although both [KSW87] and [CGH+88] focus mostly on the $k = 1$ case, it is standard [Wec85, BBJ+89] that the equalities in fact hold for any class closed under union and intersection and containing $\emptyset$ and Σ^*).

THEOREM 3.7. *Let $m \geq 0$ and $k > 2$. If $\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{DIFF}_{m+1}(\Sigma_k^p) \cap \mathrm{coDIFF}_{m+1}(\Sigma_k^p)$ then $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$.*

COROLLARY 3.8. *For each $k > 2$ and $m \geq 0$, it holds that*

$$\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{DIFF}_{m+1}(\Sigma_k^p) \cap \mathrm{coDIFF}_{m+1}(\Sigma_k^p) \Leftrightarrow \mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{DIFF}_{m+1}(\Sigma_k^p).$$

Acknowledgments. The first two authors thank Gerd Wechsung's research group for its very kind hospitality during the visit when this research was performed. The authors are grateful to two anonymous referees, Lance Fortnow, and Jörg Rothe for helpful comments and suggestions.

REFERENCES

[All86] E. ALLENDER, *The complexity of sparse sets in P*, in Proc. 1st Structure in Complexity Theory Conference, Lecture Notes in Computer Science 223, Springer-Verlag, Berlin, 1986, pp. 1–11.

[All91] E. ALLENDER, *Limitations of the upward separation technique*, Math. Systems Theory, 24 (1991), pp. 53–67.

[AW90] E. ALLENDER AND C. WILSON, *Downward translations of equality*, Theoret. Comput. Sci., 75 (1990), pp. 335–346.

[BBJ+89] A. BERTONI, D. BRUSCHI, D. JOSEPH, M. SITHARAM, AND P. YOUNG, *Generalized boolean hierarchies and boolean hierarchies over RP*, in Proc. 7th Conference on Fundamentals of Computation Theory, Lecture Notes in Computer Science 380, Springer-Verlag, Berlin, 1989, pp. 35–46.

[BCO93] R. BEIGEL, R. CHANG, AND M. OGIWARA, *A relationship between difference hierarchies and relativized polynomial hierarchies*, Math. Systems Theory, 26 (1993), pp. 293–310.

[BF96] H. BUHRMAN AND L. FORTNOW, *Two Queries*, Technical Report 96-20, Department of Computer Science, University of Chicago, Chicago, IL, September 1996.

[BFNW93] L. BABAI, L. FORTNOW, N. NISAN, AND A. WIGDERSON, *BPP has subexponential time simulations unless EXPTIME has publishable proofs*, Comput. Complexity, 3 (1993), pp. 307–318.

[BJY90] D. BRUSCHI, D. JOSEPH, AND P. YOUNG, *Strong separations for the boolean hierarchy over RP*, Internat. J. Found. Comput. Sci., 1 (1990), pp. 201–218.

[Boo74] R. BOOK, *Tally languages and complexity classes*, Inform. and Control, 26 (1974), pp. 186–193.

[CGH+88] J. CAI, T. GUNDERMANN, J. HARTMANIS, L. HEMACHANDRA, V. SEWELSON, K. WAGNER, AND G. WECHSUNG, *The boolean hierarchy* I: *Structural properties*, SIAM J. Comput., 17 (1988), pp. 1232–1252.

[CGH+89] J. CAI, T. GUNDERMANN, J. HARTMANIS, L. HEMACHANDRA, V. SEWELSON, K. WAGNER, AND G. WECHSUNG, *The boolean hierarchy* II: *Applications*, SIAM J. Comput., 18 (1989), pp. 95–111.

[Cha91] R. CHANG, *On the Structure of NP Computations under Boolean Operators*, Ph.D. thesis, Cornell University, Ithaca, NY, 1991.

[CK96] R. CHANG AND J. KADIN, *The boolean hierarchy and the polynomial hierarchy: A closer connection*, SIAM J. Comput., 25 (1996), pp. 340–354.

[HHH] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel, *What's Up with Downward Collapse: Using the Easy-Hard Technique to Link Boolean and Polynomial Hierarchy Collapses*, manuscript, 1997.

[HHH97] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel, *Translating Equality Downwards*, Technical Report TR-657, Department of Computer Science, University of Rochester, Rochester, NY, April 1997.

[HIS85] J. Hartmanis, N. Immerman, and V. Sewelson, *Sparse sets in NP−P: EXPTIME versus NEXPTIME*, Inform. and Control, 65 (1985), pp. 159–181.

[HJ95] L. Hemaspaandra and S. Jha, *Defying upward and downward separation*, Inform. and Comput., 121 (1995), pp. 1–13.

[HR97] L. Hemaspaandra and J. Rothe, *Unambiguous computation: Boolean hierarchies and sparse Turing-complete sets*, SIAM J. Comput., 26 (1997), pp. 634–653.

[HRW] L. Hemaspaandra, J. Rothe, and G. Wechsung, *Easy sets and hard certificate schemes*, Acta Informatica, to appear.

[HRZ95] L. Hemaspaandra, A. Ramachandran, and M. Zimand, *Worlds to die for*, SIGACT News, 26 (1995), pp. 5–15.

[HY84] J. Hartmanis and Y. Yesha, *Computation times of NP sets of different densities*, Theoret. Comput. Sci., 34 (1984), pp. 17–32.

[Kad88] J. Kadin, *The polynomial time hierarchy collapses if the boolean hierarchy collapses*, SIAM J. Comput., 17 (1988), pp. 1263–1282; Erratum, 20(1991), p. 404.

[Ko82] K. Ko, *Some observations on probabilistic algorithms and NP-hard problems*, Inform. Process. Lett., 14 (1982), pp. 39–43.

[Ko89] K. Ko, *Relativized polynomial time hierarchies having exactly k levels*, SIAM J. Comput., 18 (1989), pp. 392–408.

[Kre88] M. Krentel, *The complexity of optimization problems*, J. Comput. System Sci., 36 (1988), pp. 490–509.

[KSW87] J. Köbler, U. Schöning, and K. Wagner, *The difference and truth-table hierarchies for NP*, RAIRO Theoretical Informatics and Applications, 21 (1987), pp. 419–435.

[RRW94] R. Rao, J. Rothe, and O. Watanabe, *Upward separation for FewP and related classes*, Inform. Process. Lett., 52 (1994), pp. 175–180.

[Wec85] G. Wechsung, *On the boolean closure of NP*, in Proc. 5th Conference on Fundamentals of Computation Theory, Lecture Notes in Computer Science 199, Springer-Verlag, Berlin, 1985, pp. 485–493. (An unpublished precursor of this paper was coauthored by K. Wagner.)

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 394–408

GENERICITY, RANDOMNESS, AND POLYNOMIAL-TIME APPROXIMATIONS*

YONGGE WANG†

Abstract. Polynomial-time safe and unsafe approximations for intractable sets were introduced by Meyer and Paterson [Technical Report TM-126, Laboratory for Computer Science, MIT, Cambridge, MA, 1979] and Yesha [*SIAM J. Comput.*, 12 (1983), pp. 411–425], respectively. The question of which sets have optimal safe and unsafe approximations has been investigated extensively. Duris and Rolim [*Lecture Notes in Comput. Sci.* 841, Springer-Verlag, Berlin, New York, 1994, pp. 38–51] and Ambos-Spies [*Proc. 22nd ICALP*, Springer-Verlag, Berlin, New York, 1995, pp. 384–392] showed that the existence of optimal polynomial-time approximations for the safe and unsafe cases is independent. Using the law of the iterated logarithm for p-random sequences (which has been recently proven in [*Proc. 11th Conf. Computational Complexity*, IEEE Computer Society Press, Piscataway, NJ, 1996, pp. 180–189]), we extend this observation by showing that both the class of polynomial-time Δ-levelable sets and the class of sets which have optimal polynomial-time unsafe approximations have p-measure 0. Hence typical sets in $\mathbf{E}$ (in the sense of p-measure) do not have optimal polynomial-time unsafe approximations. We will also establish the relationship between resource bounded genericity concepts and the polynomial-time safe and unsafe approximation concepts.

Key words. computational complexity, resource bounded genericity, resource bounded randomness, approximation

AMS subject classifications. 68Q05, 68Q25, 68Q30, 03D15, 60F99

PII. S009753979630235X

1. Introduction.

The notion of polynomial-time safe approximations was introduced by Meyer and Paterson in [13] (see also [8]). A safe approximation algorithm for a set A is a polynomial-time algorithm M that on each input x outputs either 1 (accept), 0 (reject), or ? (I do not know) such that all inputs accepted by M are members of A and no member of A is rejected by M. An approximation algorithm is optimal if no other polynomial-time algorithm correctly decides infinitely many more inputs, that is to say, outputs infinitely many more correct 1s or 0s. In Orponen, Russo, and Schöning [14], the existence of optimal approximations was phrased in terms of $\mathbf{P}$-levelability: a recursive set A is $\mathbf{P}$-levelable if for any deterministic Turing machine M accepting A and for any polynomial p there is another machine M' accepting A and a polynomial p' such that for infinitely many elements x of A, M does not accept x within $p(|x|)$ steps while M' accepts x within $p'(|x|)$ steps. It is easy to show that A has an optimal polynomial-time safe approximation if and only if neither A nor $\bar{A}$ is $\mathbf{P}$-levelable.

The notion of polynomial-time unsafe approximations was introduced by Yesha in [19]: an unsafe approximation algorithm for a set A is just a standard polynomial-time bounded deterministic Turing machine M with outputs 1 and 0. Note that, different from the polynomial-time safe approximations, here we are allowed to make errors, and we study the amount of inputs on which M are correct. Duris and Rolim [6] further investigated unsafe approximations and introduced a levelability concept, Δ-levelability, which implies the nonexistence of optimal polynomial-time unsafe approximations. They showed that complete sets for $\mathbf{E}$ are Δ-levelable and there exists

* Received by the editors April 22, 1996; accepted for publication (in revised form) March 4, 1997; published electronically July 7, 1998.
http://www.siam.org/journals/sicomp/28-2/30235.html
† Department of Computer Science, The University of Auckland, Private Bag 92019, Auckland, New Zealand (wang@cs.auckland.ac.nz).

an intractable set in **E** which has an optimal safe approximation but no optimal unsafe approximation. But they did not succeed in producing an intractable set with optimal unsafe approximations. Ambos-Spies [1] defined a concept of weak Δ-levelability and showed that there exists an intractable set in **E** which is not weakly Δ-levelable (hence it has an optimal unsafe approximation).

Like resource-bounded randomness concepts, different kinds of resource-bounded genericity concepts were introduced by Ambos-Spies [2], Ambos-Spies, Fleischhack, and Huwig [3], Fenner [7], and Lutz [9]. It has been proved that resource-bounded generic sets are useful in providing a coherent picture of complexity classes. These sets embody the method of diagonalization construction; that is, requirements which can always be satisfied by finite extensions are automatically satisfied by generic sets.

It was shown in Ambos-Spies, Neis, and Terwijn [4] that the generic sets of Ambos-Spies are **P**-immune, and that the class of sets which have optimal safe approximations is large in the sense of resource-bounded Ambos-Spies category. Mayordomo [11] has shown that the class of **P**-immune sets is neither meager nor comeager both in the sense of resource-bounded Lutz category and in the sense of resource-bounded Fenner category. We extend this result by showing that the class of sets which have optimal safe approximations is neither meager nor comeager both in the sense of resource-bounded Lutz category and in the sense of resource-bounded Fenner category. Moreover, we will show the following relations between unsafe approximations and resource-bounded categories.

1. The class of weakly Δ-levelable sets is neither meager nor comeager in the sense of resource-bounded Ambos-Spies category [4].
2. The class of weakly Δ-levelable sets is comeager (is therefore large) in the sense of resource-bounded general Ambos-Spies [2], Fenner [7], and Lutz [9] categories.
3. The class of Δ-levelable sets is neither meager nor comeager in the sense of resource-bounded general Ambos-Spies [2], Fenner [7], and Lutz [9] categories.

In the last section, we will show the relationship between polynomial-time approximations and p-measure. Mayordomo [12] has shown that the class of **P**-bi-immune sets has p-measure 1. It follows that the class of sets which have optimal polynomial-time safe approximations has p-measure 1. Using the law of the iterated logarithm for p-random sequences which we have proved in Wang [16, 17], we will show that the following hold.

1. The class of Δ-levelable sets has p-measure 0.
2. The class of sets which have optimal polynomial-time unsafe approximations have p-measure 0. That is, the class of weakly Δ-levelable sets has p-measure 1.
3. p-Random sets are weakly Δ-levelable but not Δ-levelable.

Hence typical sets in the sense of resource-bounded measure do not have optimal polynomial-time unsafe approximations.

It should be noted that the above results show that the class of weakly Δ-levelable sets is large both in the sense of the different notions of resource-bounded category and in the sense of resource-bounded measure. That is to say, typical sets in $\mathbf{E}_2$ (in the sense of resource-bounded category or in the sense of resource-bounded measure) are weakly Δ-levelable.

In contrast to the results in this paper, we have recently shown (in [18]) the following results.

1. There is a p-stochastic set $A \in \mathbf{E}_2$ which is Δ-levelable.

2. There is a p-stochastic set $A \in \mathbf{E}_2$ which has an optimal unsafe approximation.

2. Definitions. N and $Q(Q^+)$ are the set of natural numbers and the set of (nonnegative) rational numbers, respectively. $\Sigma = \{0,1\}$ is the binary alphabet, Σ^* is the set of (finite) binary strings, Σ^n is the set of binary strings of length n, and Σ^∞ is the set of infinite binary sequences. The length of a string x is denoted by $|x|$. $<$ is the length-lexicographical ordering on Σ^*, and z_n $(n \geq 0)$ is the nth string under this ordering. λ is the empty string. For strings $x, y \in \Sigma^*$, xy is the concatenation of x and y, $x \sqsubseteq y$ denotes that x is an initial segment of y. For a sequence $x \in \Sigma^* \cup \Sigma^\infty$ and an integer number $n \geq -1$, $x[0..n]$ denotes the initial segment of length $n+1$ of x ($x[0..n] = x$ if $|x| \leq n+1$) and $x[i]$ denotes the ith bit of x, i.e., $x[0..n] = x[0] \cdots x[n]$. Lowercase letters $\ldots, k, l, m, n, \ldots, x, y, z$ from the middle and the end of the alphabet will denote numbers and strings, respectively. The letter b is reserved for elements of Σ, and lowercase Greek letters $\xi, \eta, \ldots$ denote infinite sequences from Σ^∞.

A subset of Σ^* is called a language, a problem, or simply a set. Capital letters are used to denote subsets of Σ^* and boldface capital letters are used to denote subsets of Σ^∞. The cardinality of a language A is denoted by $\|A\|$. We identify a language A with its characteristic function, i.e., $x \in A$ if and only if $A(x) = 1$. The characteristic sequence of a language A is the infinite sequence $A(z_0)A(z_1)A(z_2)\cdots$. We freely identify a language with its characteristic sequence and the class of all languages with the set Σ^∞. For a language $A \subseteq \Sigma^*$ and a string $z_n \in \Sigma^*$, $A\restriction z_n = A(z_0)\cdots A(z_{n-1}) \in \Sigma^*$. For languages A and B, $\bar{A} = \Sigma^* - A$ is the complement of A, $A\Delta B = (A - B) \cup (B - A)$ is the symmetric difference of A and B; $A \subseteq B$ (resp., $A \subset B$) denotes that A is a subset of B (resp., $A \subseteq B$ and $B \not\subseteq A$). For a number n, $A^{=n} = \{x \in A : |x| = n\}$ and $A^{\leq n} = \{x \in A : |x| \leq n\}$.

We fix a standard polynomial-time computable and invertible pairing function $\lambda x, y\langle x, y\rangle$ on Σ^* such that, for every string x, there is a real $\alpha(x) > 0$ satisfying

$$\|\Sigma^{[x]} \cap \Sigma^n\| \geq \alpha(x) \cdot 2^n \text{ for almost all } n,$$

where $\Sigma^{[x]} = \{\langle x, y\rangle : y \in \Sigma^*\}$ and $\Sigma^{[\leq x]} = \{\langle x', y\rangle : x' \leq x \ \&\ y \in \Sigma^*\}$. We will use $\mathbf{P}$, $\mathbf{E}$, and $\mathbf{E}_2$ to denote the complexity classes $DTIME(poly)$, $DTIME(2^{linear})$, and $DTIME(2^{poly})$, respectively. Finally, we fix a recursive enumeration $\{P_e : e \geq 0\}$ of $\mathbf{P}$ such that $P_e(x)$ can be computed in $O(2^{|x|+e})$ steps (uniformly in e and x).

We define a *finite function* to be a partial function from Σ^* to Σ whose domain is finite. For a finite function σ and a string $x \in \Sigma^*$, we write $\sigma(x) \downarrow$ if $x \in dom(\sigma)$, and $\sigma(x) \uparrow$ otherwise. For two finite functions σ, τ, we say σ and τ are compatible if $\sigma(x) = \tau(x)$ for all $x \in dom(\sigma) \cap dom(\tau)$. The concatenation $\sigma\tau$ of two finite functions σ and τ is defined as $\sigma\tau = \sigma \cup \{(z_{n_\sigma+i+1}, b) : z_i \in dom(\tau) \ \&\ \tau(z_i) = b\}$, where $n_\sigma = \max\{n : z_n \in dom(\sigma)\}$ and $n_\sigma = -1$ for $\sigma = \lambda$. For a set A and a string x, we identify the characteristic string $A\restriction x$ with the finite function $\{(y, A(y)) : y < x\}$. For a finite function σ and a set A, σ is extended by A if for all $x \in dom(\sigma)$, $\sigma(x) = A(x)$.

3. Genericity versus polynomial-time safe approximations. In this section, we summarize some known results on the relationship between the different notions of resource-bounded genericity and the notion of polynomial-time safe approximations.

We first introduce some concepts of resource-bounded genericity.

DEFINITION 3.1. *A partial function f from Σ^* to $\{\sigma : \sigma$ is a finite function $\}$ is* dense along *a set A if there are infinitely many strings x such that $f(A\restriction x)$ is defined.*

A set A meets *f if, for some x, the finite function $(A\upharpoonright x)f(A\upharpoonright x)$ is extended by A. Otherwise, A* avoids *f.*

DEFINITION 3.2. *A class $\mathbf{C}$ of sets is* nowhere dense *via f if f is dense along all sets in $\mathbf{C}$ and for every set $A \in \mathbf{C}$, A avoids f.*

DEFINITION 3.3. *Let $\mathbf{F}$ be a class of (partial) functions from Σ^* to $\{\sigma : \sigma$ is a finite function$\}$. A class $\mathbf{C}$ of sets is* $\mathbf{F}$-meager *if there exists a function $f \in \mathbf{F}$ such that $\mathbf{C} = \cup_{i\in N}\mathbf{C}_i$ and $\mathbf{C}_i$ is nowhere dense via $f_i(x) = f(\langle i, x\rangle)$. A class $\mathbf{C}$ of sets is* $\mathbf{F}$-comeager *if $\bar{\mathbf{C}}$ is $\mathbf{F}$-meager.*

DEFINITION 3.4. *A set G is* $\mathbf{F}$-generic *if G is an element of all $\mathbf{F}$-comeager classes.*

LEMMA 3.5 (see [2, 7, 9]). *A set G is $\mathbf{F}$-generic if and only if G meets all functions $f \in \mathbf{F}$ which are dense along G.*

For a class $\mathbf{F}$ of functions, each function $f \in \mathbf{F}$ can be considered as a finitary property $\mathcal{P}$ of sets. If $f(A\upharpoonright x)$ is defined, then all sets extending $(A\upharpoonright x)f(A\upharpoonright x)$ have the property $\mathcal{P}$. So a set A has the property $\mathcal{P}$ if and only if A meets f. f is dense along A if and only if in a construction of A along the ordering $<$, where at stage s of the construction we decide whether or not the string z_s belongs to A, there are infinitely many stages s such that by appropriately defining $A(z_s)$ we can ensure that A has the property $\mathcal{P}$ (that is to say, for some string x, $(A\upharpoonright x)f(A\upharpoonright x)$ is extended by A).

For different function classes $\mathbf{F}$, we have different notions of $\mathbf{F}$-genericity. In this paper, we will concentrate on the following four kinds of function classes which have been investigated by Ambos-Spies [2], Amos-Spies, Neis, and Terwijn [4], Fenner [7], and Lutz [9], respectively. $\mathbf{F}_1$ is the class of polynomial-time computable partial functions from Σ^* to Σ; $\mathbf{F}_2$ is the class of polynomial-time computable partial functions from Σ^* to $\{\sigma : \sigma$ is a finite function$\}$; $\mathbf{F}_3$ is the class of polynomial-time computable total functions from Σ^* to $\{\sigma : \sigma$ is a finite function$\}$; and $\mathbf{F}_4$ is the class of polynomial-time computable total functions from Σ^* to Σ^*.

DEFINITION 3.6.

1. (*See Ambos-Spies, Neis, and Terwijn* [4].) *A set G is* A-generic *if G is $\mathbf{F}_1$-generic.*
2. (*See Ambos-Spies* [2].) *A set G is* general A-generic *if G is $\mathbf{F}_2$-generic.*
3. (*See Fenner* [7].) *A set G is* F-generic *if G is $\mathbf{F}_3$-generic.*
4. (*See Lutz* [9].) *A set G is* L-generic *if G is $\mathbf{F}_4$-generic.*

Obviously, we have the following implications.

THEOREM 3.7.

1. *If a set G is general A-generic, then G is A-generic, F-generic, and L-generic.*
2. *If a set G is F-generic, then G is L-generic.*

Proof. The proof is straightforward. □

In this paper, we will also study the following n^k-time ($k > 1$) bounded genericity concepts. A set G is Ambos-Spies n^k-generic (resp., general Ambos-Spies n^k-generic, Fenner n^k-generic, Lutz n^k-generic) if and only if G meets all n^k-time computable functions $f \in \mathbf{F}_1$ (resp., $\mathbf{F}_2$, $\mathbf{F}_3$, $\mathbf{F}_4$) which are dense along G.

THEOREM 3.8 (see Ambos-Spies [2]). *A class $\mathbf{C}$ of sets is meager in the sense of Ambos-Spies category (resp., general Ambos-Spies category, Fenner category, Lutz Category) if and only if there exists a number $k \in N$ such that there is no Ambos-Spies n^k-generic (resp., general Ambos-Spies n^k-generic, Lutz n^k-generic, Fenner n^k-generic) set in $\mathbf{C}$.*

As an example, we show that Ambos-Spies n-generic sets are $\mathbf{P}$-immune.

THEOREM 3.9 (see Ambos-Spies, Neis, Terwijn [4]). *Let G be an Ambos-Spies n-generic set. Then G is* $\mathbf{P}$*-immune.*

Proof. For a contradiction assume that $A \in \mathbf{P}$ is an infinite subset of G. Then the function $f : \Sigma^* \to \Sigma$ defined by

$$f(x) = \begin{cases} 0 & z_{|x|} \in A, \\ \uparrow & z_{|x|} \notin A \end{cases}$$

is computable in time n and is dense along G. So, by the Ambos-Spies n-genericity of G, G meets f. By the definition of f, this implies that there exists some string $z_i \in A$ such that $z_i \notin G$, a contradiction. □

It has been shown (see Mayordomo [12]) that neither F-genericity nor L-genericity implies $\mathbf{P}$-immunity or non-$\mathbf{P}$-immunity.

A *partial* set A is defined by a partial characteristic function $f : \Sigma^* \to \Sigma$. A partial set A is polynomial-time computable if $dom(A) \in \mathbf{P}$ and its partial characteristic function is computable in polynomial time.

DEFINITION 3.10 (see Meyer and Paterson [13]). *A* polynomial-time safe approximation *of a set A is a polynomial-time computable partial set Q which is consistent with A, that is to say, for every string $x \in dom(Q)$, $A(x) = Q(x)$. The approximation Q is* optimal *if, for every polynomial-time safe approximation Q' of A, $dom(Q') - dom(Q)$ is finite.*

DEFINITION 3.11 (see Orponen, Russo, and Schöning [14]). *A set A is* $\mathbf{P}$-levelable *if, for any subset $B \in \mathbf{P}$ of A, there is another subset $B' \in \mathbf{P}$ of A such that $\|B' - B\| = \infty$.*

LEMMA 3.12 (see Orponen, Russo, and Schöning [14]). *A set A possesses an optimal polynomial-time safe approximation if and only if neither A nor $\bar{A}$ is* $\mathbf{P}$*-levelable.*

Proof. The proof is straightforward. □

LEMMA 3.13. *If a set A is* $\mathbf{P}$*-immune, then A is not* $\mathbf{P}$*-levelable.*

Proof. The proof is straightforward. □

THEOREM 3.14 (see Ambos-Spies [2]). *Let G be an Ambos-Spies n-generic set. Then neither G nor $\bar{G}$ is* $\mathbf{P}$*-levelable. That is to say, G has an optimal polynomial-time safe approximation.*

Proof. This follows from Theorem 3.9. □

Theorem 3.14 shows that the class of $\mathbf{P}$-levelable sets is "small" in the sense of resource-bounded (general) Ambos-Spies category.

COROLLARY 3.15. *The class of* $\mathbf{P}$*-levelable sets is meager in the sense of resource-bounded (general) Ambos-Spies category.*

Now we show that the class of $\mathbf{P}$-levelable sets is neither meager nor comeager in the sense of resource-bounded Fenner category and Lutz category.

THEOREM 3.16.

1. *There exists a set G in $\mathbf{E}_2$ which is both F-generic and* $\mathbf{P}$*-levelable.*
2. *There exists a set G in $\mathbf{E}_2$ which is F-generic but not* $\mathbf{P}$*-levelable.*

Proof. 1. Let $\delta(0) = 0, \delta(n+1) = 2^{2^{\delta(n)}}$, $I_1 = \{x : \delta(2n) \leq |x| < \delta(2n+1), n \in N\}$, $I_2 = \Sigma^* - I_1$, and $\{f_i : i \in N\}$ be an enumeration of $\mathbf{F}_3$ such that $f_i(x)$ can be computed uniformly in time $2^{\log^k(|x|+i)}$ for some $k \in N$.

In the following, we construct a set G in stages which is both F-generic and $\mathbf{P}$-levelable. In the construction we will ensure that

$$G \cap \Sigma^{[e]} \cap I_1 =^* \Sigma^{[e]} \cap I_1$$

for $e \geq 0$. Hence $G \cap \Sigma^{[e]} \cap I_1 \in \mathbf{P}$ for $e \geq 0$. In order to ensure that G is $\mathbf{P}$-levelable, it suffices to satisfy for all $e \geq 0$ the following requirements:

$L_e : P_e \subseteq G \cap I_1 \Rightarrow P_e \subseteq^* \Sigma^{[\leq e]} \cap I_1$.

To show that the requirements $L_e (e \geq 0)$ ensure that G is $\mathbf{P}$-levelable (fix a subset $C \in \mathbf{P}$ of G) we have to define a subset $C' \in \mathbf{P}$ of G such that $C' - C$ is infinite. Fix e such that $P_e = C \cap I_1$. Then, by the requirement L_e, $C \cap I_1 \subseteq^* \Sigma^{[\leq e]} \cap I_1$. So, for $C' = G \cap \Sigma^{[e+1]} \cap I_1$, $C' \in \mathbf{P}$ and C' is infinite. Since $C' \cap C = \emptyset$, C' has the required property.

The strategy for meeting a requirement L_e is as follows: if there is a string $x \in (I_1 \cap P_e) - \Sigma^{[\leq e]}$, then we let $G(x) = 0$ to refute the hypothesis of the requirement L_e (so L_e is trivially met). To ensure that G is F-generic, it suffices to meet for all $e \geq 0$ the following requirements:

G_e: There exists a string x such that G extends $(G\restriction x)f_e(G\restriction x)$.

Because the set I_1 is used to satisfy L_e, we will use I_2 to satisfy G_e. The strategy for meeting a requirement G_e is as follows: for some string $x \in I_2$, let G extend $(G\restriction x)f_e(G\restriction x)$.

Define a priority ordering of the requirements by letting $R_{2n} = G_n$ and $R_{2n+1} = L_n$. Now we give the construction of G formally.

Stage s.

If $G(z_s)$ has been defined before stage s, then go to stage $s + 1$.

A requirement L_e *requires* attention if

1. $e < s$.
2. $z_s \in P_e \cap \Sigma^{[>e]} \cap I_1$.
3. For all $y < z_s$, if $y \in P_e$ then $y \in G \cap I_1$.

A requirement G_e *requires* attention if $e < s$, G_e has not received attention yet, and $x \in I_2$ for all $z_s \leq x \leq z_t$ where z_t is the greatest element in $dom((G\restriction z_s)f_e(G\restriction z_s))$.

Fix the minimal n such that R_n requires attention. If there is no such n, then let $G(z_s) = 1$. Otherwise, we say that R_n *receives* attention. Moreover, if $R_n = L_e$ then let $G(z_s) = 0$. If $R_n = G_e$ then let $G\restriction z_{t+1} = fill_1((G\restriction z_s)f_e(G\restriction z_s), t)$, where z_t is the greatest element in $dom((G\restriction z_s)f_e(G\restriction z_s))$ and for a finite function σ and a number k, $fill_1(\sigma, k) = \sigma \cup \{(x, 1) : x \leq z_k \ \& \ x \notin dom(\sigma)\}$.

This completes the construction of G.

It is easy to verify that the set G constructed above is both $\mathbf{P}$-levelable and F-generic; the details are omitted here.

2. For a general A-generic set G, by Theorem 3.9, G is $\mathbf{P}$-immune. By Theorem 3.7, G is F-generic. Hence, G is F-generic but not $\mathbf{P}$-levelable. □

COROLLARY 3.17. *The class of* $\mathbf{P}$*-levelable sets is neither meager nor comeager in the sense of resource-bounded Fenner category and Lutz category.*

Proof. This follows from Theorem 3.16. □

4. Genericity versus polynomial-time unsafe approximations.

DEFINITION 4.1 (see Duris and Rolim [6] and Yesha [19]). *A* polynomial-time unsafe approximation *of a set A is a set $B \in \mathbf{P}$. The set $A\Delta B$ is called the* error *set of the approximation. Let f be an unbounded function on the natural numbers. A set A is* Δ-levelable with density f *if, for any set $B \in \mathbf{P}$, there is another set $B' \in \mathbf{P}$ such that*

$$\|(A\Delta B)\restriction z_n\| - \|(A\Delta B')\restriction z_n\| \geq f(n)$$

for almost all $n \in N$. A set A is Δ-levelable *if A is Δ-levelable with density f such that* $\lim_{n\to\infty} f(n) = \infty$.

Note that, in Definition 4.1, the density function f is independent of the choice of $B \in \mathbf{P}$.

DEFINITION 4.2 (see Ambos-Spies [1]). *A polynomial-time unsafe approximation B of a set A is* optimal *if, for any approximation $B' \in \mathbf{P}$ of A,*

$$\exists k \in N\ \forall n \in N\ (\|(A\Delta B)\restriction z_n\| < \|(A\Delta B')\restriction z_n\| + k).$$

A set A is weakly Δ-levelable *if, for any polynomial-time unsafe approximation B of A, there is another polynomial-time unsafe approximation B' of A such that*

$$\forall k \in N\ \exists n \in N\ (\|(A\Delta B)\restriction z_n\| > \|(A\Delta B')\restriction z_n\| + k).$$

It should be noted that our above definitions are a little different from the original definitions of Ambos-Spies [1], Duris and Rolim [6], and Yesha [19]. In the original definitions, they considered the errors on strings up to certain length (i.e., $\|(A\Delta B)^{\leq n}\|$) instead of errors on strings up to z_n (i.e., $\|(A\Delta B)\restriction z_n\|$). But it is easy to check that all our results except Theorem 5.14 in this paper hold for the original definitions also.

LEMMA 4.3 (see Ambos-Spies [1]).

1. *A set A is weakly Δ-levelable if and only if A does not have an optimal polynomial time unsafe approximation.*
2. *If a set A is Δ-levelable then it is weakly Δ-levelable.*

LEMMA 4.4. *Let A, B be two sets such that A is Δ-levelable with linear density and $A\Delta B$ is sparse. Then B is Δ-levelable with linear density.*

Proof. Let p be the polynomial such that, for all n, $\|(A\Delta B)^{\leq n}\| \leq p(n)$, and assume that A is Δ-levelable with density αn ($\alpha > 0$). Then there is a real number $\beta > 0$ such that, for large enough n, $\alpha n - 2p(1 + [\log n]) > \beta n$. We will show that B is Δ-levelable with density βn.

Now, given any set $C \in \mathbf{P}$, by Δ-levelability of A, choose $D \in \mathbf{P}$ such that

$$\|(A\Delta C)\restriction z_n\| > \|(A\Delta D)\restriction z_n\| + \alpha n$$

for almost all n. Then

$$\begin{aligned}
\|(B\Delta C)\restriction z_n\| &\geq \|(A\Delta C)\restriction z_n\| - p(1 + [\log n]) \\
&> \|(A\Delta D)\restriction z_n\| + \alpha n - p(1 + [\log n]) \\
&\geq \|(B\Delta D)\restriction z_n\| + \alpha n - 2p(1 + [\log n]) \\
&> \|(B\Delta D)\restriction z_n\| + \beta n
\end{aligned}$$

for almost all n. Hence, B is Δ-levelable with density βn. □

THEOREM 4.5.

1. *There exists a set G in $\mathbf{E}_2$ which is both A-generic and Δ-levelable.*
2. *There exists a set G in $\mathbf{E}_2$ which is A-generic but not weakly Δ-levelable.*

Proof. 1. Duris and Rolim [6] constructed a set A in $\mathbf{E}$ which is Δ-levelable with linear density and, in [4], Ambos-Spies, Neis, and Terwijn showed that, for any set $B \in \mathbf{E}$, there is an A-generic set B' in $\mathbf{E}_2$ such that $B\Delta B'$ is sparse. So, for any set A which is Δ-levelable with linear density, there is an A-generic set G in $\mathbf{E}_2$ such that $A\Delta G$ is sparse. It follows from Lemma 4.4 that G is Δ-levelable with linear density.

2. Ambos-Spies [1, Theorem 3.3] constructed a **P**-bi-immune set in **E** which is not weakly Δ-levelable. In his proof, he used the requirements

$BI_{2e} : P_e \subseteq G \Rightarrow P_e$ is finite,

$BI_{2e+1} : P_e \subseteq \bar{G} \Rightarrow P_e$ is finite,

to ensure that the constructed set G is **P**-bi-immune. In order to guarantee that G is not weakly Δ-levelable, he used the requirements

$$R : \forall e \in N\ \forall n \in N\ (\|(G\Delta B)\restriction z_n\| \leq \|(G\Delta P_e)\restriction z_n\| + e + 1)$$

to ensure that $B = \cup_{i\geq 0}\Sigma^{[2i]}$ will be an optimal unsafe approximation of G. If we change the requirements BI_{2e} and BI_{2e+1} to the requirements

R_e : if $f_e \in \mathbf{F}_1$ is dense along G, then G meets f_e,

then a routine modification of the finite injury argument in the proof of Ambos-Spies [1, Theorem 3.3] can be used to construct an A-generic set G in $\mathbf{E}_2$ which is not weakly Δ-levelable. The details are omitted here. □

COROLLARY 4.6. *The class of (weakly) Δ-levelable sets is neither meager nor comeager in the sense of resource-bounded Ambos-Spies category.*

Corollary 4.6 shows that the class of weakly Δ-levelable sets is neither large nor small in the sense of resource-bounded Ambos-Spies category. However, as we will show next, it is large in the sense of resource-bounded general Ambos-Spies category, resource-bounded Fenner category, and resource-bounded Lutz category.

THEOREM 4.7. *Let G be a Lutz n^3-generic set. Then G is weakly Δ-levelable.*

Proof. Let $B \in \mathbf{P}$. We show that $\bar{B}$ witnesses that the unsafe approximation B of G is not optimal. For any string x, define $f(x) = y$, where $|y| = |x|^2$ and $y[j] = 0$ if and only if $z_{|x|+j} \in B$. Obviously, f is computable in time n^3. Since G is Lutz n^3-generic, G meets f infinitely often. Hence, for any k and n_0, there exists $n > n_0$ such that $n^2 - 2n > k$ and, for all strings x with $z_n \leq x < z_{n^2}$, $x \in G$ if and only if $x \in \bar{B}$. Hence

$$\begin{aligned} \|(G\Delta B)\restriction z_{n^2}\| &\geq n^2 - n \\ &> n + k \\ &\geq \|(G\Delta\bar{B})\restriction z_{n^2}\| + k, \end{aligned}$$

which implies that G is weakly Δ-levelable. □

COROLLARY 4.8. *The class of weakly Δ-levelable sets is comeager in the sense of resource-bounded Lutz, Fenner, and general Ambos-Spies categories.*

Proof. This follows from Theorems 3.7, 3.8, and 4.7. □

Now we show that the class of Δ-levelable sets is neither meager nor comeager in the sense of all these resource-bounded categories we have discussed above.

THEOREM 4.9. *There exists a set G in $\mathbf{E}_2$ which is both general A-generic and Δ-levelable.*

Proof. Let $\delta(0) = 0, \delta(n+1) = 2^{2^{\delta(n)}}$. For each set $P_e \in \mathbf{P}$, let $P_{g(e)}$ be defined in such a way that

$$P_{g(e)}(x) = \begin{cases} 1 - P_e(x) & \text{if } x = 0^{\delta(<e,n>)} \text{ for some } n \in N, \\ P_e(x) & \text{otherwise.} \end{cases}$$

In the following we construct a general A-generic set G which is Δ-levelable by keeping $P_{g(e)}$ to witness that the unsafe approximation P_e of G is not optimal. Let

$\{f_i : i \in N\}$ be an enumeration of all functions in $\mathbf{F}_2$ such that $f_i(x)$ can be computed uniformly in time $2^{\log^k(|x|+i)}$ for some $k \in N$.

The set G is constructed in stages. To ensure that G is general A-generic, it suffices to meet for all $e \in N$ the following requirements:

G_e : if f_e is dense along G, then G meets f_e.

To ensure that G is Δ-levelable, it suffices to meet for all $e, k \in N$ the following requirements, as shown at the end of the proof:

$L_{\langle e,k\rangle} : \exists n_1 \in N\ \forall n > n_1\ (\|(G\Delta P_e)\restriction z_n\| > \|(G\Delta P_{g(e)})\restriction z_n\| + k)$.

The strategy for meeting a requirement G_e is as follows: at stage s, if G_e has not been satisfied yet and $f_e(G\restriction z_s)$ is defined, then let G extend $(G\restriction z_s)f_e(G\restriction z_s)$. But this action may injure the satisfaction of some requirements $L_{\langle i,k\rangle}$ and G_m. The conflict is solved by delaying the action until it will not injure the satisfaction of the requirements $L_{\langle i,k\rangle}$ and G_m which have higher priority than G_e.

The strategy for meeting a requirement $L_{\langle e,k\rangle}$ is as follows: at stage s, if $L_{\langle e,k\rangle}$ has not been satisfied yet and $P_e(z_s) \neq P_{g(e)}(z_s)$, then let $G(z_s) = P_{g(e)}(z_s)$. When a requirement G_e becomes satisfied at some stage, it is satisfied forever, so $L_{\langle e,k\rangle}$ can only be injured finitely often and then it will have a chance to become satisfied forever.

Stage s.

In this stage, we define the value of $G(z_s)$.

A requirement G_n *requires* attention if

1. $n < s$.
2. G_n has not been satisfied yet.
3. There exists $t \leq s$ such that
   A. $f_n(G\restriction z_t)$ is defined.
   B. $G\restriction z_s$ is consistent with $(G\restriction z_t)f_n(G\restriction z_t)$.
   C. For all $e, k \in N$ such that $\langle e,k\rangle < n$, there is at most one $\langle e,m\rangle \in N$ such that $0^{\delta(\langle e,m\rangle)} \in dom((G\restriction z_t)f_n(G\restriction z_t))$.
   D. For all $e, k \in N$ such that $\langle e,k\rangle < n$,

$$\|(G\Delta P_e)\restriction z_s\| - \|(G\Delta P_{g(e)})\restriction z_s\| > k + n. \tag{1}$$

Fix the minimal m such that G_m requires attention, and fix the minimal t in the above item 3 corresponding to the requirement G_m. If there is no such m, then let $G(z_s) = 1 - P_e(z_s)$ if $z_s = 0^{\delta(\langle e,n\rangle)}$ for some $e, n \in N$, and let $G(z_s) = 0$ otherwise. Otherwise we say that G_m *receives* attention. Moreover, let

$$G(z_s) = \begin{cases} ((G\restriction z_t)f_m(G\restriction z_t))(z_s) & \text{if } z_s \in dom((G\restriction z_t)f_m(G\restriction z_t)), \\ 1 - P_e(z_s) & \text{if } z_s \notin dom((G\restriction z_t)f_m(G\restriction z_t)) \ \&\ z_s = 0^{\delta(\langle e,n\rangle)} \\ & \text{for some } e, n, \\ 0 & \text{otherwise.} \end{cases}$$

This completes the construction.

We show that all requirements are met by proving a sequence of claims.

Claim 1. *Every requirement G_n requires attention at most finitely often.*

Proof. The proof is by induction. Fix n and assume that the claim is correct for all numbers less than n. Then there is a stage s_0 such that no requirement G_m with $m < n$ requires attention after stage s_0. So G_n receives attention at any stage $s > s_0$

at which it requires attention. Hence it is immediate from the construction that G_n requires attention at most finitely often. □

CLAIM 2. *Given $n_0 \in N$, if no requirement $G_n (n < n_0)$ requires attention after stage s_0 and G_{n_0} requires attention at stage s_0, then for all $\langle e, k\rangle < n_0$ and $s > s_0$,*

$$\|(G\Delta P_e)\upharpoonright z_s\| - \|(G\Delta P_{g(e)})\upharpoonright z_s\| > k + n_0 - 1.$$

Proof. The proof is straightforward from the construction. □

CLAIM 3. *Every requirement G_n is met.*

Proof. For a contradiction, fix the minimal n such that G_n is not met. Then f_n is dense along G. We have to show that G_n requires attention infinitely often which is contrary to Claim 1. Since $\|P_e\Delta P_{g(e)}\| = \infty$ for all $e \in N$, by the construction and Claim 2, there will be a stage s_0 such that at all stages $s > s_0$, (1) holds for all $e, k \in N$ such that $\langle e, k\rangle < n$. Hence G_n requires attention at each stage $s > s_0$ at which $f_n(G\upharpoonright z_s)$ is defined. □

CLAIM 4. *Every requirement $L_{\langle e,k\rangle}$ is met.*

Proof. This follows from Claims 2 and 3. □

Now we show that G is both A-generic and Δ-levelable. G is A-generic since all requirements G_n are met. For $\langle e, k\rangle \in N$, let $n_{\langle e,k\rangle}$ be the least number s_0 such that for all $s > s_0$,

$$\|(G\Delta P_e)\upharpoonright z_s\| > \|(G\Delta P_{g(e)})\upharpoonright z_s\| + k$$

and let $f(n)$ be the biggest k such that

$$\forall e \leq k\ (n \geq n_{\langle e,k\rangle}).$$

Then $\lim_{n\to\infty} f(n) = \infty$ and, for all $e \in N$,

$$\|(G\Delta P_e)\upharpoonright z_n\| \geq \|(G\Delta P_{g(e)})\upharpoonright z_n\| + f(n) \text{ a.e.}$$

That is to say, G is Δ-levelable with density f. □

THEOREM 4.10. *There exists a set G in $\mathbf{E}_2$ which is general A-generic but not Δ-levelable.*

Proof. As in the previous proof, a set G is constructed in stages. To ensure that G is general A-generic, it suffices to meet for all $e \in N$ the following requirements:

G_e : if f_e is dense along G, then G meets f_e.

Fix a set $B \in \mathbf{P}$. Then the requirements

$$NL_{\langle e,k\rangle} : P_e\Delta B \text{ infinite } \Rightarrow \exists n\ (\|(G\Delta P_e)\upharpoonright z_n\| - \|(G\Delta B)\upharpoonright z_n\| \geq k)$$

will ensure that B witnesses the failure of Δ-levelability of G.

To meet the requirements G_e, we use the strategy in Theorem 4.9. The strategy for meeting a requirement $NL_{\langle e,k\rangle}$ is as follows: at stage s such that $P_e(z_s) \neq B(z_s)$ and $\|(G\Delta P_e)\upharpoonright z_n\| - \|(G\Delta B)\upharpoonright z_n\| < k$ for all $n < s$, let $G(z_s) = B(z_s)$. If $P_e \neq^* B$, this action can be repeated over and over again. Hence $\|G\Delta P_e\|$ is growing more quickly than $\|G\Delta B\|$, and eventually the requirement $NL_{\langle e,k\rangle}$ is met at some sufficiently large stage.

Define a priority ordering of the requirements by letting $R_{2n} = G_n$ and $R_{2\langle e,k\rangle+1} = NL_{\langle e,k\rangle}$. We now describe the construction of G formally.

Stage s.

In this stage, we define the value of $G(z_s)$.

A requirement $NL_{\langle e,k\rangle}$ *requires* attention if $\langle e, k\rangle < s$ and

1. $P_e(z_s) \neq B(z_s)$.
2. $\|(G\Delta P_e)\restriction z_n\| - \|(G\Delta B)\restriction z_n\| < k$ for all $n < s$.

A requirement G_n *requires* attention if

1. $n < s$.
2. G_n has not been satisfied yet.
3. There exists $t \leq s$ such that
   - **A.** $f_n(G\restriction z_t)$ is defined.
   - **B.** $G\restriction z_s$ is consistent with $(G\restriction z_t)f_n(G\restriction z_t)$.
   - **C.** There is no $e, k \in N$ such that
     - **(1).** $\langle e, k\rangle < n$.
     - **(2).** $\forall u < s$ $(\|(G\Delta P_e)\restriction z_u\| - \|(G\Delta B)\restriction z_u\| < k)$.
     - **(3).** There exists $y \in dom((G\restriction z_t)f_n(G\restriction z_t)) - dom(G\restriction z_s)$ such that $P_e(y) \neq B(y)$.

Fix the minimal m such that R_m requires attention. If there is no such m, let $G(z_s) = B(z_s)$. Otherwise we say that R_m *receives* attention. Moreover, if $R_m = NL_{\langle e,k\rangle}$ then let $G(z_s) = B(z_s)$. If $R_m = G_n$ then fix the least t in the above item 3 corresponding to the requirement G_m. Let $G(z_s) = ((G\restriction z_t)f_m(G\restriction z_t))(z_s)$ if $z_s \in dom((G\restriction z_t)f_m(G\restriction z_t))$ and let $G(z_s) = B(z_s)$ otherwise.

This completes the construction of G.

It suffices to show that all requirements are met. Note that, by definition of requiring attention, R_m is met if and only if R_m requires attention at most finitely often. So, for a contradiction, fix the minimal m such that R_m requires attention infinitely often. By minimality of m, fix a stage s_0 such that no requirement $R_{m'}$ with $m' < m$ requires attention after stage s_0. Then R_m receives attention at any stage $s > s_0$ at which R_m requires attention. Now, we first assume that $R_m = G_n$. Then at some stage $s > s_0$, G_n receives attention and becomes satisfied forever. Finally assume that $R_m = NL_{\langle e,k\rangle}$. Then $B\Delta P_e$ is infinite and, at all stages $s > s_0$ such that $B(z_s) \neq P_e(z_s)$, the requirement $NL_{\langle e,k\rangle}$ receives attention; hence $G(z_s) = B(z_s)$. Since, for all other stages s with $s > s_0$, $B(z_s) = P_e(z_s)$, $G\Delta P_e$ grows more rapidly than $G\Delta B$; hence

$$\lim_n(\|(G\Delta P_e)\restriction z_n\| - \|(G\Delta B)\restriction z_n\|) = \infty$$

and $NL_{\langle e,k\rangle}$ is met contrary to assumption. □

COROLLARY 4.11. *The class of Δ-levelable sets is neither meager nor comeager in the sense of resource-bounded (general) Ambos-Spies, Lutz, and Fenner categories.*

Proof. The proof follows from Theorems 3.7, 4.9, and 4.10. □

5. Resource-bounded randomness versus polynomial-time approximations. We first introduce a fragment of Lutz's effective measure theory which will be sufficient for our investigation.

DEFINITION 5.1. *A* martingale *is a function $F : \Sigma^* \to R^+$ such that, for all $x \in \Sigma^*$,*

$$F(x) = \frac{F(x1) + F(x0)}{2}.$$

A martingale F succeeds *on a sequence $\xi \in \Sigma^\infty$ if $\limsup_n F(\xi[0..n-1]) = \infty$. $S^\infty[F]$ denotes the set of sequences on which the martingale F succeeds.*

DEFINITION 5.2 (see Lutz [10]). *A set $\mathbf{C}$ of infinite sequences has* p-measure 0 *($\mu_p(\mathbf{C}) = 0$) if there is a polynomial-time computable martingale $F : \Sigma^* \to Q^+$ which*

succeeds on every sequence in $\mathbf{C}$. *The set* $\mathbf{C}$ *has* p-measure 1 ($\mu_p(\mathbf{C}) = 1$) *if* $\mu_p(\bar{\mathbf{C}}) = 0$ *for the complement* $\bar{\mathbf{C}} = \{\xi \in \Sigma^\infty : \xi \notin \mathbf{C}\}$ *of* $\mathbf{C}$.

DEFINITION 5.3 (see Lutz [10]). *A sequence* ξ *is* n^k-random *if, for every* n^k*-time computable martingale* F, $\limsup_n F(\xi[0..n-1]) < \infty$; *that is to say,* F *does not succeed on* ξ. *A sequence* ξ *is* p-random *if* ξ *is* n^k*-random for all* $k \in N$.

The following theorem is straightforward from the definition.

THEOREM 5.4. *A set* $\mathbf{C}$ *of infinite sequences has* p-measure 0 *if and only if there exists a number* $k \in N$ *such that there is no* n^k*-random sequences in* $\mathbf{C}$.

Proof. See, e.g., [16]. □

The relation between p-measure and the class of $\mathbf{P}$-levelable sets is characterized by the following theorem.

THEOREM 5.5 (see Mayordomo [11]). *The class of* $\mathbf{P}$*-bi-immune sets has* p*-measure* 1.

COROLLARY 5.6. *The class of* $\mathbf{P}$*-levelable sets has* p*-measure* 0.

COROLLARY 5.7. *The class of sets which possesses optimal polynomial-time safe approximations has* p*-measure* 1.

COROLLARY 5.8. *For each* p*-random set* A, A *has an optimal polynomial-time safe approximation.*

Now we turn our attention to the relations between the p-randomness concept and the concept of polynomial-time unsafe approximations. In our following proof, we will use the law of the iterated logarithm for p-random sequences.

DEFINITION 5.9. *A sequence* $\xi \in \Sigma^\infty$ *satisfies the law of the iterated logarithm if*

$$\limsup_{n\to\infty} \frac{2\sum_{i=0}^{n-1} \xi[i] - n}{\sqrt{2n \ln\ln n}} = 1$$

and

$$\liminf_{n\to\infty} \frac{2\sum_{i=0}^{n-1} \xi[i] - n}{\sqrt{2n \ln\ln n}} = -1.$$

THEOREM 5.10 (see Wang [17]). *There exists a number* $k \in N$ *such that every* n^k*-random sequence satisfies the law of the iterated logarithm.*

For the sake of convenience, we will identify a set with its characteristic sequence. The symmetric difference of two sets can be characterized by the parity function on sequences.

DEFINITION 5.11.

1. *The parity function* $\oplus : \Sigma \times \Sigma \to \Sigma$ *on bits is defined by*

$$b_1 \oplus b_2 = \begin{cases} 0 & \text{if } b_1 = b_2, \\ 1 & \text{otherwise,} \end{cases}$$

   where $b_1, b_2 \in \Sigma$.
2. *The parity function* $\oplus : \Sigma^\infty \times \Sigma^\infty \to \Sigma^\infty$ *on sequences is defined by* $(\xi \oplus \eta)[n] = \xi[n] \oplus \eta[n]$.
3. *The parity function* $\oplus : \Sigma^* \times \{f : f \text{ is a partial function from } \Sigma^* \text{ to } \Sigma\} \to \Sigma^*$ *on strings and functions is defined by* $x \oplus f = b_0 \cdots b_{|x|-1}$, *where* $b_i = x[i] \oplus f(x[0..i-1])$ *if* $f(x[0..i-1])$ *is defined and* $b_i = \lambda$ *otherwise.*
4. *The parity function* $\oplus : \Sigma^\infty \times \{f : f \text{ is a partial function from } \Sigma^* \text{ to } \Sigma\} \to \Sigma^* \cup \Sigma^\infty$ *on sequences and functions is defined by* $\xi \oplus f = b_0 b_1 \cdots$ *where* $b_i = \xi[i] \oplus f(\xi[0..i-1])$ *if* $f(\xi[0..i-1])$ *is defined and* $b_i = \lambda$ *otherwise.*

The intuitive meaning of $\xi \oplus f$ is as follows: Given a sequence ξ and a number $n \in N$ such that $f(\xi[0..n-1])$ is defined, we use f to predict the value of $\xi[n]$ from the first n bits $\xi[0..n-1]$. If the prediction is successful, then output 0, else output 1. And $\xi \oplus f$ is the output sequence.

We first explain a useful technique which is similar to the invariance property of p-random sequences.

LEMMA 5.12. *Let $\xi \in \Sigma^\infty$ be n^k-random and $f : \Sigma^* \to \Sigma$ be a partial function computable in time n^k such that $\xi \oplus f$ is an infinite sequence. Then $\xi \oplus f$ is n^{k-1}-random.*

Proof. For a contradiction assume that $\xi \oplus f$ is not n^{k-1}-random and let $F : \Sigma^* \to Q^+$ be an n^{k-1}-martingale that succeeds on $\xi \oplus f$. Define $F' : \Sigma^* \to Q^+$ by letting $F'(x) = F(x \oplus f)$ for all $x \in \Sigma^*$. It is a routine to check that F' is an n^k-martingale. Moreover, since F succeeds on $\xi \oplus f$, F' succeeds on ξ, which is a contradiction with the hypothesis that ξ is n^k-random. □

LEMMA 5.13. *Let k be the number in Theorem 5.10, and let $A, B, C \subseteq \Sigma^*$ be three sets such that the following conditions hold.*

1. $B, C \in \mathbf{P}$.
2. $\|B \Delta C\| = \infty$.
3. *There exists $c \in N$ such that, for almost all n,*

$$\|(A \Delta C)\restriction z_n\| - \|(A \Delta B)\restriction z_n\| \geq -c. \tag{2}$$

Then A is not n^{k+1}-random.

Proof. Let α, β, and γ be the characteristic sequences of A, B, and C, respectively.

By Lemma 5.12, it suffices to define an n^2-time computable partial function $f : \Sigma^* \to \Sigma$ such that $\alpha \oplus f$ is an infinite sequence which is not n^k-random. Define the function f by

$$f(x) = \begin{cases} \beta[|x|] & \text{if } \beta[|x|] \neq \gamma[|x|], \\ \text{undefined} & \text{if } \beta[|x|] = \gamma[|x|]. \end{cases}$$

Then f is n^2-time computable and, since $\|B \Delta C\| = \infty$, $\alpha \oplus f$ is an infinite sequence. In order to show that $\alpha \oplus f$ is not n^k-random, we show that $\alpha \oplus f$ does not satisfy the law of the iterated logarithm.

We first show that, for all $n \in N^+$, the following equation holds:

$$\sum_{i=0}^{n-1} (\alpha \oplus \gamma)[i] - \sum_{i=0}^{n-1} (\alpha \oplus \beta)[i] = l_n - 2 \sum_{i=0}^{l_n - 1} (\alpha \oplus f)[i], \tag{3}$$

where $l_n = |\alpha[0..n-1] \oplus f|$.

Let

$$a(n) = \|\{i < n : \alpha[i] \neq \gamma[i] = \beta[i]\}\|,$$

$$b(n) = \|\{i < n : \alpha[i] \neq \gamma[i] \neq \beta[i]\}\|,$$

$$c(n) = \|\{i < n : \alpha[i] = \gamma[i] \neq \beta[i]\}\|,$$

$$d(n) = \|\{i < n : \alpha[i] = \gamma[i] = \beta[i]\}\|.$$

Then

$$\begin{aligned}\textstyle\sum_{i=0}^{n-1}(\alpha\oplus\gamma)[i] &= a(n)+b(n),\\ \textstyle\sum_{i=0}^{n-1}(\alpha\oplus\beta)[i] &= a(n)+c(n),\\ l_n &= b(n)+c(n),\\ \textstyle\sum_{i=0}^{l_n-1}(\alpha\oplus f)[i] &= c(n).\end{aligned}$$

Obviously, this implies (3).

The condition (2) is equivalent to

$$\sum_{i=0}^{n-1}(\alpha\oplus\gamma)[i]-\sum_{i=0}^{n-1}(\alpha\oplus\beta)[i]\geq -c.$$

So, by (3),

$$l_n-2\sum_{i=0}^{l_n-1}(\alpha\oplus f)[i]\geq -c \tag{4}$$

for almost all n, where $l_n=|\alpha[0..n-1]\oplus f|$. By (4),

$$\liminf_{n\to\infty}\frac{n-2\sum_{i=0}^{n-1}(\alpha\oplus f)[i]}{\sqrt{2n\ln\ln n}}\geq 0.$$

Hence, by Theorem 5.10, $\alpha\oplus f$ is not n^k-random. This completes the proof. □

Now we are ready to prove our main theorems of this section.

THEOREM 5.14. *The class of Δ-levelable sets has p-measure 0.*

Proof. Let A be a Δ-levelable set. Then there is a function $f(n)\geq 0$ satisfying $\lim_{n\to\infty}f(n)=\infty$ and polynomial-time computable sets B,C such that for all n,

$$\|(A\Delta C)\upharpoonright z_n\|-\|(A\Delta B)\upharpoonright z_n\|\geq f(n).$$

By Lemma 5.13, A is not n^{k+1}-random, where k is the number in Theorem 5.10. So the theorem follows from Theorem 5.4. □

THEOREM 5.15. *The class of sets which have optimal polynomial-time unsafe approximations has p-measure 0.*

Proof. If A has an optimal polynomial-time unsafe approximation, then there is a polynomial-time computable set B and a number $c\in N$ such that, for all n,

$$\|(A\Delta B)\upharpoonright z_n\|-\|(A\Delta\bar{B})\upharpoonright z_n\|<c;$$

i.e.,

$$\|(A\Delta\bar{B})\upharpoonright z_n\|-\|(A\Delta B)\upharpoonright z_n\|>-c.$$

By Lemma 5.13, A is not n^{k+1}-random, where k is the number in Theorem 5.10. So the theorem follows from Theorem 5.4. □

COROLLARY 5.16. *The class of sets which are weakly Δ-levelable but not Δ-levelable has p-measure 1.*

COROLLARY 5.17. *Every p-random set is weakly Δ-levelable but not Δ-levelable.*

Acknowledgments. I would like to thank Professor Ambos-Spies for many comments on an early version of this paper, and I would like to thank two anonymous referees for their valuable comments on this paper.

REFERENCES

[1] K. Ambos-Spies, *On optimal polynomial time approximations:* **P***-levelability vs.* Δ*-levelability*, in Proc. 22nd ICALP, Springer-Verlag, Berlin, New York, 1995, pp. 384–392.

[2] K. Ambos-Spies, *Resource-bounded genericity*, in Proc. 10th Conf. on Structure in Complexity Theory, IEEE Computer Society Press, Piscataway, NJ, 1995, pp. 162–181.

[3] K. Ambos-Spies, H. Fleischhack, and H. Huwig, *Diagonalizations over polynomial time computable sets*, Theoret. Comput. Sci., 51 (1987), pp. 177–204.

[4] K. Ambos-Spies, H.-C. Neis, and S. A. Terwijn, *Genericity and measure for exponential time*, Theoret. Comput. Sci., 168 (1996), pp. 3–19.

[5] L. Berman, *On the structure of complete sets*, in Proc. 17th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Piscataway, NJ, 1976, pp. 76–80.

[6] P. Duris and J. D. P. Rolim, **E***-complete sets do not have optimal polynomial time approximations*, Lecture Notes in Comput. Sci. 841, Springer-Verlag, New York, 1994, pp. 38–51.

[7] S. Fenner, *Notions of resource-bounded category and genericity*, in Proc. 6th Conf. on Structure in Complexity Theory, IEEE Computer Society Press, Piscataway, NJ, 1991, pp. 196–212.

[8] K. Ko and D. Moore, *Completeness, approximation and density*, SIAM J. Comput., 10 (1981), pp. 787–796.

[9] J. H. Lutz, *Category and measure in complexity classes*, SIAM J. Comput., 19 (1990), pp. 1100–1131.

[10] J. H. Lutz, *Almost everywhere high nonuniform complexity*, J. Comput. System Sci., 44 (1992), pp. 220–258.

[11] E. Mayordomo, *Almost every set in exponential time is* **P***-bi-immune*, Theoret. Comput. Sci., 136 (1994), pp. 487–506.

[12] E. Mayordomo, *Contributions to the Study of Resource-Bounded Measure*, Ph.D. thesis, Universidad Polytecnica de Catalunya, Barcelona, 1994.

[13] A. R. Meyer and M. S. Paterson, *With what frequency are apparently intractable problems difficult?* Technical Report TM-126, Laboratory for Computer Science, MIT, Cambridge, MA, 1979.

[14] P. Orponen, A. Russo, and U. Schöning, *Optimal approximations and polynomially levelable sets*, SIAM J. Comput., 15 (1986), pp. 399–408.

[15] D. A. Russo, *Optimal approximation of complete sets*, Lecture Notes in Comput. Sci. 223, Springer-Verlag, New York, 1986, pp. 311–324.

[16] Y. Wang, *Randomness and Complexity*. Ph.D. thesis, Heidelberg, 1996.

[17] Y. Wang, *The law of the iterated logarithm for p-random sequences*, in Proc. 11th Conf. Computational Complexity (formerly Conf. on Structure in Complexity Theory), IEEE Computer Society Press, Piscataway, NJ, 1996, pp. 180–189.

[18] Y. Wang, *Randomness, stochasticity, and approximations*, Lecture Notes in Comput. Sci. 1269, Springer-Verlag, New York, 1997, pp. 213–225.

[19] Y. Yesha, *On certain polynomial-time truth-table reducibilities of complete sets to sparse sets*, SIAM J. Comput., 12 (1983), pp. 411–425.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 409–432

UNIVERSAL LIMIT LAWS FOR DEPTHS IN RANDOM TREES*

LUC DEVROYE†

Abstract. Random binary search trees, b-ary search trees, median-of-$(2k+1)$ trees, quadtrees, simplex trees, tries, and digital search trees are special cases of random split trees. For these trees, we offer a universal law of large numbers and a limit law for the depth of the last inserted point, as well as a law of large numbers for the height.

Key words. binary search tree, data structures, expected time analysis, depth of a node, random tree, law of large numbers

AMS subject classifications. 68Q25, 68P05, 60F05, 60C05

PII. S0097539795283954

Random split trees. We introduce a model for a random tree that is sufficiently general that it encompasses many important families of random trees, such as random binary search trees, random m-ary search trees, random fringe-balanced trees, random median-of-$(2k+1)$ trees, random quadtrees, and random simplex trees. A *skeleton tree* T_b of branch factor b is an infinite rooted position tree, in which each node has b children, numbered 1 through b. A *split tree* (with branch factor $b > 0$, vertex capacity $s > 0$, and of cardinality $n \geq 0$) is a skeleton tree of branch factor b in which n balls are assigned to a collection of vertices, where each vertex may hold up to s balls. Nodes or vertices are denoted by u. $N(u)$ denotes the number of balls in the subtree rooted at u. $C(u)$ denotes the number of balls associated with vertex u. A vertex u is a *leaf* if $C(u) = N(u) > 0$, or equivalently, if $C(u) > 0$ and $N(v) = 0$ for all b children v of u. A node u is *useless* if $N(u) = 0$. Two split trees with the same parameters b, s, n are equivalent if for all their vertices, the $N(u)$'s are identical. The *trimmed split tree* is the (finite) split tree from which all useless nodes are deleted.

We now introduce a *random split tree* with parameters b, s, s_0, s_1, $\mathcal{V}$, and n. The branch factor b, vertex capacity s, and number of balls n are as for split trees. The additional integers s_0 and s_1 are needed to describe the ball distribution process and satisfy the inequalities

$$0 < s,\ 0 \leq s_0 \leq s,\ 0 \leq bs_1 \leq s + 1 - s_0.$$

Finally, $\mathcal{V}$ is a prototype random vector $(V_1, \ldots, V_b)$ of probabilities: $\sum_i V_i = 1; V_i \geq 0$. A random split tree is a skeleton tree T_b in which each vertex u is given an independent copy of $\mathcal{V}$, and in which n balls are distributed in the manner described below over the vertices, where each vertex may hold up to s balls. The distribution is done in an incremental fashion, as described below. Initially, there are no balls, so every node has a ball count $C(u) = 0$. Adding a ball to a tree rooted at u proceeds as follows. Let $(V_1, \ldots, V_b)$ be the probability vector associated with u.

A. If u is not a leaf (so that $C(u) = s_0$), choose child i with probability V_i, increment $N(u)$ by 1, and recursively add the ball to the subtree rooted at child i.

*Received by the editors March 31, 1995; accepted for publication (in revised form) April 3, 1997; published electronically July 7, 1998. The research of the author was sponsored by NSERC grant A3456 and by FCAR grant 90-ER-0291.

http://www.siam.org/journals/sicomp/28-2/28395.html

†School of Computer Science, McGill University, Montreal, Canada H3A 2K6 (luc@cs.mcgill.ca).

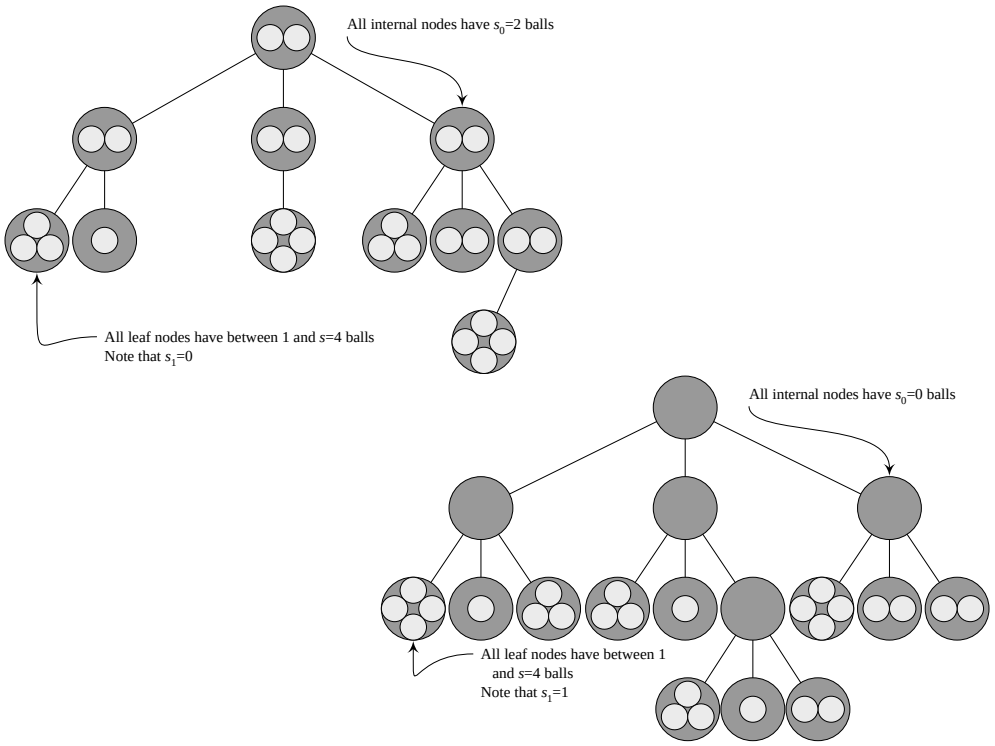


FIG. 1. *A* random trimmed split tree *with parameters b, s, s_0, s_1, $\mathcal{V}$, and n is a random split tree with parameters b, s, s_0, s_1, $\mathcal{V}$, and n from which we eliminate all useless nodes. Note that there is in general no simple relationship between the number of vertices and the number of balls (or points). The ball distribution method described above has several advantages, first and foremost among them the direct relationship with several species of trees that occur as natural data structures. The parameters s, s_0, s_1 add sufficient flexibility. We will see that all trees with fixed finite values of these parameters have the same asymptotic behavior for various shape parameters. When we refer to a random split tree, it is understood that we mean a random trimmed split tree.*

B. If u is a leaf but $C(u) = N(u) < s$ (where s is the capacity of a vertex introduced in the previous paragraph), then add the ball to u and stop. $C(u)$ and $N(u)$ are both incremented by 1.

C. If u is a leaf but $C(u) = N(u) = s$, there is no room for the ball at u. In that case, we set $N(u) = s+1$ and $C(u) = s_0$. We place $s_0 \le s$ randomly selected balls at u, and send $s + 1 - s_0$ balls down to the children of u. This is done as follows. We first give s_1 randomly selected balls to each child and adjust the ball counts for the children. The remaining $s + 1 - s_0 - bs_1$ balls are sent down by choosing a child for each ball independently, according to the probability vector $(V_1, \ldots, V_b)$, and applying the procedure "add a ball" to the tree rooted at the selected child. Note that this may have to be repeated several times if $s_0 = 0$, but only once if $s_0 > 0$ (because no child will reach the capacity s).

Note that every nonleaf node has $C(u) = s_0$ and every leaf has $0 < C(u) \le s$. Two split trees, one with $(s, s_0, s_1) = (4, 2, 0)$ and the other with $(s, s_0, s_1) = (4, 0, 1)$, are shown below.

The depth of a vertex is its distance from the root. The height of a tree is the

maximal depth of any of the leaves. When the $(n+1)$st ball is added to a random split tree of cardinality n (that is, a tree holding n balls), the depth of the vertex reached by the ball is denoted by D_{n+1}. If the leaf is split, the ball ends up, if $s_0 > 0$, at depth D_{n+1} or $D_{n+1}+1$, depending upon which ball we choose to follow in the splitting process. It is therefore convenient to study D_n.

Interestingly, the following property is valid for a tree with n balls rooted at u: if $n \le s$, all balls are in the root node, which is a leaf; if $n > s$, there are s_0 balls in the root node, and the cardinalities $(N_1, \ldots, N_b)$ of the b subtrees of the root are distributed as $(s_1, \ldots, s_1)$ plus a multinomial $(n - s_0 - bs_1, V_1, \ldots, V_b)$ random vector, where $(V_1, \ldots, V_b)$ is associated with u. This property is repeated recursively at every node. Roughly speaking, the subtrees rooted at the children have cardinalities close to $nV_1, \ldots nV_b$.

The behavior of several other parameters is easily deduced from that of D_n. For example, let D'_n be the average depth, i.e., the sum of the depths of the n balls divided by n. The incremental growing process described above shows that, if $s_0 > 0$,

$$\frac{n-an}{n}\mathbf{E}\{D_{an}\} \le \mathbf{E}\{D'_n\} \le \mathbf{E}\{D_n\} + 1$$

for any $a > 0$ such that an is integer. This implies that if $\mathbf{E}\{D_n\} \sim c \log n$ for some constant c, then $\mathbf{E}\{D'_n\} \sim c \log n$. For this reason, we will not investigate D'_n at length.

The purpose of this paper is to point out that within this general setting, the asymptotics—a law of large numbers and a limiting distribution—of D_n are easy to determine. Interestingly, one proof is offered that works for all trees mentioned above.

Throughout this paper, we assume that the components of $(V_1, \ldots, V_b)$ are identically distributed. Note that if they are not, a random permutation $(\sigma_1, \ldots, \sigma_b)$ of $(1, 2, \ldots, b)$ shows that we achieve this goal by taking $(V_{\sigma_1}, \ldots, V_{\sigma_b})$. This random permutation of the children does not affect the depth and height. If V_i has a distribution described by the probability measure μ_i, then the V_{σ_i}'s are identically distributed with common probability measure $(1/b)\sum_j \mu_j$. The latter is called the splitting distribution. A random variable V with the splitting distribution is called a *splitter*. Define $W = V_S$ where, given $(V_1, \ldots, V_b)$, $S = i$ with probability V_i. Observe that $\mathbf{E}V = \mathbf{E}W = 1/b$ in all cases. The law of large numbers and the limit law for D_n depend upon just two parameters,

$$\mu = \mathbf{E}\{\log(1/W)\} = b\mathbf{E}\{V \log(1/V)\}$$

and

$$\sigma^2 = \mathbf{Var}\{\log W\} = b\mathbf{E}\left\{V \log^2 V\right\} - \mu^2.$$

We first state our main results without proof. Then we give a brief discussion of the random trees to which the results apply. The proofs are at the end of the paper.

The main result. For all trees that follow the given model, if H_n denotes the height, that is, the maximal distance between the root and any leaf, we have $H_n = O(\log n)$ in probability. The behavior of H_n is related to that of the moment function

$$m(t) = \mathbf{E}\{V^t\},\ t \ge 0.$$

For later reference, we provide the key properties of this function.

LEMMA 1.

A. *The function m decreases monotonically from $m(0) = 1$ to $\mathbf{P}\{V = 1\}$ as $t \to \infty$.*
B. *m is differentiable for all $t > 0$.*
C. *$\log m$ is convex. In particular, m'/m is increasing on $(0, \infty)$.*
D. *$m(t)^{1/t} \le m(s)^{1/s}$ for $t < s$. Thus, $\log(m(t))/t$ is nondecreasing.*
E. *For $t > 0$, $m'(t) = \mathbf{E}\{V^t \log(V)\}$ and $m''(t) = \mathbf{E}\{V^t \log^2(V)\}$.*
F. *m'/m takes every value between $\mathbf{E}\{\log(V)\}$ (as $t \downarrow 0$) and $\log v_\infty$, where v_∞ is the rightmost point in the support of V.*
G. *The solution of the equation $m'(t)/m(t) = -1/c$ is called $t^* = t^*(c)$. Then t^* is a monotonically increasing function of c, and a solution exists when*

$$-\frac{1}{\mathbf{E}\{\log(V)\}} < c < -\frac{1}{\log v_\infty}.$$

H. *$t^*/c + \log m(t^*)$ decreases in c (or t^*). The value of $t^*/c + \log m(t^*)$ changes from 0 (at $t^* = 0$) to R (possibly $-\infty$) as $t^* \to \infty$, where $R = \lim_{t\to\infty}(\log m(t) - t\, m'(t)/m(t))$.*

THEOREM 1. *Let V be a splitter for a random split tree. Assume that $\mathbf{P}\{V = 1\} = 0$. Then there exists a finite constant c such that*

$$\lim_{n\to\infty} \mathbf{P}\{H_n > c \log n\} = 0.$$

If, additionally, $R < -\log b$, where

$$R = \lim_{t\to\infty} (\log m(t) - t\, m'(t)/m(t)),$$

then the same is true for all $c > \gamma$ and $\gamma \in (0, \infty)$ is a parameter only depending upon b and the distribution of V, and is defined by

$$\gamma = \inf\{c : e^{t^*}(b m(t^*))^c < 1\} = \inf\{c : t^*/c + \log(m(t^*)) < -\log b\},$$

where t^* is the unique solution of $m'(t)/m(t) = -1/c$. (See Lemma 1 below.)

We note that under the conditions of Theorem 1, $H_n/\log n \to \gamma$ in probability. The lower bound that goes with the upper bound of Theorem 1 can be obtained by various methods, and its straightforward proof is not included here (as we focus mainly on depths in this paper). Galton–Watson processes (see Athreya and Ney, 1972) may be used directly (Devroye, 1987). One may also use extrema in branching random walks as exhibited in Devroye (1986b) (see Biggins (1976, 1977), Hammersley (1974), and Kingman (1973) for branching random walks, and see Mahmoud (1992) for further applications). Pittel (1994) points out how one may use the Crump–Mode process in this respect.

THEOREM 2. *Let D_n be the depth of the last node in a random split tree with n nodes and splitter V. If $\mu \neq 0$ and $\mathbf{P}\{V = 1\} = 0$, then*

$$\frac{D_n}{\log n} \to \frac{1}{\mu}$$

and $\mathbf{E}\{D_n\}/\log n$ tends to the same limit. Furthermore, if $\sigma > 0$, then

$$\frac{D_n - (\log n)/\mu}{\sqrt{\sigma^2(\log n)/\mu^3}} \xrightarrow{\mathcal{L}} \mathcal{N}(0, 1),$$

where $\mathcal{N}(0,1)$ *denotes the normal distribution and* $\xrightarrow{\mathcal{L}}$ *denotes convergence in distribution.*

The law of large numbers for D_n does not apply when $\mathbf{E}\{V \log V\} = 0$, i.e., when $\mathbf{P}\{V \in \{0,1\}\} = 1$. This degenerate case is excluded from further consideration. It suffices that V has a density (as in many examples that follow below). For the limit law, we need in addition $\mathbf{Var}\{\log W\} > 0$. This is equivalent to asking that V not be monoatomic. Of all the examples below, only special cases of tries—the symmetric tries and symmetric digital search trees—have a monoatomic splitter V ($V \equiv 1/b$). All other examples satisfy the latter condition.

Some properties of the beta distribution. The beta distribution plays an important role in many important random split trees. We summarize some key properties. Define the beta (a,b) density

$$f(x) = \frac{x^{a-1}(1-x)^{b-1}}{B(a,b)},\ 0 < x < 1,$$

where $a, b > 0$ are parameters and $B(a,b) = \Gamma(a)\Gamma(b)/\Gamma(a+b)$.

LEMMA 2. *If* X *is a beta* (a,b) *random variable, then*

$$\mathbf{E}\{\log(1/X)\} = \psi(a+b) - \psi(a),$$

where $\psi(u) = \Gamma'(u)/\Gamma(u)$ *is the derivative of* $\log\Gamma$ *at* u *(also called the digamma function). Furthermore,*

$$\mathbf{E}\{X \log(1/X)\} = \frac{a}{a+b}(\psi(a+1+b) - \psi(a+1)).$$

Let ψ'*—the trigamma function—be the derivative of* ψ*. Then*

$$\mathbf{E}\{\log^2(X)\} = (\psi(a+b) - \psi(a))^2 + \psi'(a) - \psi'(a+b).$$

Finally,

$$\mathbf{E}\{X \log^2(X)\} = \frac{a}{a+b}\left(\psi(a+1+b) - \psi(a+1)\right)^2 + \frac{a}{a+b}\left(\psi'(a+1) - \psi'(a+1+b)\right).$$

For integrals such as those dealt with in Lemma 2, we refer to Sibuya (1979) or Gradshteyn and Ryzhik (1980, pp. 538, 541). We recall that the digamma function basically behaves like the harmonic numbers (Abramowitz and Stegun, 1970, pp. 258–259): if γ is Euler's constant,

$$\psi(n) = -\gamma + \sum_{k=1}^{n-1} \frac{1}{k} \quad (n \geq 2),\ \psi(1) = -\gamma;$$

$$\psi(z+1) = \psi(z) + \frac{1}{z} = -\gamma + \sum_{n=1}^{\infty} \frac{z}{n(n+z)},\ z > -1.$$

For the trigamma function, we have

$$\psi'(z) = \sum_{n=0}^{\infty} \frac{1}{(z+n)^2}.$$

Also, $\psi'(z) = \psi'(z-1) - 1/(z-1)^2$, and $\psi'(1) = \pi^2/6$.

TABLE 1

Tree	$V \stackrel{\mathcal{L}}{=}$	s	s_0	s_1	b
Random binary search tree	U, uniform $[0,1]$ beta (1,1)	1	1	0	2
Random b-ary search tree	$U_{(b-1)}=\min(U_1,\ldots,U_{b-1})$ $\text{beta}_{(1,b-1)}$	$b-1$	$b-1$	0	b
Random quadtree	$\prod_{i=1}^{d} U_i$	1	1	0	2^d
Random median-of-$(2k+1)$ binary search tree	median $(U_1,\ldots,U_{2k+1})$ beta $(k+1,k+1)$	$2k$	1	k	2
Random simplex tree	$\min_{1\le i\le d} U_i$ $\text{beta}_{(1,d)}$	1	1	0	$d+1$
AB tree	symmetric beta (a,a)	1	0(1)	0	2
Extended AB tree	uniform {beta (a,b), beta (b,a)}	1	0(1)	0	2
Trie	uniform $\{p_1,\ldots,p_b\}$	1	0	0	b
Digital search tree	uniform $\{p_1,\ldots,p_b\}$	1	1	0	b
Random m-grid tree	$\prod_{i=1}^{d} U_i'$ $U_1',\ldots,U_d'$ i.i.d. $\text{beta}_{(1,m)}$	m	m	0	$(m+1)^d$

Examples: An overview. In Table 1, we list a number of important special cases of random split trees. In this table, $U, U_1, U_2, \ldots$ are independently and identically distributed (i.i.d.) uniform $[0,1]$ random variables. Recall that s is the capacity of a node before it is split, s_0 is the number of balls left in a node after a split, s_1 is the minimum number of balls sent to any subtree, and b is the branch factor.

Table 1 shows that a large variety of trees may be dealt with in one sweep. The fixed parameters s, s_0, and s_1 are irrelevant for first term asymptotics and the law of large numbers for depths and heights. Only the distribution of the splitter V matters.

Nonetheless, many trees cannot be molded into our framework, such as all trees whose depth does not grow logarithmically with n. For example, it is well known that the uniform random binary tree has average depth and height of the order of $\sqrt{n}$ (Flajolet and Odlyzko, 1982; Vitter and Flajolet, 1990). Interestingly, Aldous (1993) has introduced a model that includes many (but not all) of the trees in Table 1 and the uniform random trees, as well as a continuum of trees that link them. Our work was inspired for a great deal by Aldous's paper.

The parameters μ and σ^2 are computed for the trees mentioned above. Most of the limit laws and laws of large numbers are known, but the unified approach of this paper explains things in a stronger way. More details are provided in the nine subsequent subsections, in which each tree is briefly discussed separately. The symbols $\mathcal{H}$ and $\mathcal{H}_2$ are properly defined in the section on tries.

Example 1: *The random binary search tree.* In a random binary search tree with n nodes, the following operation is applied independently and recursively: a random node is chosen from the n nodes at hand, and it is made the root. The nodes with a smaller label travel to the left subtree of the root, and the others, to the right subtree. The size of the left subtree is distributed as $\lfloor nU \rfloor$, where U is uniform $[0,1]$. The size of the right subtree has a similar distribution. Equivalently, attach to each of the balls an independent copy of a uniform $[0,1]$ random variable, to get $U_1, U_2, \ldots, U_n$. Put U_1 in the root and partition the others into left and right subsets by comparison with U_1. Repeating the splitting process at each node creates a random split tree. In a third equivalent representation, that of the random split tree, we may associate with

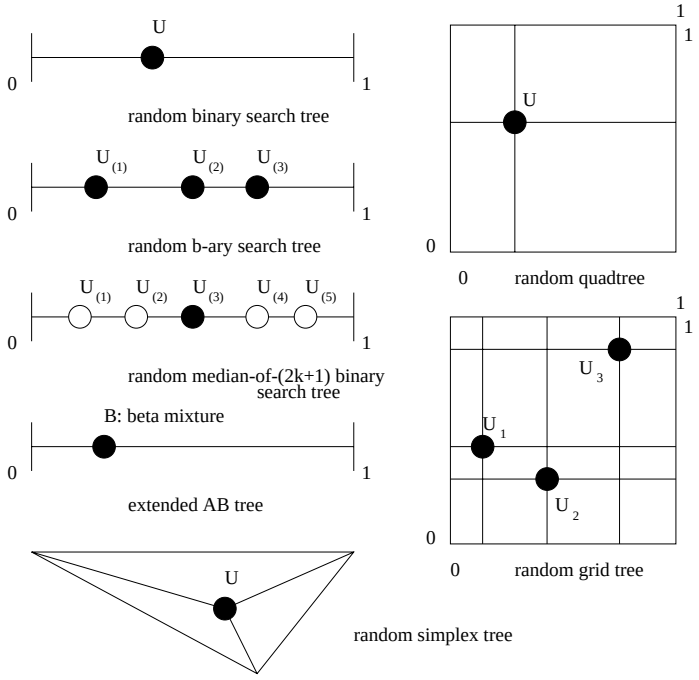


FIG. 2. *Various ways of splitting spaces are shown. All splits are applied recursively. U refers to the uniform distribution, and B, to a mixture of beta distributions.*

TABLE 2

Tree	σ^2	$1/\mu$ (limit of $D_n/\log n$)
Random binary search tree	$1/4$	2
Random b-ary search tree	$\sum_{i=2}^{b} \frac{1}{i^2}$	$\frac{1}{\sum_{i=2}^{b} \frac{1}{i}}$
Random quadtree	$d^2/4$	$2/d$
Random median-of-$(2k+1)$ binary search tree	$\sum_{j=k+2}^{2k+2} \frac{1}{j^2}$	$\frac{1}{\sum_{i=k+2}^{2k+2} \frac{1}{i}}$
Random simplex tree	$\sum_{i=2}^{d+1} \frac{1}{i^2}$	$\frac{1}{\sum_{i=2}^{d+1} \frac{1}{i}}$
AB tree	$\frac{(\psi(2a+1)-\psi(a+1))^2}{+(\psi'(a+1)-\psi'(2a+1))}$	$=\frac{\frac{1}{\psi(2a+1)-\psi(a+1)}}{\sum_{n=1}^{\infty} \frac{a}{(n+a+1)(n+2a+1)}}$
Extended AB tree	$\frac{a}{a+b}(\psi(a+1+b)-\psi(a+1))^2$ $+\frac{a}{a+b}(\psi'(a+1)-\psi'(a+1+b))$ $+\frac{b}{a+b}(\psi(a+1+b)-\psi(b+1))^2$ $+\frac{b}{a+b}(\psi'(b+1)-\psi'(a+1+b))$	$\frac{a+b}{\frac{a(\psi(a+1+b)-\psi(a+1))}{+b(\psi(a+1+b)-\psi(b+1))}}$
b-ary trie	$\mathcal{H}_2-\mathcal{H}^2$	$1/\mathcal{H}$
b-ary digital search tree	$\mathcal{H}_2-\mathcal{H}^2$	$1/\mathcal{H}$
Random m-grid tree	$d\sum_{j=2}^{m+1} \frac{1}{j^2} + d(d-1)\left(\sum_{j=2}^{m+1} \frac{1}{j}\right)^2$	$\frac{1}{d\sum_{i=2}^{m+1} \frac{1}{i}}$

each node an independent random split vector (V_1, V_2) distributed as $(U_1, 1-U_1)$.

It is known that $D_n/\log n \to 2$ in probability (Lynch, 1965; Knuth, 1973; Devroye, 1988). The limit law for D_n was derived by Devroye (1988): $(D_n - 2\log n)/\sqrt{2\log n} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1)$. Robson (1979), Pittel (1984), and Devroye (1986b, 1987) showed that $H_n/\log n \to 4.31107\ldots$ in probability. All of these results are contained in Theorems 1 and 2 as $m(t) = 1/(t+1)$, $\mu = 1/2$, and $\sigma^2 = 1/4$.

An important variant of the binary search tree related to the standard occurs when no balls are stored in internal nodes; so, $s_0 = 0$. This leads to a binary search tree in which all balls are at leaves. This too is a random split tree, and it follows the same limit laws as the ordinary random binary search tree.

Example 2: *The random b-ary search tree.* Let n balls be given and associate with each ball an independent uniform $[0,1]$ random variable. In a random b-ary search tree with n nodes, the following operation is applied independently and recursively: $b-1$ random balls are chosen from the n balls at hand and are associated with the root. The other balls, if there are any, are partitioned into b sets by membership in the intervals induced by the $b-1$ balls. If $(N_1, \ldots, N_b)$ are the number of balls in the intervals (with $\sum_i N_i = n - b + 1$, of course), then this vector is multinomial $(n-b+1, V_1, \ldots, V_b)$, where the V_i's are the lengths of the intervals (or spacings; see Pyke (1965)). The split vector of V_i's is thus distributed as the collection of b spacings induced by $b-1$ i.i.d. uniform $[0,1]$ random variables on $[0,1]$. In particular, $V = V_1$ is distributed as a beta $(1, b-1)$ random variable.

We easily compute $\mu = \sum_{i=2}^{b} 1/i$ and $\sigma^2 = \sum_{i=2}^{b} 1/i^2$. This yields

$$\frac{D_n}{\log n} \to \frac{1}{\sum_{i=2}^{b} \frac{1}{i}} \quad \text{in probability}$$

(a result of Mahmoud and Pittel (1984)) and

$$\frac{D_n - (1/\mu)\log n}{\sqrt{(\sigma^2/\mu^3)\log n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

As an example, if $b = 3$, $\mu = 5/6$, $\sigma^2 = 78/125$, and

$$\frac{D_n - (6/5)\log n}{\sqrt{(78/125)\log n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

We also know that $H_n/\log n \to c$ in probability for a function c of b given in Devroye (1990) and indicated in Theorem 1.

Example 3: *The random quadtree.* The point quadtree in R^d (Finkel and Bentley, 1974; see Samet (1990b) for a survey) generalizes the binary search tree. One ball is put in each node of a tree with branch factor 2^d; each ball has associated with it a d-vector for the point it represents; each subtree of a node corresponds to one of the quadrants formed by considering the ball's d-vector as the new origin. Insertion into point quadtrees is as for binary search trees.

We assume that a random quadtree is constructed on the basis of an i.i.d. sequence drawn from the uniform distribution on $[0,1]^d$. In that case, it is convenient to index the split vector by a bit sequence of length d: $(b_1 \ldots b_d)$. The vector $(V_{00\ldots00}, \ldots, V_{11\ldots11})$ has components that may be written as

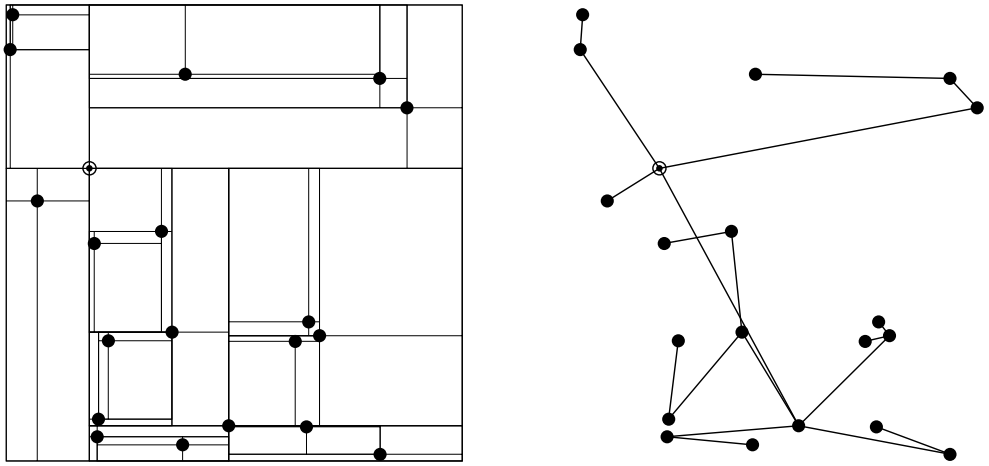


FIG. 3. *At the left, a partition of the plane by a random quadtree is shown. The circled point is the root. It partitions the space into four quadrants, and the splitting rule is recursively applied to each quadrant. At the right, the same points are shown together with the edges in the quadtree.*

$$V_{(b_1 \ldots b_d)} = \prod_{j=1}^{d} U_j^{b_j} (1 - U_j)^{1-b_j},$$

where $(U_1, \ldots, U_d)$ is the point in the node where the split takes place.

The height H_n of a random quadtree is in probability asymptotic to $(c/d) \log n$, where $c = 4.31107\ldots$ is the constant in the height of the random binary search tree (Devroye, 1987). This also follows from Theorem 1 as $m(t) = \mathbf{E}\{V^t\} = 1/(t+1)^d$. Write $V = V_{11\ldots11} = \prod_{j=1}^{d} U_j$. Then it takes just a moment to verify that

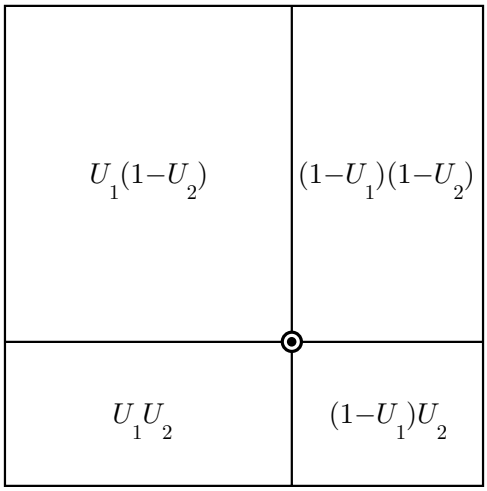


FIG. 4. *The split induced by the root of the quadtree is shown. The random variables U_1, U_2 are i.i.d. uniform $[0,1]$. The areas of the four rectangles are all distributed like products of two independent uniform $[0,1]$ random variables.*

$$\begin{aligned}\mu &= \mathbf{E}\{\log W\} = 2^d \mathbf{E}\{V \log(1/V)\} \\ &= 2^d \sum_{j=1}^{d} \mathbf{E}\left\{\prod_{k=1}^{d} U_k \log(1/U_j)\right\} \\ &= 2^d d\, \mathbf{E}\left\{\prod_{k=2}^{d} U_k\right\} \mathbf{E}\left\{U_1 \log(1/U_1)\right\} \\ &= 2d\, \mathbf{E}\left\{U_1 \log(1/U_1)\right\} \\ &= \frac{d}{2}.\end{aligned}$$

From this, we see that

$$\frac{D_n}{\log n} \to \frac{2}{d} \quad \text{in probability,}$$

a result first noted by Devroye and Laforest (1990). See also Flajolet et al. (1991). The computations of the variance are a bit more tedious. We have

$$\begin{aligned}\sigma^2 + \mu^2 &= 2^d \mathbf{E}\{V \log^2(1/V)\} \\ &= 2^d \mathbf{E}\left\{\prod_{k=1}^{d} U_k \left(\sum_{j=1}^{d} \log(1/U_j)\right)^2\right\} \\ &= 2^d d\, \mathbf{E}\left\{\prod_{k=2}^{d} U_k\right\} \mathbf{E}\left\{U_1 \log^2(1/U_1)\right\} \\ &\quad + 2^d\, d(d-1) \mathbf{E}\left\{\prod_{k=3}^{d} U_k\right\} \mathbf{E}^2\left\{U_1 \log(1/U_1)\right\} \\ &= 2d\, \mathbf{E}\left\{U_1 \log^2(1/U_1)\right\} + 4d(d-1)\mathbf{E}^2\left\{U_1 \log(1/U_1)\right\} \\ &= \frac{d^2}{4} + \frac{d}{4}.\end{aligned}$$

Hence, $\sigma^2 = d/4$. This yields the limit law

$$\frac{D_n - (2/d)\log n}{\sqrt{(2/d^2)\log n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1),$$

valid for any $d \geq 1$. This result was obtained via complex analysis by Flajolet and Lafforgue (1994).

Example 4: *The random median-of-*$(2k+1)$ *binary search tree.* Bell (1965) and Walker and Wood (1976) introduced the following method for constructing a binary search tree. Take $2k+1$ points at random from the set of n points on which a total order is defined, where k is integer. The median of these points serves as the root of a binary tree. The remaining points are thrown back into the collection of points and are sent to the subtrees. Following Poblete and Munro (1985), we may look at this tree by considering internal nodes and external nodes, where internal nodes hold one data point and external nodes are bags of capacity $2k$. Insertion proceeds as usual.

As soon as an external node overflows (i.e., when it would grow to size $2k+1$), its bag is split about the median, leaving two new external nodes (bags) of size k each and an internal node holding the median. After the insertion process is completed, we may wish to expand the bags into balanced trees. Using the branching process method of proof (Devroye, 1986b, 1987, 1990; see also Mahmoud, 1992) the almost sure limit of $H_n/\log n$ for all k may be obtained (Devroye, 1993). For another possible proof method, see Pittel (1992). The depth D_n of the last node when the fringe heuristic is used has been studied by the theory of Markov processes or urn models in a series of papers, notably by Poblete and Munro (1985) and Aldous, Flannery, and Palacios (1988). See also Gonnet and Baeza-Yates (1991, p. 109). Poblete and Munro (1985) showed that

$$\frac{D_n}{\log n} \to \frac{1}{\sum_{i=k+2}^{2k+2} \frac{1}{i}}$$

in probability. It should be clear by now that this tree is a random split tree with $s = 2k$, $s_0 = 1$, $s_1 = k$, $b = 2$ and split vector (V_1, V_2) distributed as $(B, 1-B)$, where B is beta $(k+1, k+1)$. That is, B is distributed as the median of $2k+1$ i.i.d. uniform $[0,1]$ random variables. This representation is obtained by associating with each point in the data an independent uniform $[0,1]$ random variable. Clearly,

$$\mu = \sum_{i=k+2}^{2k+2} \frac{1}{i}.$$

Also, if X is beta (a,a) and a is integer-valued, Lemma 2 and the properties of the digamma and trigamma functions imply

$$\begin{aligned}\mathbf{E}\{X\log^2(X)\} &= \frac{1}{2}\left(\psi(2a+1)-\psi(a+1)\right)^2 + \frac{1}{2}\left(\psi'(a+1)-\psi'(2a+1)\right)\\ &= \frac{1}{2}\left(\left(\sum_{j=a+1}^{2a}\frac{1}{j}\right)^2 + \sum_{j=a+1}^{2a}\frac{1}{j^2}\right).\end{aligned}$$

Thus,

$$\sigma^2 = \sum_{j=k+2}^{2k+2}\frac{1}{j^2}.$$

Therefore, we obtain a limit law for all k. As an example, for $k=1$, we obtain $\mu = 1/3 + 1/4 = 7/12$, $\sigma^2 = 1/9 + 1/16 = 25/144$, and thus

$$\frac{D_n - (12/7)\log n}{\sqrt{(300/343)\log n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

We rediscover results for the number of comparisons C_n for median-of-$(2k+1)$ quicksort. As $\mathbf{E}C_n = \mathbf{E}\{nD_n\}$, where D_n is the depth of the nth point when inserted in a median-of-$(2k+1)$ binary search tree holding $n-1$ points. From the above results,

$$\lim_{n\to\infty}\frac{\mathbf{E}C_n}{n\log n} = \frac{1}{\frac{1}{k+2}+\frac{1}{k+3}+\cdots+\frac{1}{2k+2}}.$$

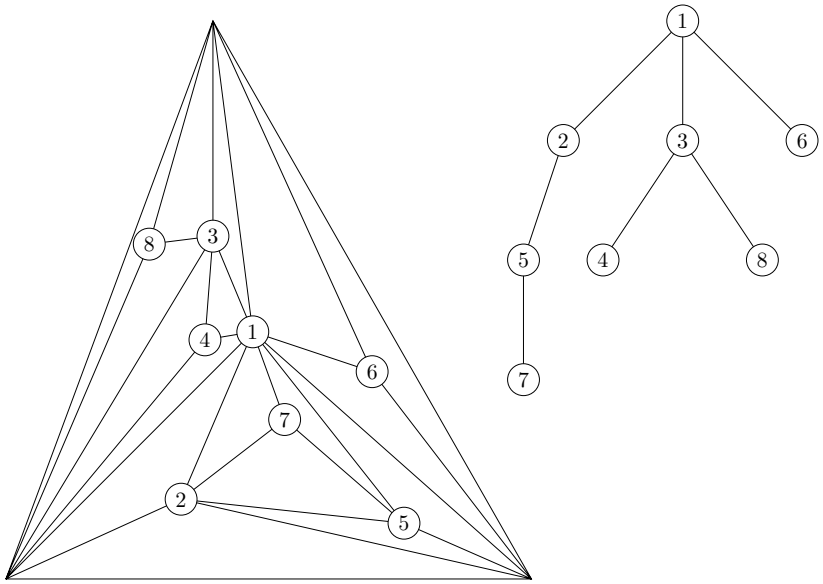


FIG. 5. *A triangle is triangulated by a cloud of random points. The corresponding ternary tree is shown on the right.*

Thus, for median-of-5 quicksort, as $60/47n \log n$, the expected number of comparisons grows. By generating function methodology (Vitter and Flajolet, 1990; Kemp, 1984; Sedgewick, 1983) or via urn models (Aldous, Flannery, and Palacios, 1988), results of this nature are harder to obtain. However, our method does not allow one to compute anything but the main asymptotic term in $\mathbf{E}C_n$.

Example 5: *Random simplex trees.* Triangulating polygons and objects in the plane is an important problem in computational geometry. An $O(n \log n)$ algorithm for triangulating n points was found by Avis and El Gindy (1987). Arkin et al. (1994) obtained a simple fast $O(n \log n)$ expected time algorithm for triangulating any collection of n planar points in general position. We look more specifically at their triangulation and its d-dimensional extension to simplices, and ask what the tree generated by this partitioning looks like if the points are uniformly distributed in the unit simplex. Given are n vectors $X_1, \ldots, X_n$ taking values in a fixed simplex S of $\mathbb{R}^d$. It is assumed that this is an i.i.d. sequence with a uniform distribution on S for the purposes of analysis. X_1 is associated with the root of a $d+1$-ary tree. It splits S into $d+1$ new simplices by connecting X_1 with the $d+1$ vertices of S. Associate with each of these simplices the subset of $X_2, \ldots, X_n$ consisting of those points that fall in the simplex. Each nonempty subset is sent to a child of the root, and the splitting is applied recursively to each child. As every split takes linear time in the number of points processed, it is clear that the expected time is proportional to $n\mathbf{E}D_n$, where D_n is the expected depth of a random node in the tree. The partition consists of $dn+1$ simplices, each associated with an external node of the tree. There are precisely n nodes in the tree and each node contains one point.

If $|S|$ denotes the size of a simplex S, then the following crucial property is valid.

LEMMA 3. *If simplex S is split into $d+1$ simplices $S_1, \ldots, S_{d+1}$ by a point X distributed uniformly in S, then $(|S_1|, \ldots, |S_{d+1}|)$ is jointly distributed as $(|S|V_1, \ldots,$*

$|S|V_{d+1})$, *where* $V_1, \ldots, V_{d+1}$ *are the spacings of* $[0,1]$ *induced by* d *i.i.d. uniform* $[0,1]$ *random variables.*

Proof. It is known that X is distributed as $\sum_{i=1}^{d+1} V_i t_i$, where $t_1, \ldots, t_{d+1}$ are the vertices of S (see Rubinstein (1982), Smith (1984), or Devroye (1986a)). But for a simplex S, we know that

$$|S| = \frac{1}{d!} \det \begin{pmatrix} t_1 & t_2 & t_3 & \cdots & t_{d+1} \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix}.$$

Apply this formula to S_1 by replacing t_1 by X:

$$\begin{aligned} |S_1| &= \frac{1}{d!} \det \begin{pmatrix} X & t_2 & t_3 & \cdots & t_{d+1} \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix} \\ &= \frac{1}{d!} \det \begin{pmatrix} \sum_i V_i t_i & t_2 & t_3 & \cdots & t_{d+1} \\ \sum_i V_i & 1 & 1 & \cdots & 1 \end{pmatrix} \\ &= \frac{V_1}{d!} \det \begin{pmatrix} t_1 & t_2 & t_3 & \cdots & t_{d+1} \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix} \\ &= V_1 |S|. \end{aligned}$$

The statement then follows trivially. □

It is immediate that the random simplex tree is a random split tree with split vector distributed as the spacings defined by d i.i.d. uniform $[0,1]$ random variables on $[0,1]$, $s_0 = 1$, $s = 1$, $s_1 = 0$, and $b = d+1$. Therefore, by Theorem 2, D_n behaves precisely as for the random d+1-ary tree discussed earlier. Thus, $\mu = \sum_{i=2}^{d+1} 1/i$ and $\sigma^2 = \sum_{i=2}^{d+1} 1/i^2$. This yields

$$\frac{D_n}{\log n} \to \frac{1}{\sum_{i=2}^{d+1} \frac{1}{i}} \quad \text{in probability}$$

and

$$\frac{D_n - (1/\mu)\log n}{\sqrt{(\sigma^2/\mu^3)\log n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

As an example, if $d = 2$, then $\mu = 5/6$, $\sigma^2 = 78/125$, and

$$\frac{D_n - (6/5)\log n}{\sqrt{(78/125)\log n}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

For $d = 3$, $\mu = 1/2 + 1/3 + 1/4 = 13/12$. Thus, $D_n/\log n \to 12/13$ in probability. We also know that $H_n/\log n \to c$ in probability for a function c of d that may be computed via the recipe described in Theorem 1.

Example 6: *Extended AB trees and simulation.* When generating random trees that resemble botanical trees, a number of mathematical models have been proposed. We refer to Viennot's survey (1990) or the book by Prusinkiewicz and Lindenmayer (1990) for further references. Stripped from geometrical considerations, most trees are binary. The main parameter one needs to control is the height of the tree as a function of the number of nodes. Alternately, one may wish to control the average distance from a node to the root. For this, it is necessary to have a family of random trees in which these parameters can take any large value. In the context of this paper,

if we had a family of splitting trees—a continuum of trees, really—with parameter α and for which $D_n/\log n \to c(\alpha)$ in probability, we would be saved if the domain of values of $c(\alpha)$ would be $(\log 2, \infty)$ as α varies over a given range.

In 1993, Aldous introduced a family of random split trees with $s = 1$, $s_1 = 0$, and $s_0 = 0$. With this set-up, all balls are put in leaves, and internal nodes have no balls. Aldous splits with the aid of $(V, 1-V)$, where V is beta (a, a) for some parameter $a > 0$. By varying a and even extending it beyond its natural range, Aldous creates a one-parameter family that may be used to model certain splitting processes in biology. He also studies the depths of nodes in these trees and obtains laws of large numbers for their heights. We define an AB tree (for *Aldous beta*) in a similar fashion but take $s_0 = s = 1$ and $s_1 = 0$. This change is only cosmetic, as it will not affect any asymptotic result. For $a = 1$, we obtain the random binary search tree. As $a \downarrow 0$, the tree becomes more elongated, and the amount of stretching may be controlled by a. As $a \to \infty$, every split is nearly 50-50, and the height of the tree is in probability asymptotic to $\log_2 n$. The laws of large numbers for depths and heights are essentially those obtained by Aldous for his model.

We feel that a lot is gained by considering two-parameter families for modeling biological phenomena and simulating botanical trees. This may be achieved by extending the AB trees and taking B as an equal mixture of a beta (a, b) and a beta (b, a) density. The splitter V remains symmetric, but as a and b diverge so that $a/(a+b) \to p \in (0, 1)$, we see that in the limit V is p or $1-p$ with equal probability. This creates trees of height about $\log_{1/\min(p,1-p)} n$. The AB trees are obtained at $b = a$. We call this versatile family of trees extended AB trees. We will report on the drawing of realistic-looking trees via extended AB trees elsewhere.

From Lemma 2, the parameters are easily obtained:

$$\mu = \frac{a(\psi(a+1+b) - \psi(a+1)) + b(\psi(a+1+b) - \psi(b+1))}{a+b},$$

and

$$\sigma^2 = \frac{a}{a+b}\left(\psi(a+1+b) - \psi(a+1)\right)^2 + \frac{a}{a+b}\left(\psi'(a+1) - \psi'(a+1+b)\right) + \frac{b}{a+b}\left(\psi(a+1+b) - \psi(b+1)\right)^2 + \frac{b}{a+b}\left(\psi'(b+1) - \psi'(a+1+b)\right).$$

If we set $b = 1$ and a is integer, the limit for $\mathbf{E}D_n/\log n$ is

$$\frac{a+1}{\sum_{i=1}^{a} \frac{1}{i}}.$$

This grows unbounded like $a/\log a$ as $a \to \infty$. As $a \downarrow 0$, the limit is $(a+1)/(a/(a+1) + \psi(a+2) - 1) \sim 1/a$. This grows unbounded as well. In the AB trees, we have $a = b$, and thus,

$$\frac{D_n}{\log n} \to \frac{1}{\psi(2a+1) - \psi(a+1)} = \frac{1}{\sum_{n=1}^{\infty} \frac{a}{(n+a+1)(n+2a+1)}}.$$

This result matches that of Aldous (1993), where the limit is written as an integral. It is easy to verify that as $a \downarrow 0$, the limit is asymptotic to $1/(a(\pi^2/6 - 1))$. At $a = 1$, we have a limit of 2 as in the random binary search tree. As $a \to \infty$, the limit approaches $1/\log 2$, and the splits because nearly all perfectly balanced. The variance is given by

$$\sigma^2 = \left(\psi(2a+1) - \psi(a+1)\right)^2 + \left(\psi'(a+1) - \psi'(2a+1)\right).$$

Finally, consider extended AB trees in which as $a, b \to \infty$, $a = p(a+b)$ with $p \in (0,1)$ fixed. The limit then behaves as

$$\frac{1}{-p\log p - (1-p)\log(1-p)} = \frac{1}{\mathcal{H}(p)},$$

where $\mathcal{H}(p)$ is the entropy of a Bernoulli (p) random variable. Here we rediscover a known property of the entropy $\mathcal{H} = -\sum_i p_i \log(p_i)$ of a discrete distribution $(p_1, p_2, \ldots)$: split a set of size n into subsets of sizes close to $np_1, np_2, np_3, \ldots$. Associate each subset with a child of the root and repeat this process until no further splitting is possible (note that there is no randomness involved in this splitting). If one grabs a random node in the resulting tree, its depth is in probability equal to $(1/\mathcal{H})\log n$. This is exactly like the behavior of random nodes in tries (Fredkin, 1960) in which the symbols have probabilities $p_1, p_2, p_3, \ldots$; see Pittel (1985, 1986) and Szpankowski (1988) and the next section.

Example 7: *Tries and digital search trees.* Tries are b-ary trees for storing infinite strings. Assume that the data consists of n infinite strings of $\{1, \ldots, b\}$-valued symbols, called $X_1, \ldots, X_n$. Each string carves out an infinite path in the infinite complete b-ary tree. For a node u, let $N(u)$ denote the number of strings that pass through node u. Now, eliminate all nodes with $N(u) = 0$ and eliminate all those with $N(u) = 1$ whose parent also has $N(u) = 1$. The resulting tree has n leaves with $N(u) = 1$, and every nonleaf v has $N(v) > 1$. Invented in 1960 by Fredkin, this structure is called a trie. Assume that all the symbols are drawn independently, and that each symbol takes the value i with probability p_i. Define the entropy $\mathcal{H}$ by

$$\mathcal{H} = -\sum_{i=1}^{b} p_i \log p_i,$$

and the second-order entropy by

$$\mathcal{H}_2 = \sum_{i=1}^{b} p_i \log^2 p_i.$$

The trie may be viewed as a random split tree with $s = 1$, $s_0 = s_1 = 0$, in which a node u at which $N(u) = n$ is not split if $n = 1$, and in which a split occurs when $n > 1$; in the latter case, the sizes of the subtrees are distributed jointly as a multinomial $(n, p_1, \ldots, p_b)$ random variable. If V is p_S, where S is uniformly distributed on $\{1, 2, \ldots, b\}$, then

$$b\mathbf{E}\{V \log(1/V)\} = \mathcal{H}.$$

Therefore, from Theorem 2,

$$\frac{D_n}{\log n} \to \frac{1}{\mathcal{H}}.$$

Also,

$$\frac{D_n - \log n/\mathcal{H}}{\sqrt{(\mathcal{H}_2 - \mathcal{H}^2)\log n/\mathcal{H}^3}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

The law of large numbers is due to Pittel (1985). The limit law was discovered independently by Jacquet and Régnier (1986) and Pittel (1986). See Szpankowski

(1988) for additional results and references. Our result unifies the analysis of tries and binary search trees. The normal limit law as stated above is not valid if $\mathcal{H}_2 = \mathcal{H}^2$. This occurs if and only if $p_1 = p_2 = \cdots = p_b = 1/b$. In the random split tree, this situation corresponds to a monoatomic distribution for the splitter ($V \equiv 1/b$), which has zero variance.

The digital search tree of Coffman and Eve (1970) is like a trie. It is best described by its incremental construction. It has n nodes, one per string. Nodes are added one by one, starting with X_1 and ending with X_n. The node associated with X_n is the first node u in the infinite path of string X_n that has $N(u) = 0$ before X_n is inserted. This is a random split tree in which a node with $N(u) = n$ spawns subtrees when $n > 1$ of sizes that are jointly distributed as a multinomial $(n-1, p_1, \ldots, p_b)$ random variable. The limit laws given above for tries remain valid here, without change. These were known; see Pittel (1985) for the law of large numbers and Pittel (1986) and Louchard (1987) for the normal limit law. Again, when all p_i's are equal, the normal limit law as stated above is not valid. Another proof method is needed to deal with that situation. Also, Theorem 1 only states that for nondegenerate tries and digital search trees, $H_n = O(\log n)$ in probability. Theorem 1 cannot be used to get a finer result. However, the behavior of the height is well known (Pittel, 1985).

Example 8: *The random grid tree.* The quadtree is easily generalized as follows: consider a collection of m $\mathbb{R}^d$-valued points drawn from the data, and partition the space into $(m+1)^d$ hyperrectangles by the d perpendicular hyperplanes centered at each of the m points. In a quadtree, $m = 1$. This generates a tree, the m-grid tree, with fan-out $(m+1)^d$, and with up to m points per node. If the data consist of n independent random vectors uniformly distributed over $\mathbb{R}^d$, the tree thus constructed becomes a random split tree with split vector $(V_1, \ldots, V_b)$ in which each V_i is distributed as $V = \prod_{j=1}^{d} B_j$, and $B_1, \ldots B_d$ are independent beta $(1, m)$ random variables. While not exactly the same, the random grid tree borrows ideas from the celebrated grid file data structure (Nievergelt, Hinterberger, and Sevcik, 1984; Nievergelt and Hinrichs, 1993). We note the following:

$$\begin{aligned}
\mu &= \mathbf{E}\{\log W\} = (m+1)^d \mathbf{E}\{V \log(1/V)\} \\
&= (m+1)^d \sum_{j=1}^{d} \mathbf{E}\left\{ \prod_{k=1}^{d} B_k \log(1/B_j) \right\} \\
&= (m+1)^d d\, \mathbf{E}\left\{ \prod_{k=2}^{d} B_k \right\} \mathbf{E}\left\{ B_1 \log(1/B_1) \right\} \\
&= (m+1) d\, \mathbf{E}\left\{ B_1 \log(1/B_1) \right\} \\
&= d \sum_{j=2}^{m+1} \frac{1}{j}.
\end{aligned}$$

From this, we see that

$$\frac{D_n}{\log n} \to \frac{1}{d \sum_{j=2}^{m+1} \frac{1}{j}} \quad \text{in probability.}$$

Also,

$$\begin{aligned}
\sigma^2+\mu^2 &= (m+1)^d\mathbf{E}\{V\log^2(1/V)\}\\
&= (m+1)^d\mathbf{E}\left\{\prod_{k=1}^{d}B_k\left(\sum_{j=1}^{d}\log(1/B_j)\right)^2\right\}\\
&= (m+1)^d d\,\mathbf{E}\left\{\prod_{k=2}^{d}B_k\right\}\mathbf{E}\left\{B_1\log^2(1/B_1)\right\}\\
&\quad+(m+1)^d\,2d(d-1)\mathbf{E}\left\{\prod_{k=3}^{d}B_k\right\}\mathbf{E}^2\left\{B_1\log(1/B_1)\right\}\\
&= (m+1)d\,\mathbf{E}\left\{B_1\log^2(1/B_1)\right\}+2(m+1)^2d(d-1)\mathbf{E}^2\left\{B_1\log(1/B_1)\right\}\\
&= d\sum_{j=2}^{m+1}\frac{1}{j^2}+d\left(\sum_{j=2}^{m+1}\frac{1}{j}\right)^2+2d(d-1)\left(\sum_{j=2}^{m+1}\frac{1}{j}\right)^2.
\end{aligned}$$

Hence,

$$\sigma^2 = d\sum_{j=2}^{m+1}\frac{1}{j^2}+d(d-1)\left(\sum_{j=2}^{m+1}\frac{1}{j}\right)^2.$$

This yields the limit law obtained earlier for the quadtree when $m=1$. For $m=2$, we have

$$\frac{D_n-(6/5d)\log n}{\sqrt{\left(\frac{180d-102}{125d^2}\right)\log n}}\xrightarrow{\mathcal{L}}\mathcal{N}(0,1)$$

for any $d\geq 1$. For $d=1$, this coincides with the result obtained earlier for the random 3-ary search tree.

Proof of Theorem 1: *The height.* We show that $\mathbf{P}\{H_n\geq(c+3\epsilon)\log n\}\to 0$ for all $\epsilon>0$ and $c>\gamma$. Define $\delta=s_1$, $k'=\lfloor\epsilon\log n\rfloor$, and $l=k'(\delta+1)$. If n is the number of balls stored in a random split tree, then the cardinalities of the subtrees at distance k from the root are bounded from above by quantities of the form

$$\begin{aligned}
Z_k &\stackrel{\text{def}}{=} \text{binomial}\left(n,\prod_{i=1}^{k}V^{(i)}\right)+\text{binomial}\left(\delta,\prod_{i=2}^{k}V^{(i)}\right)\\
&\quad+\text{binomial}\left(\delta,\prod_{i=3}^{k}V^{(i)}\right)+\cdots+\text{binomial}\left(\delta,V^{(k)}\right)+\delta\\
&\leq \text{binomial}\left(n,\prod_{i=1}^{k}V^{(i)}\right)+\text{binomial}\left(\delta(k-k'+1),\prod_{i=k-k'+1}^{k}V^{(i)}\right)+k'\delta,
\end{aligned}$$

where $V^{(1)},\ldots,V^{(k)}$ is a sequence of i.i.d. random variables distributed as V, and the inequality is in a stochastic sense only. For each of the b^k paths down to a node at distance k, a different sequence is obtained. Thus, by Boole's inequality and the fact that all splitters are identically distributed, we have for integer $k,l>0$,

$$\mathbf{P}\{H_n\geq k+3l\}\leq b^k\mathbf{P}\{Z_k\geq 3l\}.$$

We further argue as follows:

$$\begin{aligned}\mathbf{P}\{Z_k \geq 3l\} \leq & \mathbf{P}\left\{\text{binomial}\left(n, \prod_{i=1}^{k} V^{(i)}\right) \geq l\right\} \\ & + \mathbf{P}\left\{\text{binomial}\left(\delta(k-k'+1), \prod_{i=k-k'+1}^{k} V^{(i)}\right) \geq l\right\} \\ & + \mathbf{P}\left\{k'\delta \geq l\right\}.\end{aligned}$$

The last term is zero by the choice of l. Conditioned on the $V^{(i)}$'s, the first term is easily bounded using Markov's inequality and Chernoff's bounding method. Let $t > 0$ be picked later. Then, if $Z = \prod_{i=1}^{k} V^{(i)}$,

$$\begin{aligned}\mathbf{P}\{\text{binomial}(n, Z) \geq l | Z\} & \leq \mathbf{E}\left\{(1 - Z + Ze^t)^n | Z\right\} e^{-tl} \\ & \leq \mathbf{E}\left\{e^{(e^t-1)nZ} | Z\right\} e^{-tl} \\ & \leq \mathbf{E}\left\{e^{l - nZ + l\log(nZ)} | Z\right\} \quad (\text{take } e^t = l/(nZ)).\end{aligned}$$

Take the expectation with respect to Z. The inequality is then further developed by noting the following: for $z \in (0,1)$ and $t > 0$,

$$\begin{aligned}\mathbf{P}\{\text{binomial}(n, Z) \geq l\} & \leq e^{l - z + l\log(z)} + \mathbf{P}\{nZ > z\} \\ & \leq (ez)^l + (n/z)^t \mathbf{E}\{Z^t\}.\end{aligned}$$

Similarly, for $z' \in (0,1)$, $t' > 0$, and $Z' = \prod_{i=k-k'+1}^{k} V^{(i)}$,

$$\begin{aligned}\mathbf{P}\{\text{binomial}(\delta(k-k'+1), Z') \geq l\} & \leq e^{l - z' + l\log(z')} + \mathbf{P}\{\delta(k-k'+1)Z' > z'\} \\ & \leq (ez')^l + (\delta(k-k'+1)/z')^{t'} \mathbf{E}\{Z'^{t'}\}.\end{aligned}$$

Take $z = z' = b^{-2k/l}/e$ and note that $ze \sim b^{-2c/\epsilon(\delta+1)}$. Then, combining the previous bounds,

$$\begin{aligned}b^k \mathbf{P}\{Z_k \geq 3l\} & \leq 2b^{-k} + b^k (ne)^t b^{2kt/l} m(t)^k + b^k (\delta(k-k'+1)e)^{t'} b^{2kt'/l} m(t')^{k'} \\ & \stackrel{\text{def}}{=} I + II + III,\end{aligned}$$

where $m(t)$ is the tth moment of V. Clearly, $I = o(1)$. Choose t' large enough so that $b\, m(t')^{\epsilon/c} < 1$. This is possible, as $\mathbf{P}\{V = 1\} = 0$ and thus $m(t') \to 0$ as $t' \to \infty$ (Lemma 1). With this choice of t', $III = o(1)$. To treat II, fix t and observe that $II = o(1)$ if

$$b^k n^t m(t)^k \to 0,$$

which occurs if

$$(bm(t))^c e^t < 1 \text{ or, equivalently, } c\log(bm(t)) + t < 0.$$

Here we distinguish between the two statements in the theorem. For the first statement, take t so large that $bm(t) < 1$. Then take c large enough to ensure that $(bm(t))^c e^t < 1$. For the second statement, we must be a bit more careful. The

minimal value of $c\log(bm)+t$ is obtained at the solution $t^*=t^*(c)$ of the equation $m'(t)/m(t)=-1/c$. From Lemma 1, a solution exists when

$$-\frac{1}{\mathbf{E}\{\log(V)\}}<c<-\frac{1}{\log v_\infty}.$$

Replace t by t^* (as we are allowed to choose any positive t) and let $c>\gamma$. As $R<-\log b$, $c\log(bm(t^*))+t^*<0$, and thus $II=o(1)$. This concludes the proof of Theorem 1.

Proof of Theorem 2: *The depth.* The following lemma will be useful for bounding tail probabilities.

LEMMA 4. *If X is binomial (n,Z) (written $B_{n,Z}$) where $Z\in[0,1]$ is a random variable, then for $0<a<n$,*

$$\mathbf{P}\{X\geq a\}\leq\mathbf{P}\{Z>a/(2n)\}+\left(\frac{e}{4}\right)^{a/2}.$$

Similarly,

$$\mathbf{P}\{X\leq a\}\leq\mathbf{P}\{Z<2a/n\}+\left(\frac{2}{e}\right)^{a}.$$

Proof. If X is binomial (n,p), then, for $1>u\geq p$ by Chernoff's bound (Chernoff, 1952; Okamoto, 1958),

$$\mathbf{P}\{X\geq nu\}\leq\left(\left(\frac{p}{u}\right)^u\left(\frac{1-p}{1-u}\right)^{1-u}\right)^n.$$

Interestingly, the same bound applies for $\mathbf{P}\{X\leq nu\}$ if $0<u\leq p$. In particular,

$$\mathbf{P}\{X\geq 2np\}\leq\left(\left(\frac{1}{2}\right)^{2p}\left(\frac{1-p}{1-2p}\right)^{1-2p}\right)^n\leq\left(\left(\frac{1}{2}\right)^{2p}e^p\right)^n=\left(\frac{e}{4}\right)^{np}.$$

Also,

$$\mathbf{P}\{X\leq np/2\}\leq\left((2)^{p/2}\left(1-\frac{p/2}{1-p/2}\right)^{1-p/2}\right)^n\leq\left((2)^{p/2}e^{-p/2}\right)^n=\left(\sqrt{\frac{2}{e}}\right)^{np}.$$

Applying this, we have

$$\begin{aligned}\mathbf{P}\{X>a\}&\leq\mathbf{P}\{Z>a/(2n)\}+\mathbf{P}\{B_{n,a/(2n)}>a\}\\&\leq\mathbf{P}\{Z>a/(2n)\}+\left(\frac{e}{4}\right)^{a/2}.\end{aligned}$$

Similarly, assuming without loss of generality that $n\geq 2$,

$$\begin{aligned}\mathbf{P}\{X\leq a\}&\leq\mathbf{P}\{Z<2a/n\}+\mathbf{P}\{B_{n,2a/n}\leq a\}\\&\leq\mathbf{P}\{Z<2a/n\}+\left(\frac{2}{e}\right)^{a}.\qquad\square\end{aligned}$$

The convergence of $\mathbf{E}\{D_n\}$ follows from the weak convergence,

$$\mathbf{E}\{D_n\}=\int_0^1\mathbf{P}\{D_n>t\}\,dt,$$

and the tail probabilities for $\mathbf{P}\{D_n > t\}$ developed below. The details are omitted.

For the proof of Theorem 2, we consider an infinite random path in the tree, $u_0, u_1, u_2, \ldots$, where u_0 is the root, and given u_i and the split vector $(V_1, \ldots, V_b)$ for u_i, u_{i+1} is the j-child of u_i with probability V_j. Put n balls in the tree as in the construction of a random split tree, and let u^* be the unique leaf on the infinite path. Then D_n is less than or equal to the distance between u^* and the root.

We first show that for all $c > 1/\mu$, $\mathbf{P}\{D_n > c \log n\} \to 0$. Take $k = \lfloor c \log n \rfloor$. Let u_0 be the root, and let u_0, u_1, ... be the path of nodes followed by the inserted point from the root down. We have, if $\beta = (s_0 + 1)k$,

$$[D_n > k + l] \subseteq [N(u_k) > \beta] \cup [H_\beta > l],$$

where H_β is the height of a random split tree with β balls. But

$$\mathbf{P}\{N(u_k) > \beta\} \leq \mathbf{P}\left\{ks_0 + \text{binomial}\left(n, \prod_{i=0}^{k-1} W_i\right) > \beta\right\},$$

where $W_0, W_1, \ldots$ are i.i.d. random variables distributed as $W = V_S$, and $S = i$ with probability V_i. Here we made use of the fact that a binomial (N, p) in which N is binomial (n, q) is distributed as a binomial (n, pq). By Lemma 4, we see that

$$\begin{aligned}
\mathbf{P}\left\{\text{binomial}\left(n, \prod_{i=0}^{k-1} W_i\right) > \beta - ks_0\right\} &\leq \mathbf{P}\left\{\prod_{i=0}^{k-1} W_i > \frac{\beta - ks_0}{2n}\right\} + \left(\frac{e}{4}\right)^{\frac{\beta - ks_0}{2}} \\
&= \mathbf{P}\left\{\sum_{i=0}^{k-1} \log W_i > \log\left(\frac{\beta - ks_0}{2n}\right)\right\} + \left(\frac{e}{4}\right)^{\frac{k}{2}} \\
&= I + II.
\end{aligned}$$

Clearly, $II = o(1)$. Also, $I = o(1)$ by the law of large numbers, as

$$\frac{\sum_{i=0}^{k-1} \log W_i}{k\mathbf{E}\{\log W\}} \to 1$$

almost surely. Recall that $\mu = \mathbf{E}\{\log(1/W)\}$. Also, we used the fact that

$$\liminf_{n\to\infty} \frac{\log\left(\frac{\beta - ks_0}{2n}\right) + k\mu}{k} = -\frac{1}{c} + \mu > 0$$

since $c > 1/\mu$. To wrap up the proof of the first part, we must show that $\mathbf{P}\{H_\beta > l\} \to 0$, where l is our choice. Let us pick $l = \lfloor 2\gamma \log \beta \rfloor = \lfloor 2\gamma \log((s_0 + 1)k) \rfloor$, where $\gamma > 0$ is as in Theorem 1. Then $\mathbf{P}\{H_\beta > l\} \to 0$, because

$$\lim_{n\to\infty} \mathbf{P}\{H_\beta > 2\gamma \log \beta\} = 0.$$

As $l \sim \log \log n$, the first part of the law of large numbers is proved.

Next, we show that for all $c < 1/\mu$, $\mathbf{P}\{D_n < c \log n\} \to 0$. Take $k = \lfloor c \log n \rfloor$. Then, if $N(.)$ refers to the tree with $n - 1$ balls,

$$[D_n < k] \subseteq [N(u_k) = 0].$$

But

$$\mathbf{P}\{N(u_k)=0\} \le \mathbf{P}\left\{-ks + \text{binomial}\left(n-1, \prod_{i=0}^{k-1} W_i\right) \le 0\right\},$$

where $W_0, W_1, \ldots$ are as in the earlier part of this proof. By Lemma 4, we see that

$$\begin{aligned}\mathbf{P}\left\{\text{binomial}\left(n-1, \prod_{i=0}^{k-1} W_i\right) \le ks\right\} &\le \mathbf{P}\left\{\prod_{i=0}^{k-1} W_i \le \frac{2ks}{n-1}\right\} + \left(\frac{2}{e}\right)^{ks} \\ &= \mathbf{P}\left\{\sum_{i=0}^{k-1} \log W_i \le \log\left(\frac{2ks}{n-1}\right)\right\} + \left(\frac{2}{e}\right)^{ks} \\ &= I + II.\end{aligned}$$

Obviously, $II = o(1)$. $I = o(1)$ by the law of large numbers, as

$$\frac{\sum_{i=0}^{k-1} \log W_i}{k\mathbf{E}\{\log W\}} \to 1$$

almost surely and

$$\limsup_{n\to\infty} \frac{\log\left(\frac{2ks}{n}\right) - k\mathbf{E}\{\log W\}}{k} = -\frac{1}{c} - \mathbf{E}\{\log W\} = -\frac{1}{c} + \mu < 0.$$

This concludes the proof of the lower bound for the law of large numbers.

The limit law is obtained by using the same upper and lower bounds introduced in the proof of the law of large numbers. Additionally, we will use the fact that

$$\frac{\sum_{i=0}^{k-1} \log W_i + k\mu}{\sqrt{k\sigma^2}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

Consider first the probability $\mathbf{P}\{D_n > k\}$, where $k = \lfloor (1/\mu)\log n + u\sqrt{\log n}\rfloor$ and $u \in \mathbb{R}$. We have, if $\beta = (s_0+1)k$,

$$[D_n > k + l] \subseteq [N(u_k) > \beta] \cup [H_\beta > l],$$

where H_β is the height of a random split tree with β balls. Arguing as in the first part of this proof,

$$\begin{aligned}\mathbf{P}\{N(u_k) > \beta\} &\le \mathbf{P}\left\{\text{binomial}\left(n, \prod_{i=0}^{k-1} W_i\right) > \beta - ks_0\right\} \\ &= \mathbf{P}\left\{\sum_{i=0}^{k-1} \log W_i > \log\left(\frac{\beta - ks_0}{2n}\right)\right\} + \left(\frac{e}{4}\right)^{\frac{k}{2}} \\ &= \mathbf{P}\left\{\frac{\sum_{i=0}^{k-1} \log W_i + k\mu}{\sqrt{k\sigma^2}} > \frac{\log\left(\frac{\beta-ks_0}{2n}\right) + k\mu}{\sqrt{k\sigma^2}}\right\} + o(1) \\ &= \mathbf{P}\left\{\mathcal{N}(0,1) > \frac{\log\left(\frac{\beta-ks_0}{2n}\right) + k\mu}{\sqrt{k\sigma^2}}\right\} + o(1) \\ &= \mathbf{P}\left\{\mathcal{N}(0,1) > \frac{u\mu^{3/2}}{\sigma}\right\} + o(1).\end{aligned}$$

Recall that for $l = \lfloor 2\gamma \log \beta \rfloor = \lfloor 2\gamma \log((s_0+1)k) \rfloor$, we obtain $\mathbf{P}\{H_\beta > l\} \to 0$. As $l \sim \log\log n$, the first part of the limit law is proved:

$$\mathbf{P}\{D_n > k\} \le \mathbf{P}\{N > u\mu^{3/2}/\sigma\} + o(1).$$

Using the arguments for the lower bound, we may prove in a similar fashion that

$$\mathbf{P}\{D_n < k\} \le \mathbf{P}\{N < u\mu^{3/2}/\sigma\} + o(1).$$

Taken together, this proves that

$$\lim_{n\to\infty} \mathbf{P}\{D_n < k\} = \mathbf{P}\{N < u\mu^{3/2}/\sigma\},$$

which was to be shown.

Other possible universal models for random split trees. We could have developed this theory based on other models. In a random split tree, the subtree sizes are multinomial $(n, V_1, \ldots, V_b)$, where $(V_1, \ldots, V_b)$ in turn is a random split vector. This introduces two levels of randomization. A more rigid and perhaps less universal model would fix an integer δ and require that the subtree sizes $N(u_1), \ldots, N(u_b)$ for the children $u_1, \ldots, u_b$ of a node u satisfy:

$$\max_{1\le i\le b} |N(u_i) - nV_i| \le \delta.$$

Some of the trees discussed earlier fall into this framework. For example, in a random binary search tree, it is well known that the left and right subtrees of the root have cardinalities (N_1, N_2) that are jointly distributed as $(\lfloor nU \rfloor, \lfloor n(1-U) \rfloor)$, where U is uniform $[0,1]$ and n is the cardinality of the tree. Setting $(V_1, V_2) = (U, 1-U)$, we thus have $\max_i |N_i - nV_i| \le 1$. Theorems 1 and 2 have straightforward equivalent versions (with the same dependence upon μ, σ^2 and $m(t)$). We should note that for generating extended AB trees for the purpose of simulation, the model of this section is more convenient. Here the split vectors V_1 and $V_2 = 1 - V_1$ are mixtures of beta random variables, but no multinomial sampling is necessary, as we use $(N_1, N_2) = (\lfloor nV_1 \rfloor, \lfloor nV_2 \rfloor)$ to determine subtree sizes at the root of a subtree of cardinality n. This way, each node will receive one ball.

Acknowledgment. I thank Paul Kruszewski and two anonymous referees for great feedback on the manuscript.

REFERENCES

M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Tables*, Dover, New York, 1970.

A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Reading, MA, 1983.

D. Aldous, B. Flannery, and J. L. Palacios, *Two applications of urn processes: The fringe analysis of search trees and the simulation of quasi-stationary distributions of Markov chains*, Probab. Engrg. Inform. Sci., 2 (1988), pp. 293–307.

D. Aldous, *Probability Distributions on Cladograms*, Technical Report, Institute of Mathematics and Applications, University of Minnesota, Minneapolis, 1993.

E. Arkin, M. Held, J. Mitchell, and S. Skiena, *Hamiltonian triangulations for fast rendering*, in Algorithms–ESA'94, Lecture Notes in Computer Science 855, J. van Leeuwen, ed., Springer-Verlag, New York, 1994, pp. 36–47.

K. B. Athreya and P. E. Ney, *Branching Processes*, Springer-Verlag, Berlin, 1972.

D. Avis and H. El Gindy, *Triangulating point sets in space*, Disc. Comput. Geom., 2 (1987), pp. 99–111.

C. J. Bell, *An Investigation into the Principles of the Classification and Analysis of Data of an Automatic Digital Computer*, Ph.D. Thesis, Leeds University, UK, 1965.
J. D. Biggins, *The first and last-birth problems for a multitype age-dependent branching process*, Adv. Appl. Probab., 8 (1976), pp. 446–459.
J. D. Biggins, *Chernoff's theorem in the branching random walk*, J. Appl. Probab., 14 (1977), pp. 630–636.
H. Chernoff, *A measure of asymptotic efficiency of tests of a hypothesis based on the sum of observations*, Ann. Math. Stat., 23 (1952), pp. 493–507.
E. G. Coffman and J. Eve, *File structures using hashing functions*, Comm. ACM, 13 (1970), pp. 427–436.
L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, New York, 1986a.
L. Devroye, *A note on the height of binary search trees*, J. Assoc. Comput. Mach., 33 (1986b), pp. 489–498.
L. Devroye, *Branching processes in the analysis of the heights of trees*, Acta Inform., 24 (1987), pp. 277–298.
L. Devroye, *Applications of the theory of records in the study of random trees*, Acta Inform., 26 (1988), pp. 123–130.
L. Devroye, *On the height of random m-ary search trees*, Random Structures Algorithms, 1 (1990), pp. 191–203.
L. Devroye, *On the expected height of fringe-balanced trees*, Acta Inform., 30 (1993), pp. 459–466.
L. Devroye and L. Laforest, *An analysis of random d-dimensional quadtrees*, SIAM J. Comput., 19 (1990), pp. 821–832.
S. W. Dharmadhikari and K. Jogdeo, *Bounds on moments of certain random variables*, Ann. Math. Stat., 40 (1969), pp. 1506–1508.
R. A. Finkel and J. L. Bentley, *Quad trees: A data structure for retrieval on composite keys*, Acta Inform., 4 (1974), pp. 1–9.
P. Flajolet, G. Gonnet, C. Puech, and J. M. Robson, *The analysis of multidimensional searching in quad-trees*, in Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1991, pp. 100–109.
P. Flajolet and T. Lafforgue, *Search costs in quadtrees and singularity perturbation analysis*, Disc. Comput. Geom., 12 (1994), pp. 151–175.
P. Flajolet and A. Odlyzko, *The average height of binary trees and other simple trees*, J. Comput. System Sci., 25 (1982), pp. 171–213.
P. Flajolet and R. Sedgewick, *Digital search trees revisited*, SIAM J. Comput., 15 (1986), pp. 748–767.
E. H. Fredkin, *Trie memory*, Comm. ACM, 3 (1960), pp. 490–500.
D. K. Fuk and S. V. Nagaev, *Probability inequalities for sums of independent random variables*, Theory Probab. Appl., 16 (1971), pp. 643–660.
G. H. Gonnet and R. Baeza-Yates, *Handbook of Algorithms and Data Structures*, Addison-Wesley, Workingham, UK, 1991.
I. S. Gradshteyn and I. M. Ryzhik, *Table of Integrals, Series and Products*, Academic Press, New York, 1980.
G. R. Grimmett and D. R. Stirzaker, *Probability and Random Processes*, Oxford University Press, Oxford, UK 1992.
J. M. Hammersley, *Postulates for subadditive processes*, Ann. Probab., 2 (1974), pp. 652–680.
P. Jacquet and M. Régnier, *Trie Partitioning Process: Limiting Distributions*, Lecture Notes in Computer Science 214, Springer-Verlag, New York, 1986, pp. 196–210.
R. Kemp, *Fundamentals of the Average Case Analysis of Particular Algorithms*, B. G. Teubner, Stuttgart, 1984.
J. F. C. Kingman, *Subadditive ergodic theory*, Ann. Probab., 1 (1973), pp. 883–909.
D. E. Knuth, *The Art of Computer Programming, Vol.* 3: *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
G. Louchard, *Exact and asymptotic distributions in digital and binary search trees*, Theoret. Inform. Appl., 21 (1987), pp. 479–496.
W. C. Lynch, *More combinatorial problems on certain trees*, Comput. J., 7 (1965), pp. 299–302.
H. M. Mahmoud, *On the average internal path length of m-ary search trees*, Acta Inform., 23 (1986), pp. 111–117.
H. M. Mahmoud, *Evolution of Random Search Trees*, John Wiley, New York, 1992.
H. Mahmoud and B. Pittel, *On the most probable shape of a search tree grown from a random permutation*, SIAM J. Algebraic Disc. Meth., 5 (1984), pp. 69–81.
H. M. Mahmoud and B. Pittel, *On the joint distribution of the insertion path length and the number of comparisons in search trees*, Disc. Appl. Math., 20 (1988), pp. 243–251.

J. Marcinkiewicz and A. Zygmund, *Sur les fonctions indépendantes*, Fundamentales de Mathématiques, 29 (1937), pp. 60–90.
S. V. Nagaev and N. F. Pinelis, *Some inequalities for sums of independent random variables*, Theory Probab. Appl., 22 (1977), pp. 248–256.
J. Nievergelt and K. H. Hinrichs, *Algorithms and Data Structures with Applications to Graphics and Geometry*, Prentice-Hall, Englewood Cliffs, NJ, 1993.
J. Nievergelt, H. Hinterberger, and K.C. Sevcik, *The grid file: An adaptable, symmetric multikey file structure*, ACM Trans. on Database Systems, 9 (1984), pp. 38–71.
M. Okamoto, *Some inequalities relating to the partial sum of binomial probabilities*, Ann. Math. Statist., 10 (1958), pp. 29–35.
V. V. Petrov, *Sums of Independent Random Variables*, Springer-Verlag, Berlin, 1975.
B. Pittel, *On growing random binary trees*, J. Math. Anal. Appl., 103 (1984), pp. 461–480.
B. Pittel, *Asymptotical growth of a class of random trees*, Ann. Probab., 13 (1985), pp. 414–427.
B. Pittel, *Paths in a random digital tree: Limiting distributions*, Adv. Appl. Probab., 18 (1986), pp. 139–155.
B. Pittel, *Note on the heights of random recursive trees and random m-ary search trees*, Random Structures Algorithms, 5 (1994), pp. 337–347.
P. V. Poblete and J. I. Munro, *The analysis of a fringe heuristic for binary search trees*, J. Algorithms, 6 (1985), pp. 336–350.
P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*, Springer-Verlag, New York, 1990
R. Pyke, *Spacings*, J. Roy. Statist. Soc. Ser. B, 7 (1965), pp. 395–445.
J. M. Robson, *The height of binary search trees*, Austral. Comput. J., 11 (1979), pp. 151–153.
R. Y. Rubinstein, *Generating random vectors uniformly distributed inside and on the surface of different regions*, European J. Oper. Res., 10 (1982), pp. 205–209.
H. Samet, *Applications of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990a.
H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990b.
R. Sedgewick, *Mathematical analysis of combinatorial algorithms*, in Probability Theory and Computer Science, G. Louchard and G. Latouche, eds., Academic Press, London, 1983, pp. 123–205.
M. Sibuya, *Generalized hypergeometric, digamma and trigamma distributions*, Ann. Inst. Statist. Math., 31 (1979), pp. 373–390.
R.L. Smith, *Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions*, Oper. Res., 32 (1984), pp. 1296–1308.
W. Szpankowski, *Some results on V-ary asymmetric tries*, J. Algorithms, 9 (1988), pp. 224–244.
X.G. Viennot, *Trees everywhere*, in CAAP 90, Lecture Notes in Computer Science 431, A. Arnold, ed., Springer-Verlag, Berlin, 1990, pp. 18–41.
J. S. Vitter and P. Flajolet, *Average-case analysis of algorithms and data structures*, in Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity, J. van Leeuwen, ed., MIT Press, Amsterdam, 1990, pp. 431–524.
A. Walker and D. Wood, *Locally balanced binary trees*, Comput. J., 19 (1976), pp. 322–325.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 433–446

AVERAGE-CASE LOWER BOUNDS FOR NOISY BOOLEAN DECISION TREES*

WILLIAM EVANS† AND NICHOLAS PIPPENGER‡

Abstract. We present a new method for deriving lower bounds to the expected number of queries made by noisy decision trees computing Boolean functions. The new method has the feature that expectations are taken with respect to a uniformly distributed random input, as well as with respect to the random noise, thus yielding stronger lower bounds. It also applies to many more functions than do previous results. The method yields a simple proof of the result (previously established by Reischuk and Schmeltz) that almost all Boolean functions of n arguments require $\Omega(n \log n)$ queries, and strengthens this bound from the worst-case over inputs to the average over inputs. The method also yields bounds for specific Boolean functions in terms of their spectra (their Fourier transforms). The simplest instance of this spectral bound yields the result (previously established by Feige, Peleg, Raghavan, and Upfal) that the parity function of n arguments requires $\Omega(n \log n)$ queries and again strengthens this bound from the worst-case over inputs to the average over inputs. In its full generality, the spectral bound applies to the "highly resilient" functions introduced by Chor, Friedman, Goldreich, Hastad, Rudich, and Smolensky, and it yields nonlinear lower bounds whenever the resiliency is asymptotic to the number of arguments.

Key words. fault-tolerance, reliability, noisy computation, error-correction

AMS subject classifications. 68M15, 68P10, 68R05

PII. S0097539796310102

1. Introduction.

We shall deal in this paper with dynamic decision trees for computing Boolean functions. A *dynamic decision tree* is a binary tree in which each internal node N is labelled with an argument index $\alpha(N) \in \{1, \ldots, n\}$, each child M of an internal node N is labelled with a Boolean value $\beta(M) \in \{0, 1\}$ that might be assumed by this argument (with siblings being labelled with distinct values), and each leaf L is labelled with a Boolean function value $\phi(L) \in \{0, 1\}$. Such a dynamic decision tree computes a Boolean function f of n Boolean arguments $x_1, \ldots, x_n$ in an obvious way: start at the root; when at an internal node N, query the argument $x_{\alpha(N)}$ and proceed to the child M of N such that $\beta(M) = x_{\alpha(N)}$; when at a leaf L, announce the function value $f(x_1, \ldots, x_n) = \phi(L)$. For such a dynamic decision tree, we may speak of the *worst-case cost* (the maximum over argument values of the depth of the leaf that announces the function value) or the *average-case cost* (the average with a uniform distribution over argument values of the depth of the leaf that announces the function value).

We shall be interested in the situation in which dynamic decision trees are noisy, that is, in which each internal node independently passes control to the incorrect child (that is, the child M of the internal node N such that $\beta(M) = \neg x_{\alpha(N)}$) with some fixed probability $0 < \varepsilon < 1/2$. We shall say that such a tree (ε, δ)-computes a

*Received by the editors October 2, 1996; accepted for publication (in revised form) April 10, 1997; published electronically July 7, 1998. A preliminary version of this paper appeared in *Proc. 28th Annual ACM Symposium on Theory of Computing*, ACM, New York, 1996, pp. 620–628.
http://www.siam.org/journals/sicomp/28-2/31010.html

†Department of Computer Science, The University of Arizona, Tucson, AZ 85721-0077 (will@cs.arizona.edu). The work of this author was supported by an NSERC Canada International Fellowship.

‡Department of Computer Science, The University of British Columbia, Vancouver, BC V6T 1Z4 Canada (nicholas@cs.ubc.ca). The work of this author was supported by an NSERC Operating Grant.

Boolean function f if, for all $x_1, \ldots, x_n \in \{0,1\}$, the probability that control reaches an incorrectly labelled leaf (that is, a leaf L labelled $\phi(L) = \neg f(x_1, \ldots, x_n)$) is at most $\delta < 1/2$. For such a noisy dynamic decision tree, we may again speak of the worst-case or average-case cost (where we may maximize or average over argument values but always average over noise).

An alternative to the error model we have adopted is to assume that errors occur with probability at most ε, rather than exactly ε. This alternative model gives stronger upper bounds. Our interest in this paper is in lower bounds, for which the model we have adopted gives stronger results.

To describe the history of our results, we shall need to refer to two additional computational models. The first of these is the *static decision tree*, which we may regard as a dynamic decision tree in which the argument queried by an internal node does not depend on the outcomes of previous queries (and thus depends only on the depth of the node in the tree), and in which all leaves appear at the same depth. The cost in this case is simply the common depth C of the leaves. It is not hard to see that we may ignore the tree structure, and simply focus on the number of queries C_i to each argument x_i. We then have $C_1 + \cdots + C_n = C$. Furthermore, we may ignore the sequence of answers to the queries to a given argument and focus on the number D_i of affirmative answers among answers to the C_i queries to x_i. We then have $0 \leq D_i \leq C_i$ for $1 \leq i \leq n$. While a noisy static decision tree might announce distinct function values for the same values of $D_1, \ldots, D_n$, it is not hard to see that these announcements can be replaced by a consistent announcement $\phi(D_1, \ldots, D_n)$, without increasing the probability of an incorrect announcement in any situation. Thus we may describe a static decision tree by specifying the numbers $C_1, \ldots, C_n$ and the labelling $\phi(D_1, \ldots, D_n)$ for $0 \leq D_1 \leq C_1, \ldots, 0 \leq D_n \leq C_n$.

Our final computational model is the *circuit with noisy gates*. We shall not describe this model in detail but merely remark that a lower bound to static decision tree cost yields a lower bound to the size (number of gates) of a circuit with noisy gates.

Work on reliable computation in the presence of noise was begun by von Neumann [14], who argued (although he did not give a rigorous proof) that a computation that can be performed by a noiseless network with L gates could be reliably performed by a noisy network with $O(L \log L)$ gates. Dobrushin and Ortyukov [4] provided a rigorous proof of this result, and [3] claimed the following matching lower bound: a noisy network that reliably computes a function f must have $\Omega(S \log S)$ gates, where S is the *sensitivity* of f (the maximum over inputs $x_1, \ldots, x_n$ of the number of indices i such that $f(x_1, \ldots, x_{i-1}, \neg x_i, x_{i+1}, \ldots, x_n) \neq f(x_1, \ldots, x_n)$). Since there are many functions (for example, the disjunction, conjunction, or parity of n arguments) that have sensitivity $S = n$ and can be computed by noiseless networks with $O(n)$ gates, this result shows that the logarithmic ratio of noisy to noiseless gates is necessary for certain functions.

There are, however, several errors in the proof of the lower bound of Dobrushin and Ortyukov [3]. These were pointed out by Pippenger, Stamoulis, and Tsitsiklis [16], who gave a proof of the weaker result that a noisy network that reliably computes the parity function of n arguments must have $\Omega(n \log n)$ gates. The full strength of the lower bound in terms of sensitivity was regained by Gál [9] (see also Gács and Gál [10]) and by Reischuk and Schmeltz [17]. An important consequence of this stronger result is that a noisy network that reliably computes the disjunction (or conjunction) of n arguments must have $\Omega(n \log n)$ gates. All of these lower bound

arguments apply to static decision trees as well as to circuits. For noisy static decision trees, lower bounds of $\Omega(n \log n)$ are best possible, since any Boolean function of n arguments can be computed by a noisy static decision tree with $O(n \log n)$ queries (with $2\log(n/\delta) \,/\, \log\bigl(1/4\varepsilon(1-\varepsilon)\bigr) = O(\log n)$ queries, it is possible to determine a single argument with error probability at most δ/n).

Noisy dynamic decision trees were considered by Feige et al. [6, 7], who showed that there are noisy dynamic decision trees that reliably compute the disjunction or conjunction of n arguments with $O(n)$ queries. Since we have seen that noisy static decision trees require $\Omega(n \log n)$ queries, this exhibits a clear separation between the two models. For noisy dynamic decision trees, Feige et al. [6, 7] showed that $\Omega(n \log n)$ queries are needed to compute the parity or majority of n arguments, and Reischuk and Schmeltz [17] showed that $\Omega(n \log n)$ queries are needed for almost all Boolean functions of n arguments. (This last result contrasts with results of Muller [13] and Pippenger [15] for circuits, to the effect that for almost all Boolean functions of n arguments, $\Omega(2^n/n)$ noiseless gates are necessary and $O(2^n/n)$ noisy gates are sufficient.) The lower bound proofs of both Feige et al. and of Reischuk and Schmeltz depend on locating particular sets of inputs that are difficult for a dynamic decision tree, and thus they yield lower bounds for the worst-case over inputs but not for the average-case over inputs (and clearly no proof that applied to disjunction or conjunction could give a nontrivial lower bound for the average over inputs).

The present paper gives a new method of establishing lower bounds for noisy dynamic decision trees. The gist of the method is to argue that for certain Boolean functions there cannot be even one leaf in the decision tree that has both a small depth and a small probability of error (conditional on control reaching the leaf). The Boolean functions to which the method applies are difficult to compute for all inputs rather than just for certain inputs. This implies that lower bounds established by the method apply to the average case over inputs rather than just the worst case. (It also implies of course that the method is powerless to deal with functions such as disjunction, conjunction, and majority that have inputs such as $x_1 = \cdots x_n = 1$, $x_1 = \cdots x_n = 0$, or both for which it is easy to reliably determine the function value.) These strengths and weaknesses of our new method are embodied in a new complexity measure for Boolean functions, which we call "noisy leaf complexity." In section 2 we shall define noisy leaf complexity and relate it to noisy dynamic decision tree complexity described above.

Our method considers the situation in which control has arrived at a leaf L. Arrival at L conditions the uniform prior distribution on the input x to a posterior distribution. Our method is based on the fact that, if the depth of L is small, this posterior distribution must be spread over a large range of possible input values. In section 3 we shall calculate this posterior distribution and derive quantitative versions of the assertion that it is spread over a large range.

Section 4 deals with random Boolean functions and establishes a lower bound of the form $\Omega(n \log n)$ for the noisy leaf complexity of "almost all" Boolean functions of n arguments. Specifically, we show that if L is a leaf of cost

$$C \le n \log_E(n/2) - n \log_E \log\bigl(2n^2/(1-2\delta)^2\bigr),$$

where $E = (1-\varepsilon)/\varepsilon$, then the probability is at most $2e^{-n^2}$ that L has error probability (conditional on arrival at L) at most δ for a random Boolean function of n arguments. This strengthens (from the worst-case over inputs to the average-case over inputs) the lower bound of Reischuk and Schmeltz [17].

Section 5 establishes a lower bound of the form $\Omega(n \log n)$ for the noisy leaf complexity of the parity function of n arguments. Specifically, we show that if a leaf with cost C has conditional error probability at most δ for the parity function of n arguments, then

$$C \geq n \log_E n - n \log_E \log\bigl(1/(1-2\delta)\bigr),$$

where $E = (1-\varepsilon)/\varepsilon$. This strengthens (from the worst-case over inputs to the average-case over inputs) the lower bound of Feige et al. [6, 7] for the parity function. The proof of our lower bound uses the Fourier transform of the parity function, which has a particularly simple form. Other examples of the use of the Fourier transform to derive lower bounds to the computational complexity of Boolean functions are given by Brandman, Orlitsky, and Hennessy [1] (noiseless decision trees) and by Linial, Mansour, and Nisan [12] (bounded-depth circuits). It would be possible to rephrase this proof so as not to refer to the Fourier transform. Indeed, Fourier analysis on finite groups such as the Boolean n-cube is tantamount to linear algebra in finite-dimensional vector spaces. Fourier analysis lends this linear algebra a certain suggestive terminology, however, that provides a vivid intuition to guide the manipulations. This intuition was valuable in discovering the more general results of section 6.

A general class of Boolean functions to which our method applies is the class of "highly resilient" functions. If a Boolean function is significantly "biased" (that is, if it assumes the values 0 and 1 with significantly unequal probabilities under the uniform input distribution), then even a leaf at depth 0 can announce the function value with a probability of output error significantly less than $1/2$. This suggests we focus our attention on "unbiased" functions, which assume the values 0 and 1 each with probability $1/2$. Extending this reasoning, we see that if a Boolean function can be significantly biased by substituting constants for a small number of arguments, then a leaf with small depth can achieve a probability of output error significantly less than $1/2$. This suggests we focus our attention on functions that are unbiased and which remain unbiased even when constants are substituted for some number t of arguments. Such functions are called "t-resilient" by Chor et al. [2]. Though defined combinatorially, the highly resilient functions have natural characterizations in terms of their "spectra," either in the sense of their Fourier transforms or in the sense of the eigenvalues of the adjacency matrix of the Boolean hypercube. These characterizations are discussed by Friedman [8].

Section 6 establishes a lower bound for the noisy leaf complexity of t-resilient Boolean functions. Specifically, we show that if f is t-resilient and a leaf with cost C has conditional error probability at most δ for f, then

$$C \geq (t+1) \log_E \frac{t+1}{\frac{n}{2} H\left(\frac{t+1}{n}\right) + \log \frac{1}{1-2\delta}},$$

where $E = (1-\varepsilon)/\varepsilon$, and $H(\eta) = -\eta \log \eta - (1-\eta)\log(1-\eta)$ for $0 < \eta < 1$, extended by continuity to $H(0) = H(1) = 0$. The most resilient function of n arguments is the parity function, which is $(n-1)$-resilient. Thus we recover the lower bound of section 5 in this special case. There are, however, many highly resilient functions that are not parity functions. For these functions, our method yields a nonlinear lower bound whenever $t \sim n$, that is, whenever the resiliency is asymptotic to the number of arguments.

2. Noisy leaf complexity. Let f be a Boolean function of n arguments $x_1, \ldots, x_n$. Let T be a decision tree and let L be a leaf of T. By the *cost* of L we shall mean the number of queries along the path from the root of T to L. Suppose now that the input x is chosen at random with the uniform distribution (with each possible input having probability 2^{-n}). Suppose further that the tree T is applied to the input x with query error probability $\varepsilon > 0$ at each internal node. We shall say that L is (ε, δ)-*good* for f if the probability $\Pr(\phi(L) = \neg f(x) \mid L)$ of output error at L, conditional on control reaching L, is at most $\delta < 1/2$. It is clear that whether or not a leaf L is (ε, δ)-good for f depends only on the numbers $C_1, \ldots, C_n$ of queries to the arguments $x_1, \ldots, x_n$, and on the numbers $D_1, \ldots, D_n$ of affirmative responses to these queries, and not on the rest of T. By the (ε, δ)-*leaf complexity* of a Boolean function f, we shall mean the smallest possible cost of a leaf that is (ε, δ)-good for f.

PROPOSITION 2.1. *Suppose that the noisy dynamic decision tree T (ε, δ)-computes the Boolean function f with expected cost C averaged over both inputs and noise. Let δ' be such that $\delta < \delta' < 1/2$. Then f has (ε, δ')-leaf complexity at most $C' = C/(1 - \delta/\delta')$.*

Proof. Let the input x be chosen with the uniform distribution. For each leaf L in T, let $p_L = \Pr(L)$ denote the probability that control reaches L, let $\delta_L = \Pr(\phi(L) = \neg f(x) \mid L)$ denote the probability of error conditional on control reaching L, and let C_L denote the cost of L. Let A denote the set of leaves L such that $\delta_L > \delta'$. If A is nonempty we have

$$\delta' \sum_{L \in A} p_L < \sum_{L \in A} p_L \delta_L \leq \sum_{L} p_L \delta_L = \delta,$$

and if A is empty we have

$$\delta' \sum_{L \in A} p_L = 0 < \delta,$$

so in any case we have

$$\sum_{L \in A} p_L < \delta/\delta'.$$

Let B denote the set of leaves L such that $C_L > C'$. If B is nonempty we have

$$C' \sum_{L \in B} p_L < \sum_{L \in B} p_L C_L \leq \sum_{L} p_L C_L = C,$$

and if B is empty we have

$$C' \sum_{L \in B} p_L = 0 < C,$$

so in any case we have

$$\sum_{L \in B} p_L < C/C'.$$

These inequalities yield

$$\sum_{L \notin A \cup B} p_L > 1 - \delta/\delta' - C/C' = 0.$$

Thus with positive probability control arrives at a leaf L such that $\delta_L \leq \delta'$ and $C_L \leq C'$, which shows that the (ε, δ')-leaf complexity of f is at most C'. □

3. The posterior distribution. Suppose that we choose an input x at random with a uniform distribution: $\Pr(x) = 2^{-n}$. Then suppose that we apply a noisy dynamic decision tree T with query error probability $\varepsilon > 0$ and arrive at a leaf L. We shall calculate the posterior probability distribution on x, given arrival at L: $\Pr(x \mid L)$.

Suppose that along the path from the root of T to L the input x_i is queried C_i times, with D_i affirmative responses (and thus $C_i - D_i$ negative responses). The event of arrival at L is the conjunction of n events $L_1, \ldots, L_n$, where L_i specifies a particular sequence of responses of the C_i queries to x_i. The prior distribution of x_i is $\Pr_i(x_i) = 1/2$. The conditional probability $\Pr_i(L_i \mid x_i)$ of L_i given x_i is

$$\Pr_i(L_i \mid 0) = \varepsilon^{D_i}(1-\varepsilon)^{C_i - D_i},$$
$$\Pr_i(L_i \mid 1) = \varepsilon^{C_i - D_i}(1-\varepsilon)^{D_i},$$

and thus

$$\Pr(L_i) = \frac{\varepsilon^{D_i}(1-\varepsilon)^{C_i - D_i} + \varepsilon^{C_i - D_i}(1-\varepsilon)^{D_i}}{2}.$$

Thus the posterior distribution $\Pr_i(x_i \mid L_i)$ of x_i, conditioned on L_i, is

$$\Pr_i(0 \mid L_i) = \frac{\varepsilon^{D_i}(1-\varepsilon)^{C_i - D_i}}{\varepsilon^{D_i}(1-\varepsilon)^{C_i - D_i} + \varepsilon^{C_i - D_i}(1-\varepsilon)^{D_i}}, \tag{3.1}$$

$$\Pr_i(1 \mid L_i) = \frac{\varepsilon^{C_i - D_i}(1-\varepsilon)^{D_i}}{\varepsilon^{D_i}(1-\varepsilon)^{C_i - D_i} + \varepsilon^{C_i - D_i}(1-\varepsilon)^{D_i}}. \tag{3.2}$$

Finally, since the x_i and the responses to the queries given the x_i are all independent, we have

$$\Pr(x \mid L) = \prod_{1 \le i \le n} \Pr_i(x_i \mid L_i). \tag{3.3}$$

Formulas (3.1), (3.2), and (3.3) give the desired posterior distribution of x.

It will be convenient to have bounds for $\Pr_i(x_i \mid L_i)$ that are independent of D_i. If we divide the numerator and denominator of (3.1) by the numerator, we obtain

$$\Pr_i(0 \mid L_i) = \frac{1}{1 + E^{2D_i - C_i}},$$

where $E = (1-\varepsilon)/\varepsilon$ (and $E > 1$, since $\varepsilon < 1/2$). The right-hand side is maximized when $D_i = 0$, so we have

$$\Pr_i(0 \mid L_i) \le \frac{1}{1 + E^{-C_i}}.$$

Similar reasoning from (3.2) yields an expression that is maximized when $D_i = C_i$, resulting in the same bound for $\Pr_i(1 \mid L_i)$. Thus if we set $P_i = \max\{\Pr_i(0 \mid L_i), \Pr_i(1 \mid L_i)\}$, we have

$$\begin{aligned} P_i &\le \frac{1}{1 + E^{-C_i}} \\ &= \frac{E^{C_i}}{E^{C_i} + 1} \\ &= 1 - \frac{1}{E^{C_i} + 1} \\ &\le 1 - \frac{1}{2E^{C_i}}. \end{aligned} \tag{3.4}$$

This is the desired bound.

4. Random Boolean functions. Throughout this section, f will denote a random Boolean function of n arguments, for which $f(x)$ is equally likely to be 0 or 1, independently for each value of x. Our strategy will be to consider a leaf L of small depth and bound the probability that L is (ε,δ)-good for f. Our main result is the following.

THEOREM 4.1. *Let L be a leaf of cost*

$$C \le n\log_E(n/2) - n\log_E\log\bigl(2n^2/(1-2\delta)^2\bigr),$$

where $E = (1-\varepsilon)/\varepsilon$. Then L is (ε,δ)-good for a random Boolean function of n arguments with probability at most $2e^{-n^2}$.

This result easily yields a lower bound for the noisy leaf complexity of almost all Boolean functions.

COROLLARY 4.2. *For all sufficiently large n (depending on $E = (1-\varepsilon)/\varepsilon > 1$ and $\delta < 1/2$), the fraction of all Boolean functions of n arguments having (ε,δ)-leaf complexity at most $(n/2)\log_E(n/2)$ is at most $2e^{-n^2/2}$.*

Proof. For all sufficiently large n, we have

$$C = (n/2)\log_E(n/2) \le n\log_E(n/2) - n\log_E\log\bigl(2n^2/(1-2\delta)^2\bigr),$$

so we may apply Theorem 4.1 to any leaf of cost at most C. But such a leaf is determined by specifying (1) which of the n arguments is queried at each of the C queries and (2) the response (affirmative or negative) to each query. Thus there are at most $(2n)^C$ leaves, and thus the probability that some leaf is (ε,δ)-good for f is at most $2e^{-n^2}(2n)^{(n/2)\log_E(n/2)}$. For sufficiently large n, this bound is at most $2e^{-n^2/2}$. □

It will be convenient to work not only with the Boolean function f but also with the rescaled real-valued function $F(x) = 1 - 2f(x)$, which is equally likely to be $+1$ or -1, independently for each value of x. Similarly, it will be convenient to work not only with the probability of error δ_L associated with a leaf L but also with the *correlation* $\xi_L = 1 - 2\delta_L$ between the rescaled label $\Phi(L) = 1 - 2\phi(L)$ of L and the rescaled function $F(x)$. If $\delta_L \le \delta < 1/2$, then $\xi_L \ge 1-2\delta > 0$. Thus if L is (ε,δ)-good for f we have

$$1 - 2\delta \le \xi_L = \mathrm{Ex}_x\bigl(\Phi(L)F(x)\bigr) = \Phi(L)\sum_x \Pr(x \mid L)F(x).$$

Since $\Phi(L) = \pm 1$, this implies

$$1 - 2\delta \le \left|\sum_x \Pr(x \mid L)F(x)\right|. \tag{4.1}$$

The terms $\Pr(x \mid L)F(x)$ are independent random variables that assume the values $\pm\Pr(x \mid L)$ each with probability 1/2. Thus, to estimate the probability that (4.1) holds, it will suffice to use an estimate for the probability of large deviations for sums of independent, but not necessarily identically distributed, random variables. The following result of Hoeffding [11, Theorem 2] suits our purpose.

PROPOSITION 4.3 (see [11]). *If A_x are independent random variables with mean 0 and range $|A_x| \le \Delta_x$, then*

$$\Pr\left(\sum_x A_x \ge T\right) \le \exp(-T^2/2S),$$

where

$$S = \sum_x \Delta_x^2.$$

Since the random variables $\Pr(x \mid L)F(x)$ are distributed symmetrically about 0, the probability that (4.1) holds is just twice the probability that

$$1 - 2\delta \le \sum_x \Pr(x \mid L)F(x) \tag{4.2}$$

holds. We can bound this using Proposition 4.3 by taking $A_x = \Pr(x \mid L)F(x)$, so that $\Delta_x = \Pr(x \mid L)$, and $T = 1 - 2\delta$. Thus we seek an estimate for

$$S = \sum_x \Pr(x \mid L)^2.$$

We observe that by virtue of (3.3) we have

$$S = \sum_x \Pr(x \mid L)^2 = \prod_{1 \le i \le n} \left(\Pr_i(0 \mid L_i)^2 + \Pr_i(1 \mid L_i)^2\right).$$

Since

$$u^2 + (1-u)^2 = 1 - 2u(1-u) \le \max\{u, 1-u\},$$

we have

$$S \le \prod_{1 \le i \le n} P_i,$$

with $P_i = \max\{\Pr_i(0 \mid L_i), \Pr_i(1 \mid L_i)\}$ as defined in section 3. Using (3.4) we have

$$S \le \prod_{1 \le i \le n} \left(1 - \frac{1}{2E^{C_i}}\right).$$

Since $1 - u \le \exp(-u)$, we have

$$\begin{aligned} S &\le \prod_{1 \le i \le n} \exp\left(-\frac{1}{2}E^{-C_i}\right) \\ &= \exp\left(-\frac{1}{2}\sum_{1 \le i \le n} E^{-C_i}\right). \end{aligned}$$

Since $E^u = \exp_E u$ is a convex function of u, we have

$$\begin{aligned} S &\le \exp\left(-\frac{n}{2}\exp_E\left(-\frac{1}{n}\sum_{1 \le i \le n} C_i\right)\right) \\ &= \exp\left(-\exp_E\left(\log_E \frac{n}{2} - \frac{C}{n}\right)\right). \end{aligned}$$

Thus if

$$C \le n\log_E(n/2) - n\log_E\log\left(2n^2/(1-2\delta)^2\right),$$

we have $S \le (1-2\delta)^2/2n^2$. Proposition 4.3 then implies that (4.2) holds with probability at most e^{-n^2}, so (4.1) holds with probability at most $2e^{-n^2}$. This completes the proof of Theorem 4.1.

5. The parity function. In this section we shall derive a lower bound for the (ε, δ)-leaf complexity of the parity function:

$$f(x_1, \ldots, x_n) = x_1 + \cdots + x_n \pmod 2.$$

Our result is the following.

THEOREM 5.1. *If the leaf L with cost C is (ε, δ)-good for the parity function f of n arguments, then*

$$C \geq n \log_E n - n \log_E \log\bigl(1/(1-2\delta)\bigr),$$

where $E = (1-\varepsilon)/\varepsilon$.

The proof of this theorem depends on the notion of the Fourier transform of a Boolean function. This notion has already been applied to the computational complexity of Boolean functions by circuits (see Linial, Mansour, and Nisan [12]) and noiseless dynamic decision trees (see Brandman, Orlitsky, and Hennessy [1]), but the present paper appears to mark its debut for the complexity of noisy computation.

Let $F : \mathbf{B}^n \to \mathbf{R}$ be a real-valued function of n Boolean arguments. By the *Fourier transform* of F we shall mean the function $\hat{F} : \mathbf{B}^n \to \mathbf{R}$ defined by

$$\hat{F}(y) = \frac{1}{\sqrt{2^n}} \sum_x (-1)^{x \cdot y} F(x),$$

where $x \cdot y = \sum_{1 \leq j \leq n} x_j y_j$ denotes the inner product of x and y. (The factor $(-1)^{x \cdot y}$ is the specialization of the usual Fourier kernel $e^{2\pi i x \cdot y/m}$ to $m = 2$.) The normalization factor $1/\sqrt{2^n}$ has been chosen to make the transform an involution: we have

$$\begin{aligned}
\hat{\hat{F}}(z) &= \frac{1}{\sqrt{2^n}} \sum_y (-1)^{y \cdot z} \hat{F}(y) \\
&= \frac{1}{\sqrt{2^n}} \sum_y (-1)^{y \cdot z} \frac{1}{\sqrt{2^n}} \sum_x (-1)^{x \cdot y} F(x) \\
&= \frac{1}{2^n} \sum_x F(x) \sum_y (-1)^{x \cdot y + y \cdot z} \\
&= F(z),
\end{aligned}$$

since

$$\sum_y (-1)^{x \cdot y + y \cdot z} = \begin{cases} 2^n & \text{if } x = z, \\ 0 & \text{otherwise.} \end{cases}$$

(The general Fourier transform is not an involution, but rather has period four, and the effect of applying the transform twice is to reverse the function by negating its argument. But in $\mathbf{B}$, regarded as an additive group of order two, every element is its own negative, so each function is its own reversal.)

The key result we shall need is the Parseval identity

$$\sum_y \hat{F}(y)\, \hat{G}(y) = \sum_y F(y)\, G(y),$$

which says that the Fourier transform is an isometry of the Hilbert space $\mathbf{R}^{\mathbf{B}^n}$. This follows from a calculation similar to the one above:

$$\begin{aligned}\sum_y \hat{F}(y)\,\hat{G}(y) &= \sum_y \frac{1}{\sqrt{2^n}} \sum_x (-1)^{x\cdot y} F(x) \frac{1}{\sqrt{2^n}} \sum_z (-1)^{z\cdot y} G(z) \\ &= \frac{1}{2^n} \sum_x \sum_z F(x)\,G(z) \sum_y (-1)^{x\cdot y + z\cdot y} \\ &= \sum_x F(x)\,G(x).\end{aligned}$$

For the proof of Theorem 5.1, we take $F(x) = 1 - 2f(x)$ to be the rescaled parity function. As in the preceding section, we have

$$1 - 2\delta \le \left| \sum_x \Pr(x \mid L) F(x) \right|.$$

Setting $G(x) = \Pr(x \mid L)$ and applying the Parseval identity, we have

$$1 - 2\delta \le \left| \sum_y \hat{G}(y) \hat{F}(y) \right|. \tag{5.1}$$

For F the rescaled parity function, a simple calculation yields $\hat{F}$:

$$\hat{F}(y) = \begin{cases} \sqrt{2^n} & \text{if } y = (1, \ldots, 1), \\ 0 & \text{otherwise.} \end{cases}$$

Substituting this formula into (5.1) yields

$$1 - 2\delta \le \sqrt{2^n}\ \ |\hat{G}(1, \ldots, 1)|.$$

From the definitions of G and $\hat{G}$, this reduces to

$$1 - 2\delta \le \left| \sum_x (-1)^{|x|} \Pr(x \mid L) \right|, \tag{5.2}$$

where $|y| = \sum_{1 \le i \le n} y_i$ denotes the number of i such that $y_i = 1$.

To estimate the right-hand side of (5.2), we observe that

$$\begin{aligned}\sum_x (-1)^{|x|} \Pr(x \mid L) &= \prod_{1 \le i \le n} \big(\Pr_i(0 \mid L_i) - \Pr_i(1 \mid L_i) \big) \\ &= \prod_{1 \le i \le n} \big(1 - 2\Pr_i(1 \mid L_i) \big).\end{aligned}$$

Since

$$|1 - 2u| = 2 \max\{u, 1 - u\} - 1,$$

we have

$$\begin{aligned}1 - 2\delta &\le \left| \sum_x (-1)^{|x|} \Pr(x \mid L) \right| \\ &= \prod_{1 \le i \le n} (2P_i - 1),\end{aligned}$$

with $P_i = \max\{\Pr_i(0 \mid L_i), \Pr_i(1 \mid L_i)\}$ as defined in section 3. Using (3.4) we have

$$1 - 2\delta \leq \prod_{1 \leq i \leq n} \left(1 - \frac{1}{E^{C_i}}\right).$$

Since $1 - u \leq \exp -u$, we have

$$\begin{aligned} 1 - 2\delta &\leq \prod_{1 \leq i \leq n} \exp\left(-E^{-C_i}\right) \\ &= \exp\left(-\sum_{1 \leq i \leq n} E^{-C_i}\right). \end{aligned}$$

Since $E^u = \exp_E u$ is a convex function of u, we have

$$\begin{aligned} 1 - 2\delta &\leq \exp\left(-n \exp_E\left(-\frac{1}{n} \sum_{1 \leq i \leq n} C_i\right)\right) \\ &= \exp\left(-\exp_E\left(\log_E n - \frac{C}{n}\right)\right). \end{aligned}$$

Thus we obtain

$$C \geq n \log_E n - n \log_E \log\bigl(1/(1 - 2\delta)\bigr).$$

This completes the proof of Theorem 5.1.

6. Resilient Boolean functions. A Boolean function f of n arguments is *unbiased* if

$$\sum_x F(x) = 0,$$

where $F(x) = 1 - 2f(x)$ is the rescaled real-valued function as in the preceding section, and the sum is over all 2^n values of x. Thus a function is unbiased if it assumes the values 0 and 1 for equal numbers of inputs.

A Boolean function f is *t-resilient* if every function obtained from f by substituting constants for at most t arguments is an unbiased function of the remaining arguments. Thus a function is 0-resilient if and only if it is unbiased. Our main result in this section is the following.

THEOREM 6.1. *If f is t-resilient and the leaf L with cost C is (ε, δ)-good for f, then*

$$C \geq (t+1) \log_E \frac{t+1}{\frac{n}{2} H\left(\frac{t+1}{n}\right) + \log \frac{1}{1-2\delta}},$$

where $E = (1 - \varepsilon)/\varepsilon$, and $H(\eta) = -\eta \log \eta - (1 - \eta) \log(1 - \eta)$ for $0 < \eta < 1$, extended by continuity to $H(0) = H(1) = 0$.

The projection functions, of the form $f(x_1, \ldots, x_n) = x_i$, are 0-resilient but not 1-resilient. The parity functions, of the form $f(x_1, \ldots, x_n) = x_1 + \cdots + x_n + c \pmod 2$, are $(n-1)$-resilient, which is the maximum possible for a function of n arguments.

Theorem 5.1 applies to many other functions however. If g and h are t-resilient functions of k arguments, then

$$f(x_1,\dots,x_{k+1}) = \begin{cases} g(x_1,\dots,x_k) & \text{if } x_{k+1}=0,\\ h(x_1,\dots,x_k) & \text{if } x_{k+1}=1, \end{cases}$$

defines a t-resilient function of $k+1$ arguments. Since there are two distinct t-resilient parity functions of $t+1$ arguments, and this scheme allows us to square the number of functions by adding one argument, we conclude that there are at least $2^{2^{n-t-1}}$ t-resilient functions of n arguments.

Our proof of Theorem 5.1 will exploit a characterization of resilient functions in terms of their Fourier transforms. Friedman [8] has observed that this characterization is implicit in the work of Chor et al. [2], although the terminology of Fourier transforms is not used there.

PROPOSITION 6.2 (see [2]). *Let $\hat{F}$ be the Fourier transform of $F(x) = 1 - 2f(x)$ for some Boolean function f of n arguments. Then for $t \geq 0$, f is t-resilient if and only if $\hat{F}(y) = 0$ for all y such that $|y| \leq t$.*

In particular, a function f is unbiased if and only if $\hat{F}(0,\dots,0) = 0$, and the parity functions are the only functions for which $\hat{F}(y) = 0$ for all y except $y = (1,\dots,1)$.

We shall also need the following standard estimate for sums of binomial coefficients.

LEMMA 6.3. *If $l \geq n/2$, then*

$$\sum_{k\geq l} \binom{n}{k} \leq \exp\bigl(nH\bigl(l/n\bigr)\bigr).$$

Proof. For $\xi \geq 1$ we have

$$\sum_{k\geq l} \binom{n}{k} \leq \xi^{-l} \sum_{k} \binom{n}{k} \xi^k = \xi^{-l}(1+\xi)^n.$$

Taking $\xi = l/(n-l)$, so that $\xi \geq 1$ follows from $l \geq n/2$, we obtain

$$\sum_{k\geq l} \binom{n}{k} \leq \frac{n^n}{l^l (n-l)^{n-l}} = \exp\bigl(nH(l/n)\bigr),$$

as claimed. □

As in the preceding, section we have

$$1 - 2\delta \leq \left| \sum_y \hat{G}(y)\hat{F}(y) \right|, \tag{6.1}$$

where $\hat{G}$ is the Fourier transform of $G(x) = \Pr(x \mid L)$. Since f is t-resilient, we have $\hat{F}(y) = 0$ for $|y| \leq t$, and thus we have

$$1 - 2\delta \leq \sum_{\substack{y\\ |y|\geq t+1}} |\hat{G}(y)||\hat{F}(y)|.$$

Using Cauchy's inequality we obtain

$$(1-2\delta)^2 \leq \left(\sum_{\substack{y\\ |y|\geq t+1}} \hat{G}(y)^2 \right) \left(\sum_{\substack{y\\ |y|\geq t+1}} \hat{F}(y)^2 \right). \tag{6.2}$$

Since $F(x) = \pm 1$, Parseval's identity yields

$$\sum_{\substack{y \\ |y| \geq t+1}} \hat{F}(y)^2 \leq \sum_y \hat{F}(y)^2 = \sum_x F(x)^2 = 2^n.$$

Thus from (6.2) we obtain

$$(1-2\delta)^2 \leq \left(\sum_{\substack{y \\ |y| \geq t+1}} \hat{G}(y)^2 \right) 2^n. \tag{6.3}$$

We have

$$\begin{aligned} \sum_{\substack{y \\ |y| \geq t+1}} \hat{G}(y)^2 &\leq \left(\max_{\substack{y \\ |y| \geq t+1}} \hat{G}(y)^2 \right) \left(\sum_{\substack{y \\ |y| \geq t+1}} 1 \right) \\ &\leq \left(\max_{\substack{y \\ |y| \geq t+1}} \hat{G}(y)^2 \right) \left(\sum_{k \geq t+1} \binom{n}{k} \right) \\ &\leq \left(\max_{\substack{y \\ |y| \geq t+1}} \hat{G}(y)^2 \right) \exp\left(nH\left(\frac{t+1}{n} \right) \right). \end{aligned}$$

Thus from (6.3) we obtain

$$(1-2\delta)^2 \leq \left(\max_{\substack{y \\ |y| \geq t+1}} \hat{G}(y)^2 \right) \exp\left(nH\left(\frac{t+1}{n} \right) \right) 2^n.$$

We have

$$\hat{G}(y)^2 = \frac{1}{2^n} \left(\sum_x (-1)^{x \cdot y} \Pr(x \mid L) \right)^2.$$

The sum on the right-hand side can be estimated in the same way as the sum in (5.2): if $|y| = k \geq t+1$, the sum factors into a product of k factors, and the final result is

$$\hat{G}(y)^2 \leq \frac{1}{2^n} \exp\left(-2 \exp_E\left(\log_E k - \frac{C}{k} \right) \right),$$

so that

$$\max_{\substack{y \\ |y| \geq t+1}} \hat{G}(y)^2 \leq \frac{1}{2^n} \exp\left(-2 \exp_E\left(\log_E(t+1) - \frac{C}{t+1} \right) \right).$$

Thus from (6.3) we obtain

$$(1-2\delta)^2 \leq \exp\left(-2 \exp_E\left(\log_E(t+1) - \frac{C}{t+1} \right) \right) \exp\left(nH\left(\frac{t+1}{n} \right) \right).$$

This yields

$$C \geq (t+1) \log_E \frac{t+1}{\frac{n}{2} H\left(\frac{t+1}{n} \right) + \log \frac{1}{1-2\delta}},$$

which completes the proof of Theorem 6.1.

REFERENCES

[1] Y. Brandman, A. Orlitsky, and J. Hennessy, *A spectral lower bound technique for the size of decision trees and two-level AND/OR circuits*, IEEE Trans. Comput., 39 (1990), pp. 282–287.

[2] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolensky, *The bit extraction problem or t-resilient functions*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1985, pp. 396–407.

[3] R. L. Dobrushin and S. I. Ortyukov, *Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements*, Problems Inform. Transmission, 13 (1977), pp. 59–65.

[4] R. L. Dobrushin and S. I. Ortyukov, *Upper bound for the redundancy of self-correcting arrangements of unreliable functional elements*, Problems Inform. Transmission, 13 (1977), pp. 203–218.

[5] W. Evans and N. Pippenger, *Lower bounds for noisy Boolean decision trees*, in Proc. 28th Annual ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 620–628.

[6] U. Feige, D. Peleg, P. Raghavan, and E. Upfal, *Computing with unreliable information*, in Proc. 22nd Annual ACM Symposium on Theory of Computing, ACM, New York, 1990, pp. 128–137.

[7] U. Feige, P. Raghavan, D. Peleg, and E. Upfal, *Computing with noisy information*, SIAM J. Comput., 23 (1994), pp. 1001–1018.

[8] J. Friedman, *On the bit extraction problem*, in Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 314–319.

[9] A. Gál, *Lower bounds for the complexity of reliable Boolean circuits with noisy gates*, in Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 594–601.

[10] P. Gács and A. Gál, *Lower bounds for the complexity of reliable Boolean circuits with noisy gates*, IEEE Trans. Inform. Theory, 40 (1994), pp. 579–583.

[11] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.

[12] N. Linial, Y. Mansour, and N. Nisan, *Constant depth circuits, Fourier transform, and learnability*, J. Assoc. Comput. Mach., 40 (1993), pp. 607–620.

[13] D. E. Muller, *Complexity in electronic switching circuits*, Institute of Radio Engineers Trans. Elec. Comput., 5 (1956), pp. 15–19.

[14] J. von Neumann, *Probabilistic logics and the synthesis of reliable organisms from unreliable components*, in Automata Studies, C. E. Shannon and J. McCarthy, eds., Princeton University Press, Princeton, NJ, 1956, pp. 43–98.

[15] N. Pippenger, *On networks of noisy gates*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1985, pp. 30–36.

[16] N. Pippenger, G. D. Stamoulis, and J. N. Tsitsiklis, *On a lower bound for the redundancy of reliable networks with noisy gates*, IEEE Trans. Inform. Theory, 37 (1991), pp. 639–643.

[17] R. Reischuk and B. Schmeltz, *Reliable computation with noisy circuits and decision trees—A general $n \log n$ lower bound*, in Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 602–611.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 447–462

COMPETITIVE ALGORITHMS FOR LAYERED GRAPH TRAVERSAL*

AMOS FIAT†, DEAN P. FOSTER‡, HOWARD KARLOFF§, YUVAL RABANI¶, YIFTACH RAVID‖, AND SUNDAR VISHWANATHAN**

Abstract. A layered graph is a connected graph whose vertices are partitioned into sets $L_0 = \{s\}, L_1, L_2, ...$, and whose edges, which have nonnegative integral weights, run between consecutive layers. Its width is $\max\{|L_i|\}$. In the on-line layered graph traversal problem, a searcher starts at s in a layered graph of unknown width and tries to reach a target vertex t; however, the vertices in layer i and the edges between layers $i-1$ and i are only revealed when the searcher reaches layer $i-1$.

We give upper and lower bounds on the competitive ratio of layered graph traversal algorithms. We give a deterministic on-line algorithm which is $O(9^w)$-competitive on width-w graphs and prove that for no w can a deterministic on-line algorithm have a competitive ratio better than 2^{w-2} on width-w graphs. We prove that for all w, $w/2$ is a lower bound on the competitive ratio of any randomized on-line layered graph traversal algorithm. For traversing layered graphs consisting of w disjoint paths tied together at a common source, we give a randomized on-line algorithm with a competitive ratio of $O(\log w)$ and prove that this is optimal up to a constant factor.

Key words. competitive analysis, layered graphs, search strategies

AMS subject classifications. 68Q10, 68Q25

PII. S0097539795279943

1. Introduction.

Finding the shortest path in a graph from a source to a target is a well-studied problem. Dijkstra's algorithm [Dij] appeared in 1959. Other algorithms can be found in [Bel, Flo, FF, AMOT].

Baeza-Yates, Culberson, and Rawlins [BCR] and Papadimitriou and Yannakakis [PY] consider a large family of shortest path problems that operate with incomplete information. They describe algorithms that start at a source, search for the target, and learn about the environment as they progress. The complexity measure associated with such an algorithm is the ratio of the total distance traversed by the algorithm to the length of the shortest source–target path. Related work on exploring graphs with incomplete information is considered in [DP].

*Received by the editors January 13, 1995; accepted for publication (in revised form) December 23, 1996; published electronically July 28, 1998.
http://www.siam.org/journals/sicomp/28-2/27994.html

†Computer Science Department, School of Mathematics, Tel-Aviv University, Tel-Aviv 69978, Israel (fiat@math.tau.ac.il).

‡Department of Statistics, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104-6302 (foster@hellspark.wharton.upenn.edu). This work was done while the author was at the Graduate School of Business, University of Chicago.

§College of Computing, Georgia Tech, Atlanta, GA 30332-0280 (howard@cc.gatech.edu). This author was supported in part by NSF grant CCR-8807534. This work was done while the author was at the Department of Computer Science, University of Chicago.

¶Computer Science Department, The Technion, Haifa 32000, Israel (rabani@cs.technion.ac.il). This work was done while the author was a graduate student at the Computer Science Department, Tel Aviv University.

‖IBM Haifa Research Laboratory–Tel Aviv Annex, IBM House, 2 Weizmann St., Tel Aviv 61336, Israel (yiftach@haifa.vnet.ibm.com). This work was done while the author was a graduate student at the Computer Science Department, Tel Aviv University.

**Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, India 400076 (sundar@cse.iitb.ernet.in). This author was supported in part by NSF grants CCR-8710078 and CCR-8906799. This work was done while the author was a graduate student at the Department of Computer Science, University of Chicago.

This measure is closely related to the concept of *competitive analysis*, introduced by Sleator and Tarjan [ST], which gives a worst case complexity measure for on-line algorithms. An *on-line algorithm* is an algorithm which must deal with a sequence of events, responding to events in real time without knowing what the future holds. The *competitive ratio* of an on-line algorithm A is defined as the supremum, over all sequences of events σ, and all possible (on- or off-line) algorithms ADV, of the ratio between the cost associated with A to deal with σ and the cost associated with ADV to deal with σ. We say that A is c-competitive if this supremum is at most c. (In some of the on-line literature, especially that dealing with paging and the k-server problem, from the cost of A on σ a constant additive term is subtracted, before dividing by the cost of ADV on σ. Where ambiguity might arise, we shall say that A is *strictly* c-competitive, meaning that the definition without an additive term is used.)

The *layered graph traversal problem* was introduced in [PY] and generalizes work of [BCR]. A *layered graph* is a connected graph in which the vertices are partitioned into sets $L_0 = \{s\}, L_1, L_2, L_3, ...$ and all edges run between L_{i-1} and L_i for some i. Each edge has a nonnegative integral weight. Vertex s is known as the *source*. Let $w = \max\{|L_i|\}$; w is called the *width* of the graph. An on-line layered graph traversal algorithm starts at the source and, without knowing w, moves along the edges of the graph, paying a cost equal to the weight of the edge traversed. Its goal is to reach the vertex t in the last layer known as the "target"; which vertex is the target is not revealed until the searcher occupies a vertex in the last layer.

Edges can be traversed in either direction, but the on-line algorithm pays whenever it crosses the edge. The edges between L_{i-1} and L_i, and their lengths, become known only when a node in L_{i-1} is reached.

We define the competitive ratio of a layered graph traversal algorithm to be the worst case ratio between the total distance traveled by the on-line algorithm and the length of the shortest source–target path. (If the algorithm is randomized, we use the expected distance it travels.) The competitive ratio of a layered graph traversal algorithm is given as a function of the width w.

A layered graph is said to consist of w *disjoint paths* if it is formed from w paths which are vertex disjoint except that each contains the common source. [BCR] gave optimal deterministic algorithms for all w with a competitive ratio which is asymptotic to $2ew$.

For arbitrary layered graphs, [PY] gave an optimal algorithm for width 2, with a competitive ratio of 9. It follows from [BCR] that $1 + 2w(1 + \frac{1}{w-1})^{w-1} \sim 2ew$ is a lower bound on the competitive ratio. Prior to this paper no other bounds were known.

Section 2 proves that general layered graphs of width w weighted with arbitrary nonnegative integers are no more difficult to traverse than width-w layered *trees* whose weights are $0-1$. Notice that if we know a lower bound on the smallest nonzero weight of an edge, then we can express the weights as multiples of this lower bound and round to the closest integer, thereby converting the problem with arbitrary nonnegative weights to one with integer weights. The competitive ratio is affected by at most a constant factor due to this conversion. This factor can be made arbitrarily close to one by taking the lower bound arbitrarily close to zero.

In sections 3 and 4 we give upper and lower bounds, exponential in w, on the competitive ratio for deterministic layered graph traversal.

- Section 3 gives an algorithm which attains a competitive ratio of $O(9^w)$ on layered graphs of width w. This algorithm does not need to know w in

advance and automatically adjusts itself to deal with the real width on hand.
- Section 4 proves that for all w, 2^{w-2} is a lower bound on the competitive ratio of any deterministic on-line layered graph traversal algorithm.

Thus arbitrary layered graphs are much harder to traverse than those consisting of disjoint paths.

Randomized on-line algorithms are addressed in several papers including [BLS, RS, CDRS, FKLMSY, BBKTW, KRR]. An oblivious adversary is one who constructs the sequence of events in advance and deals with the sequence optimally. For this adversary model [BLS] and [FKLMSY] give examples where randomization can improve the competitive ratio exponentially. This adversary models a world in which the on-line algorithm's actions do not themselves influence future events. One can consider a situation where the on-line algorithm's actions have a direct influence on the future. In such cases [BBKTW] showed that randomization cannot improve the competitive ratio more than polynomially. We deal with randomized layered graph traversal algorithms (assuming an oblivious adversary), and present the following results.

- Section 5 gives a randomized on-line algorithm for the disjoint path traversal problem. The competitive ratio is $O(\log w)$. We also show that this is optimal up to a constant factor. This is an exponential improvement over the bound for deterministic algorithms. This result immediately gives a randomized min operator [FRR] for on-line k-server algorithms: given a set of w possibly conflicting on-line strategies, a new on-line strategy can be devised which is no worse than $O(\log w)$ times the best of these strategies on every input.
- Section 6 gives a lower bound of $w/2$ on the competitive ratio of any randomized traversal algorithm for general layered graphs.

The problem of traversing layered graphs generalizes numerous on-line problems. For instance, metrical task systems (see [BLS]) can be modeled as layered graphs where layers represent tasks, and in each layer there is a node for each possible state. The k-server problem (see [MMS]), viewed in the servers' configuration space, is the problem of traversing the layered graph of permitted configurations for each request. Unfortunately, the width of this graph depends on the cardinality of the metric space, and not just on the number of servers, so layered graph techniques are inadequate for producing solutions to the k-server problem directly. However, the algorithm given in [BCR] for traversing layered graphs consisting of disjoint paths was used by [FRR] in their construction of competitive k-server algorithms.

As an additional example of the power of layered graph traversal as a tool for designing on-line algorithms, consider the problem of metrical service systems, suggested by [CL]. A single server moving among points of a metric space is presented with requests. Each request is a set of at most w points. One of these points is then selected by the on-line algorithm, and the server is moved to that point; the cost is the distance moved. [CL] gave a competitive metrical service system algorithm for uniform metric spaces and deterministic and randomized algorithms for all metric spaces for the case of $w = 2$. Note that the k-server problem can be reduced to the metrical service systems problem in the configuration space. Section 7 shows that the metrical service systems problem with requests of size w (in metric spaces with integral distances) is equivalent to the width-w layered graph traversal problem, when w is known in advance, in that a c_w-competitive algorithm exists for one problem if and only if one exists for the other. Related recent work appears in [FL].

2. Trees are sufficient. We first prove that given a competitive on-line algorithm for traversing width-w layered trees, in which each edge has a $0-1$ weight and each nonsource vertex has a neighbor in the previous layer, one can construct an on-line algorithm, with the same competitive ratio, for traversing arbitrary width-w layered graphs.

DEFINITION 1. *Let H be any layered graph with source s, and let v be a vertex in H in, say, layer L_j. Define H_v to be a shortest $s-v$ path in H which contains no vertex of $L_{j+1} \cup L_{j+2} \cup L_{j+3} \cup \cdots$ (if such a path exists).*

Let G be a layered graph of width at most w with nonnegative integral edge weights and with source s. We start by proving that an on-line algorithm traversing G can construct, on the fly, a layered tree T with the following properties.

1. A vertex v is in T's ith layer if and only if v is in G's ith layer and G_v exists.
2. For all v, the length of T_v is at most the length of G_v (if G_v exists).
3. Each nonsource vertex in T has exactly one neighbor in the previous layer. (We call such a tree *rooted*.)

Furthermore, any on-line traversal algorithm for T can be simulated on G without increasing the cost.

The tree $T = T(G)$ is defined by induction on the layer index i, starting from a one-node graph ($i = 0$). Let $i > 0$. For every v in G's ith layer L_i for which G_v exists, one vertex and one edge are added to T as follows. Let $u_0 = s$ and let $G_v = \langle u_0, u_1, u_2, \ldots, u_\ell, v\rangle$. Let u_k be the first vertex in G_v which is in layer L_{i-1}. Add to T vertex v and edge (u_k, v) with weight equal to the weight of the portion of G_v between u_k and v.

LEMMA 2. *For all v, the length of T_v is at most the length of G_v.*

Proof. The proof by induction on the index of the layer containing v.

Basis: $i = 0$. Trivial.

Inductive Step: $i > 0$. Assume correctness for $i-1$. Suppose that v is adjacent in T to u_k in T's $i-1$st layer. In path G_v, let a be the length of the prefix from s to u_k and let b be the length of the $u_k - v$ suffix. The length of T_v equals the length of T_{u_k} plus b. By the inductive hypothesis, the length of path T_{u_k} is at most the length of G_{u_k}, which is itself at most a. Therefore, the length of T_v is at most $a + b$, the length of G_v. ∎

Given an algorithm $\mathcal{A}$ to traverse T, we show how to traverse G without increasing the cost. Suppose that $\mathcal{A}$ moves in T from u in layer $i-1$ to v in layer i. The weight of the edge traversed in T is the length of a portion of G_v in G. This portion avoids layers $i+1, i+2, ...$, so the G-traversal algorithm can follow it. Similarly, if $\mathcal{A}$ moves from v in layer i to u in layer $i-1$, the G-traversal algorithm can traverse backward the corresponding portion of G_v.

A layered tree with arbitrary nonnegative integral weights can be converted to a layered tree with $0-1$ weights by inserting additional intermediate layers, on the fly.

3. A deterministic algorithm. Without loss of generality, we may assume that the original problem asks for a traversal algorithm for $0-1$, rooted, layered trees of arbitrary width, each having a target. Instead, for each w we will build a traversal algorithm A_w that maintains the following property. For each $0-1$ rooted tree T of width at most w without a target, for each i, the cost incurred by A_w on T between the start and the time it visits its first layer-i vertex is at most $8 \cdot 9^w$ times the length of a shortest path between s and any vertex of L_i.

We can easily solve the original problem via algorithms $A_1, A_2, ...$ We need only run A_j, starting with $j = 1$, until the width exceeds j, or until we reach some vertex in the same layer as the target. If, including the newly revealed layer, the width is $k > j$, we backtrack to the source and execute procedure A_k, starting at the source, forgetting everything we know about the graph. As soon as we learn that the layer we occupy contains the target, we backtrack to the source and then travel optimally to t. The total cost incurred by this algorithm on a width-w graph whose shortest source−target path is of length d is bounded by

$$d[8 \cdot 9^1 + 8 \cdot 9^2 + \cdots + 8 \cdot 9^w + (8 \cdot 9^w + 1)].$$

This is $O(9^w)$ times the source−target distance.

In order to define algorithms A_w, we need some terminology.

1. We refer to the time just after layer t and the edges from layer $t-1$ to t have been revealed as *time* t. The algorithm must move to a vertex in layer t after time t and before time $t+1$.

2. Vertex v is *active* at time t if it has a descendant in layer t. At time t, vertices in layer t are called *active leaves*.

3. At time t, $SP(v)$ denotes the length of the shortest path from v to a descendant of v in layer t (if v is active at time t).

Now we construct the algorithms. A_1 is the obvious algorithm. A_w for $w > 1$ is constructed from $A_1, A_2, A_3, ..., A_{w-1}$ as follows. Its execution is divided into phases. Within each phase, a vertex r, initially the source, is designated as the root for that entire phase. We will maintain the invariant that every path from the source to an active leaf passes through the root r. The searcher occupies r at the start of the phase. Furthermore, an integer d is fixed for the entire duration of the phase.

To start a phase, we let $d = SP(r)$. If $d = 0$, the searcher moves along length-0 edges from r, visiting all descendents of r at distance 0 from r (using, say DFS), then returning back to r, all at no cost.

At this point, $d = SP(r) \geq 1$ is fixed for the phase, and the searcher occupies r. If y is a descendant of x, let $d(x, y)$ denote the length of the unique $x - y$ path. At all times, let $S = \{s | s$ is an active descendant of r, $d(r, s) = d$, s's parent u satisfies $d(r, u) = d - 1$, and $SP(s) < d\}$. (A function of time, S may change many times within a phase to reflect its definition; however, d is defined once at the beginning of a phase and remains constant.) Because some active leaf is at distance exactly d from r at the start of a phase, $S \neq \emptyset$ at that time. Because the active leaf descendants of different $s \in S$ are distinct, $|S| \leq w$ always.

Let S_t denote the set S at time t. A phase ends as soon as either (1) there is an $x \in S_t$ such that at time t, x has w active leaf descendants, or (2) $S_t = \emptyset$. If either (1) or (2) occurs, the current phase ends at time $t - 1$, and a new phase, possibly with a new root, begins immediately afterward.

Each phase is divided into subphases. The start of a phase marks the beginning of its first subphase. A new subphase begins at a later time t if S_t is strictly smaller than S_{t-1}. (A phase may end in the middle of a subphase.) At the start of a subphase the searcher occupies the root r. He chooses an arbitrary $s \in S$ and at a cost of d moves from r to s. Where $z = |S|$, if $z = 1$, then the searcher executes procedure A_{w-1} with s as the root, and if $z \geq 2$, he executes procedure $A_{w-(z-1)}$ with s as the root.

When the subphase terminates, the searcher retraces all of his steps within that subphase back to r. This ensures that the searcher occupies r at the beginning of the next subphase.

If a phase terminates because of termination condition (1), i.e., there is an $x \in S_t$ such that the tree rooted at x has w active leaves, then $S_t = \{x\}$. In this case the searcher moves from r to x, a distance of d, and makes x the root for the next phase. If a phase terminates because of termination condition (2), i.e., $S_t = \emptyset$, the root remains the same vertex r. Notice that in this case, $SP(r)$ increased during the phase by at least d, so the next phase will begin with the new d at least double its value in the previous phase. This concludes the definition of A_w.

Analysis.

We state four easily proven facts.

FACT 3. *If $z = |S|$ at the beginning of a subphase which starts at s, then throughout that subphase the width of the subtree rooted at s is at most $w - (z-1)$.*

Proof. At any time during the subphase, each vertex in S has at least one active leaf as a descendant. Since $|S - \{s\}|$ equals $z-1$ during the subphase, s can have at most $w-(z-1)$ active leaf descendants at any time, and therefore the width of the subtree rooted at s cannot exceed $w-(z-1)$. ∎

FACT 4. *Within one phase, algorithm A_{w-1} is executed at most twice. For $i < w-1$, A_i is executed at most once within a phase. An invocation of A_i $(1 \le i \le w-1)$ starting at vertex s terminates with $SP(s) \le d$.*

Proof. For a given z, only one recursive call is made while $|S| = z$. For $z \le 2$, A_w calls A_{w-1}. A_i for $i < w-1$ can be called by A_w only if $z = w - i + 1$. As soon as $SP(s) \ge d$, s is evicted from S and the subphase terminates (if not before). ∎

FACT 5. *If a phase ends because of phase termination condition (1), i.e., there is an $x \in S$ such that the tree rooted at x has w active leaves, then the new root x satisfies $d(source, x) = d(source, r) + d$, and, at the phase end, every source$-$active leaf path passes through x.*

Proof. Since $x \in S$ implies that x is a descendant of r satisfying $d(r,x) = d$, clearly $d(source, x) = d(source, r) + d$. And if the tree rooted at x has w active leaves when a phase ends, the width bound of w implies that from that time onward every source–leaf path contains x. ∎

FACT 6. *If condition (2) triggers the end of a phase, then the length of a shortest path from the source to an active leaf is at least d greater at the end of the phase than at the end of the previous phase.*

Proof. When the phase starts, $SP(r) = d$. If $S = \emptyset$ at the phase end, then every vertex originally in S has been evicted from S. All vertices in S at the beginning of the phase evicted by reason of inactivity are inactive at the end of the phase.

If y is any active leaf at the phase end, on the $r - y$ path there must be a vertex x closest to r such that $d(r,x) = d$. The only possible reason why this active vertex is not in S at the end of the phase is that $SP(x) \ge d$ at the end. Therefore, $d(r,y) = d(r,x) + d(x,y) \ge d + d = 2d$ and $SP(r) \ge 2d$ at the end of the phase. ∎

THEOREM 7. *For each w, for each rooted $0-1$ tree T of width at most w, the cost incurred by A_w on T is at most $8 \cdot 9^w$ times the length of a shortest path from the source to a vertex in the highest-numbered layer.*

Proof. We prove the statement by induction on w. For $w = 1$ the statement is clear.

Let $w > 1$. At the start of a phase rooted at, say, r, the searcher occupies r. It incurs no cost until every path from r to an active leaf has positive cost. Moving from r to the designated s costs d. Within a subphase, let z denote $|S|$ at the beginning of the subphase. If $z \ge 2$, algorithm $A_{w-(z-1)}$ is invoked, and by Fact 3 the width of the tree on which $A_{w-(z-1)}$ is invoked does not exceed $w-(z-1)$ during the subphase. A_{w-1}

is invoked if $z = 1$, but the width cannot exceed $w - 1$ during the subphase—for if it did, the tree rooted at s would have w active leaves and phase termination condition (1) would hold, thereby aborting the current phase (and subphase). Furthermore, within a subphase which starts at s, $SP(s)$ cannot exceed $d - 1$. If it did, s would be evicted from S.

By the inductive hypothesis, if $z > 1$ at the start of the subphase, the cost incurred during this subphase is bounded by d (the cost of moving from r to s), plus $8 \cdot 9^{w-(z-1)}d$, plus the cost of backtracking to s and then to r, a total of at most $d + 2(8 \cdot 9^{w-(z-1)}d) + d$. If instead $z = 1$, the cost is at most $2d + 16 \cdot 9^{w-1}d$. There is an additional cost of d at the end of a phase if we move the root forward.

By Fact 4, the total cost in a phase is at most

$$d + \sum_{z=2}^{w}(2d + 16 \cdot 9^{w-(z-1)}d) + (2d + 16 \cdot 9^{w-1}d)$$

$$= (2w + 1)d + 16d[(9 + 9^2 + 9^3 + \cdots + 9^{w-2} + 9^{w-1}) + 9^{w-1}]$$

$$= (2w + 1)d + 16d\left[\frac{9^w - 9}{8} + 9^{w-1}\right]$$

$$< 2wd + 16d\left[\frac{17}{72} \cdot 9^w\right]$$

$$= d\left[2w + \frac{34}{9} \cdot 9^w\right]$$

$$\leq d\left[\frac{2}{9} \cdot 9^w + \frac{34}{9} \cdot 9^w\right] = 4d \cdot 9^w.$$

Suppose v is of minimum distance from the root among those vertices in the jth and final layer. For the analysis alone, add w dummy children to v via length-0 edges. At time $i+1$, v has w active leaf descendants. Thus either $d = 0$ in the current phase, or one vertex $x \in S$ has w active leaf descendants. Hence either $d = 0$, or a phase ends at time j and x becomes the new root. In either case, we can study the cost incurred during *complete* phases.

At all times, define Φ to be the distance from the source to the current root r. Define Ψ to be the length of a shortest path from the source to an active leaf; $\Psi = \Phi + SP(r)$. In a phase, either Φ increases by d, if (1) terminated the phase, or if (2) ended the phase, Ψ increases by at least d. Thus $\Phi + \Psi$ increases within a phase by at least d, and neither Φ nor Ψ ever decreases. It follows that the cost incurred by A_w to visit some vertex in L_i is at most $4 \cdot 9^w$ times the final value of $\Phi + \Psi$, which is at most twice the final value of Ψ. Therefore, A_w is $8 \cdot 9^w$-competitive. □

4. A lower bound for deterministic algorithms. Fix a competitive deterministic layered graph algorithm A for arbitrary layered graphs. A traces out a path in each layered graph. We construct a layered tree that forces A to perform poorly. Figure 1 illustrates the lower bound construction. The construction is recursive. The

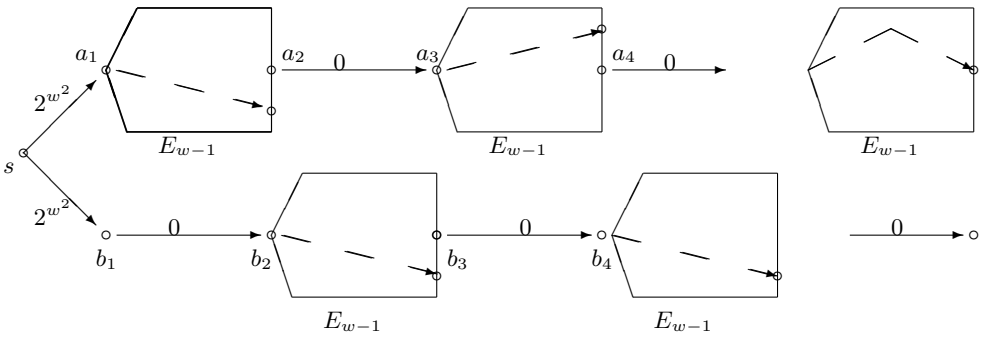


FIG. 1. *Deterministic lower bound.*

idea is that A is forced to move back and forth between the two subtrees attached to the source s, thus incurring a large cost compared with the shortest path to the target.

DEFINITION 8. *Let H be a layered tree. Suppose that $v \in L_{i-1} \neq \emptyset$ is the vertex visited by A at time i.*

1. *Define $T(v)$ to be the minimum $j > i$, if any, such that A visits a nondescendant of v at time j.*

2. *Define $L(v)$ to be the length of a shortest path from v to a descendant of v in layer $T(v)$ (if $T(v)$ and any descendants in layer $T(v)$ exist).*

3. *Define $C(v)$ to be the cost incurred by A from the time when v is first visited until the path traced out by A first exits the subtree rooted at v (if ever). This is exactly the cost incurred by A at times $i+1, i+2, ..., T(v)-1$, plus the portion of the cost incurred at time $T(v)$ attributable to edges in the subgraph rooted at v.*

LEMMA 9. *Let $w \geq 1$. Let H be a layered tree of height i, say, and arbitrary width, with at least two vertices in the ith layer, and let s be the leaf in layer i visited by A. Then there is an infinite rooted tree E_w of width at most w with these properties.*

1. *The root of E_w has $\min\{2, w\}$ children. The edge(s) out of the root are of length 2^{w^2}.*

2. *If E_w is attached to vertex s and an infinite path of length 0 is attached to all other vertices in the ith layer of H, then for this new infinite tree, $L(s)$ exists and $C(s) \geq 2^{w-1}(L(s) - 2^{w^2})$.*

Proof. The proof is by induction on w. Let $w = 1$ and let H be a tree with at least two leaves. If we attach to s an infinite path of edges of length $2^{1^2} = 2$ and attach infinite paths of length 0 to other vertices in the last layer, because A is competitive, $T(s)$ must exist. $C(s) \geq L(s) - 2$. So, clearly, $C(s) \geq 2^{1-1}(L(s) - 2^{1^2})$.

Let $w \geq 2$. Let H be a layered tree and let $s \in L_i$ be visited by A, where $L_{i+1} = \emptyset$ and $|L_i| \geq 2$. Attach to s two children a_1, b_1 via edges of length 2^{w^2}. Add to all other vertices in L_i an edge of length 0.

If A occupies neither a_1 nor b_1 at time $i+1$, then $T(s) = i+1$, $L(s) = 2^{w^2}$, and $C(s) = 0$, so clearly $C(s) \geq 2^{w-1}(L(s) - 2^{w^2})$.

So we may suppose without loss of generality that A visits a_1 at time $i+1$. By induction, there is an infinite tree E_{w-1} of width at most $w-1$ such that if a_1 is extended by E_{w-1} and all other leaves are extended by infinite paths of length 0,

$$C(a_1) \geq 2^{w-2}(L(a_1) - 2^{(w-1)^2}).$$

At time $T(a_1)$, A occupies either a descendant of b_1 or a nondescendant of s. Suppose A occupies a descendant b_2 of b_1. Choose a descendant of a_1 in layer $T(a_1)$ of minimum distance from a_1. Call it a_2. (Such a descendant exists because E_{w-1} is infinite.) "Kill" all other descendants of a_1 in layer $T(a_1)$, i.e., mark them as inactive. They will have no children. Now "truncate" the entire infinite tree to level $T(a_1)$ by removing all vertices in layers $T(a_1)+1, T(a_1)+2, T(a_1)+3, \ldots$.

By the inductive assertion we can find a new infinite tree E'_{w-1} of width at most $w-1$ so that if E'_{w-1} is attached to b_2 and all other vertices in layer $T(a_1)$ (including a_2 but no other descendants of a_1) are extended by 0-length infinite paths, $C(b_2) \geq 2^{w-2}(L(b_2) - 2^{(w-1)^2})$. Now truncate the tree to level $T(b_2)$ by eliminating all vertices in layers $T(b_2)+1, T(b_2)+2, T(b_2)+3, \ldots$. At time $T(b_2)$, A occupies either a descendant a_3 of a_2 or a nondescendant of s. If A occupies a descendant a_3 of a_2 we attach a new infinite tree E''_{w-1} to a_3 and "kill" all descendants of b_2 in layer $T(b_2)$ except for one descendant b_3 of minimum distance from b_2.

This process continues until at some point A visits a nondescendant of s. This must happen eventually, because there is at least one infinite 0-path. Since each stage adds at least 2^{w^2} to A's cost, every competitive algorithm must eventually switch at some time $T(s)$ to a nondescendant of s.

Suppose that the algorithm has constructed $a_1, b_1, a_2, b_2, \ldots, a_k, b_k$ but neither a_{k+1} nor b_{k+1}. Thus A visits either a_k or b_k but exits the subtree rooted at s at time $T(a_k)$ or $T(b_k)$, whichever is defined.

Claim. $C(s)$ increases by at least

$$2^{w^2} + 2^{w-2}(L(a_i) - 2^{(w-1)^2}) \geq 2^{w-2}L(a_i)$$

between the time when A occupies a_i and time $T(a_i)$. Similarly, between the time when A occupies b_i and time $T(b_i)$, $C(s)$ increases by at least

$$2^{w^2} + 2^{w-2}(L(b_i) - 2^{(w-1)^2}) \geq 2^{w-2}L(b_i).$$

Proof of Claim. In moving from a_i to a nondescendant of a_i, A incurs a cost of at least 2^{w^2} on the edges out of s. On the edges in the subtree rooted at a_i, A incurs a cost of

$$C(a_i) \geq 2^{w-2}(L(a_i) - 2^{(w-1)^2})$$

by the inductive case of the theorem. The proof of the second statement is similar.

But if

$$\alpha = L(a_1) + L(a_3) + L(a_5) + \cdots$$

and

$$\beta = L(b_2) + L(b_4) + L(b_6) + \cdots,$$

then

$$L(s) = 2^{w^2} + \min\{\alpha, \beta\}.$$

Thus $C(s) \geq 2^{w-2}(\alpha+\beta) \geq 2^{w-1}\min\{\alpha,\beta\} = 2^{w-1}(L(s) - 2^{w^2})$. Now make the tree infinite, as required, by attaching infinite length-0 paths to each leaf in the final layer. □

Now we prove a lower bound of 2^{w-2} on the competitive ratio.

THEOREM 10. *If A is a layered graph traversal algorithm, then its competitive ratio on width-w graphs is at least 2^{w-2}.*

Proof. We may assume $w \geq 2$. Let s be a source with two children a_1 and b_1 via edges of length 2^{w^2}. Suppose A moves from s to a_1. As in Lemma 9, we can attach to a_1 an infinite tree E_{w-1} of width at most $w-1$ such that $C(a_1) \geq 2^{w-2}(L(a_1) - 2^{(w-1)^2})$ if b_1 is extended by an infinite path of length 0. At time $T(a_1)$, A occupies a descendant b_2 of b_1. Truncate the tree to height $T(a_1)$. Let a_2 be a descendant of a_1 in layer $T(a_1)$, of minimum distance from a_1. All descendants of a_1 in layer $T(a_1)$, other than a_2, will have no children. Now attach to b_2 an infinite tree E'_{w-1}, as in Lemma 9, and to a_2 attach an infinite length-0 path

$$C(b_2) \geq 2^{w-2}(L(b_2) - 2^{(w-1)^2}).$$

At time $T(b_2)$, A occupies a descendant a_3 of a_2. Truncate the tree to height $T(b_2)$. Let b_3 be a descendant of b_2 in layer $T(b_2)$, of minimum distance from b_2. All descendants of b_2 in layer $T(b_2)$, other than b_3, will get no children.

Repeat this process *ad infinitum*. Each pair of additions increases the length of the shortest root−active-leaf path by at least $2^{(w-1)^2}$. Eventually we reach a situation in which we have constructed $a_1, b_1, a_2, b_2, ..., a_k, b_k$ so that if

$$\alpha = L(a_1) + L(a_3) + L(a_5) + \cdots$$

and

$$\beta = L(b_2) + L(b_4) + L(b_6) + \cdots,$$

then $\min\{\alpha, \beta\} \geq 2^{w^2}$. By the claim embedded in the proof of Lemma 9, by that time A's cost is at least

$$2^{w-2}(\alpha + \beta) \geq 2^{w-1} \min\{\alpha, \beta\}.$$

The adversary's cost is

$$2^{w^2} + \min\{\alpha, \beta\} \leq 2 \min\{\alpha, \beta\}.$$

Therefore the competitive ratio is at least

$$\frac{2^{w-1} \min\{\alpha, \beta\}}{2 \min\{\alpha, \beta\}} = 2^{w-2}. \qquad \square$$

5. Disjoint paths. Let L be a layered graph which consists of a set of disjoint paths except that they share the common source. Each edge has a $0-1$ length.

We define the algorithm in phases. At the beginning, while some path has length 0, the algorithm simply chooses such a path and follows it until, if ever, its length increases. It then switches to another path of length 0, and follows that one until its length increases. This continues until all paths have positive length. Then the first phase begins.

In the kth phase ($k = 1, 2, ...$), the length of the shortest path from the source to the current layer lies in the interval $I_k = [2^{k-1}, 2^k)$. At the start of phase k the algorithm chooses a path randomly and uniformly from among those paths of length in I_k running from the source to the current layer. It then backtracks through the

source to the current layer on the chosen path, incurring a cost of at most $2 \cdot 2^k$ in the process.

Whatever path the algorithm is following in phase k, it blindly continues to follow that path until its length reaches 2^k. Whenever the length of the current path reaches 2^k, the algorithm replaces it by a path chosen randomly from those paths of length less than 2^k—if any exist—backtracking through the source and incurring a cost of at most $2 \cdot 2^k$ in the process. A new phase begins and k is incremented as soon as every path has length at least 2^k.

Analysis.

Our initial backtracking cost at the start of a phase is at most $2 \cdot 2^k$. If E_w is an upper bound on the expected number of times the algorithm switches paths within any phase, then the expected cost within phase k is at most $2^{k+1}+E_w 2^{k+1} = 2^{k+1}(1+E_w)$. Let ℓ denote the number of phases. Our total expected cost is bounded above by $(1+E_w)\sum_{k=1}^{\ell} 2^{k+1} < (1+E_w)2^{\ell+2}$. The adversary's cost is at least $2^{\ell-1}$, giving us a competitive ratio bounded by $8+8E_w$. We show that we can take $E_w = H_w = 1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{w} \sim \ln w$.

We now describe a probabilistic game which models the path selection process in a phase. Let S be a set of size n. There are two players A and B. Initially B randomly and uniformly picks one element, hiding his choice from A. At each step A chooses one element of S and removes it from S. Whenever A discards the element selected by B, B pays A \$1 and B uniformly at random picks a new item (if S is still nonempty).

We prove that the expected cost F_n incurred by B is exactly H_n. Clearly, $F_1 = 1$, and for $n \geq 2$, F_n satisfies

$$F_n = \frac{1}{n}(1+F_{n-1}) + \frac{1}{n}(1+F_{n-2}) + \frac{1}{n}(1+F_{n-3}) + \cdots + \frac{1}{n}(1+F_1) + \frac{1}{n}(1+0).$$

This recurrence and the fact that $F_1 = 1$ imply that $F_n = H_n$ for all n since

$$F_n = 1 + \frac{1}{n}(F_1 + F_2 + F_3 + \cdots + F_{n-1}).$$

Thus

$$nF_n = n + (F_1 + F_2 + \cdots + F_{n-1})$$

and

$$(n-1)F_{n-1} = (n-1) + (F_1 + F_2 + \cdots + F_{n-2}),$$

if $n \geq 3$, so

$$nF_n - (n-1)F_{n-1} = 1 + F_{n-1}.$$

Therefore, for $n \geq 3$, $n(F_n - F_{n-1}) = 1$ and $F_n = F_{n-1} + 1/n$. Since $F_2 = 3/2$, it follows that $F_n = H_n$ for all n.

The connection between the experiment and layered graph traversal.

A corresponds to the adversary and B corresponds to the algorithm. Each element in the set is associated with a path in the layered graph of length less than 2^k at the beginning of the kth phase. A discards an element from the set when the length of the corresponding path reaches 2^k. He pays \$1 every time this happens. The expected

number of times B backtracks is at most the expected cost to B of the game above. Thus we may take $E_w = H_w$. We have proven the following theorem.

THEOREM 11. *The competitive ratio of the randomized algorithm above for traversing disjoint paths is at most* $8 + 8H_w$.

A lower bound.

THEOREM 12. *Let w and M be any positive integers. For any randomized on-line algorithm A for traversing disjoint paths of width at most w, there is a width-w layered graph for which the length of the shortest source−target path is M, but on which A's expected cost is at least $M(2H_w - 1)$.*

Proof. Each path in the width-w layered graph begins with M unit-cost edges. For a layered graph that begins this way, at time M there is at least one layer-M vertex which is occupied by the searcher with probability at least $1/w$. We give that vertex no children, but to every other layer-M vertex we give a child via a length-0 edge. At time $M + 1$, at least one of the $w - 1$ layer-$(M + 1)$ vertices is occupied by the searcher with probability at least $1/(w - 1)$. We add a length-0 edge to layer $M + 2$ from every layer-$(M + 1)$ vertex but that one. That one dies. We repeat this process for layers $M+2, M+3, ..., M+(w-1)$; in layer $M+i$ there are exactly $w-i$ vertices, $i = 0, 1, 2, ..., w-1$. The unique vertex in layer $M+w-1$ is the target. The expected cost incurred by A is bounded below by M plus $2M$ times the sum, over each leaf in the graph other than the target, of the probability that A visits that leaf. This sum of probabilities is $\frac{1}{w} + \frac{1}{w-1} + \frac{1}{w-2} + \cdots + \frac{1}{2} = H_w - 1$. The total expected cost is hence at least $M(1 + 2(H_w - 1)) = (2H_w - 1)M$. □

6. A randomized lower bound. Now we return to general layered graphs. Fix an integer $m \geq 2$. Let $r_w = w(1 - 1/m)$ for all w.

By induction on w, we construct for each w a probability distribution $\mathcal{G}(w)$ on a finite family of layered graphs of width w. Every graph drawn from $\mathcal{G}(w)$ has a designated vertex as the root and another as the target; the target is the unique vertex in the final layer. From the inductive construction it will be easy to verify that the following quantities depend only on w and m:

- the length L_w of the shortest root−target path in the graph,
- the sum S_w of the edge lengths,
- the number F_w of layers, excluding L_0 (the layer containing the source).

Let $E_w = 2S_wF_w$. It is clear that this is an upper bound on the distance traversed by any algorithm when it traverses any layered graph drawn from $\mathcal{G}(w)$.

Now we construct the probability distributions. See Figure 2.

Basis: $w = 1$. With probability 1 we draw a single edge (s, t) of length 1 with s the root and t the target.

Inductive Step: $w > 1$. We start with a vertex designated as the root, say s. To s we attach two edges $(s, u_1), (s, l_1)$ of length $(1/2)E_{w-1}$ each. We now construct the graph in stages. For stage 1 we draw a copy H_1 from $\mathcal{G}(w - 1)$ and attach it to u_1 (i.e., make u_1 the root of this copy). The target of H_1 we call u_2. H_1 has F_{w-1} layers of nonsource vertices in it. For these F_{w-1} layers we extend l_1 by a path of F_{w-1} length-0 edges ending at l_2. For stage 2, we extend l_2 by independently drawing a graph H_2 from $\mathcal{G}(w - 1)$, and we extend u_2 by a path of F_{w-1} length-0 edges. We continue this pattern for $N = N_w = mr_{w-1}E_{w-1}$ stages (N is an even integer). In the ith stage, for i odd, we independently select a graph H_i as in stage 1, and for i even, we choose H_i independently as in stage 2. In the last layer we have vertices

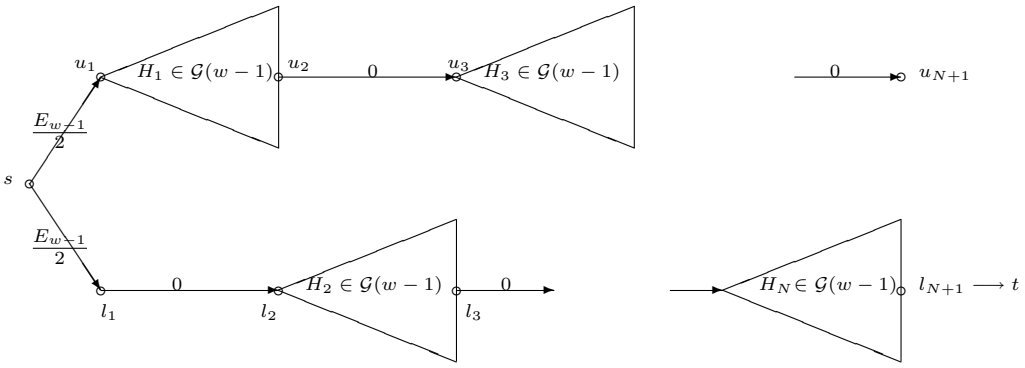


FIG. 2. *Randomized lower bound.*

u_{N+1} and l_{N+1}. We toss a coin and equiprobably choose one. It gets a child, the target, via a length-0 edge; the other gets none. This completes the construction.

LEMMA 13. *For all positive integers w, for all deterministic algorithms A_w designed to traverse graphs drawn from $\mathcal{G}(w)$, the expected cost of A_w to traverse a graph drawn randomly from $\mathcal{G}(w)$ is at least $r_w L_w$.*

Proof. The proof is by induction on w. The $w = 1$ case is trivial.

Let $w \geq 2$. Choose a deterministic algorithm A_w for graphs drawn from $\mathcal{G}(w)$.

Within this proof, we imagine that the random graph H is generated "on the fly"; i.e., only when the searcher reaches either u_i or l_i, for i odd, are the two graphs for stages i and $i+1$ drawn from $\mathcal{G}(w-1)$, and only then are stages i and $i+1$ of H built. This makes no difference, since A_w is on-line and its behavior cannot depend on the future.

Pick an odd $i < N$. At the end of stage $i-1$, the searcher occupies either u_i or l_i. Let J be a graph having $i-1$ stages that induces the searcher to occupy u_i at the end of stage $i-1$ (if possible). Now define an algorithm A_{w-1} (dependent on J) for traversing graphs drawn from $\mathcal{G}(w-1)$, as follows. A_{w-1} mimics A_w in the graph drawn from $\mathcal{G}(w-1)$ in the F_{w-1} layers succeeding u_i, until, if ever, A_w backtracks through s to a nondescendant of u_i. At this point, A_{w-1} blindly marches ahead in a naive way, until u_{i+1} is reached.

The cost of backtracking through s is so large that the cost incurred by A_w in the $2F_{w-1}$ layers succeeding u_i, given that the first $i-1$ stages equal J, is at least the cost of A_{w-1} on those same layers. The inductive hypothesis now implies that the expected cost of A_w in the $2F_{w-1}$ layers succeeding u_i, given J, is at least $r_{w-1}L_{w-1}$.

Now choose an $i-1$-stage graph J', if possible, so that A_w occupies l_i at the end of stage $i-1$. A similar argument implies that the conditional expected cost incurred by A_w in the $2F_{w-1}$ layers succeeding l_i, given that the first $i-1$ stages of H equal J', is at least $r_{w-1}L_{w-1}$. It follows that the (unconditional) expected cost incurred by A_w in progressing from either u_i or l_i to either u_{i+2} or l_{i+2} is at least $r_{w-1}L_{w-1}$.

At the end of stage N, we flip a coin to decide which vertex, u_{N+1} or l_{N+1}, becomes the parent of the target. With probability $1/2$, the searcher must backtrack through s to the target. Thus it incurs an additional expected cost of at least $(1/2)(NL_{w-1} + E_{w-1})$. The total expected cost divided by L_w is at least

$$\begin{aligned}
&\frac{(N/2)r_{w-1}L_{w-1} + (1/2)(NL_{w-1} + E_{w-1})}{(1/2)E_{w-1} + (N/2)L_{w-1}} \\
&= \frac{(N/2)r_{w-1}L_{w-1} + (1/2)r_{w-1}E_{w-1} + (1/2)(NL_{w-1} + E_{w-1}) - (1/2)r_{w-1}E_{w-1}}{(1/2)E_{w-1} + (N/2)L_{w-1}} \\
&= (r_{w-1} + 1) - \frac{(1/2)r_{w-1}E_{w-1}}{(1/2)E_{w-1} + (N/2)L_{w-1}} \\
&\geq (r_{w-1} + 1) - \frac{(1/2)r_{w-1}E_{w-1}}{(N/2)L_{w-1}} \\
&\geq r_{w-1} + 1 - \frac{r_{w-1}E_{w-1}}{N} \\
&= r_{w-1} + (1 - 1/m). \qquad \square
\end{aligned}$$

Now we prove the following theorem.

THEOREM 14. *For every positive integer w, for every randomized algorithm $\mathcal{B}$ for traversing graphs drawn from $\mathcal{G}(w)$, there exists a layered graph K of width at most w such that the ratio of the expected distance traversed by $\mathcal{B}$ to the length of the shortest root−target path in K is at least r_w.*

Proof. The proof follows Yao's observation regarding the minimax principle [Yao]. Choose a randomized algorithm $\mathcal{B}$ and a width w. Lemma 13 implies that the expected cost incurred by every deterministic algorithm $\mathcal{A}$ on a graph drawn randomly from $\mathcal{G}(w)$ is at least $r_w L_w$. However, $\mathcal{B}$ is nothing more than a probability distribution on deterministic algorithms. It follows that the expected cost of $\mathcal{B}$ on a graph drawn randomly from $\mathcal{G}(w)$ is at least $r_w L_w$. It follows that on some graph K assigned positive probability under $\mathcal{G}(w)$, $\mathcal{B}$'s expected cost is at least $r_w L_w$. But the source−target distance in K is L_w. $\square$

7. Metrical service systems. In the following section, w-MSS abbreviates "metrical service systems with requests of size at most w," w-LGT abbreviates "traversal of layered graphs of width at most w," and w-LTT abbreviates "traversal of $0-1$ rooted layered trees of width at most w." (Notice that w-LGT and w-LTT algorithms traverse only graphs of width at most w.)

LEMMA 15. *If A is a c_w-competitive algorithm for w-LGT, then there exist strictly c_w-competitive on-line algorithms for w-MSS in all metric spaces with integral distances.*

Proof. Fix a metric space where the distances are integral. Given a sequence of w-MSS requests, we construct, in an on-line manner, a layered graph. Layer 0 contains a single vertex, which is the starting point of the server. The vertices of layer $i > 0$ are the points of the ith request. For every $i \geq 0$, every vertex of layer i is connected to every vertex of layer $i + 1$ by an edge of weight equal to the distance between the two points. Apply the w-LGT algorithm A to this graph. When A first encounters layer i, it chooses a vertex in that layer to move to. The w-MSS algorithm serves the ith request by moving to that point. $\square$

DEFINITION 16. *Let I be an infinite rooted layered tree in which each vertex has $2w$ children. Let r denote the root of I. Let $\mathcal{M}$ be an infinite metric space whose underlying set is $V(I)$ and in which the distance between u and v is the length of the $u - v$ path in I.*

LEMMA 17. *Let B be a strictly c_w-competitive w-MSS algorithm for the infinite metric space $\mathcal{M}$. Then there exists a c_w-competitive on-line w-LTT algorithm A (and therefore one for w-LGT).*

Proof. Let T be an instance of the w-LTT problem. Let s be the source vertex of T, initially occupied by the searcher. We use B to define algorithm A which traverses T as follows. From the, say, $l_i \leq w$ vertices $v_1^i, v_2^i, ..., v_{l_i}^i$ in the ith layer of T, we construct, on-the-fly, a sequence $p_1^i, p_2^i, ..., p_{l_i}^i$ of l_i vertices of the metric space $\mathcal{M}$ (p_j^i "representing" v_j^i), and then present the set $\{p_1^i, p_2^i, ..., p_{l_i}^i\}$ as a request of $l_i \leq w$ points to B. B will choose one of the points, say, p_j^i, to move to. We stipulate, then, that A moves to v_j^i.

Let us start by defining $v_1^0 := s$, the source vertex of T. Representing v_1^0 is $p_1^0 := r$, the root of I. A starts on the node $p = p_1^0$.

At a generic time, A will occupy some node in, say, layer i of the layered graph. When layer $i+1$ is revealed, we must choose request $i+1$ in $\mathcal{M}$, the response to which tells to which node of layer $i+1$ A should move. This is done as follows. Let the $l_{i+1} \leq w$ nodes of the $i+1$st layer of T be $v_1^{i+1}, v_2^{i+1}, ..., v_{l_{i+1}}^{i+1}$. Look at the edge between a node v_j^{i+1} in the $i+1$st layer and its parent called, say, v_k^i. If the edge between v_j^{i+1} and its parent v_k^i is of weight 0, then we represent v_j^{i+1} by the same node p_k^i that represented its parent: $p_j^{i+1} := p_k^i$. If, on the other hand, the edge from v_j^{i+1} to its parent v_k^i is of weight 1, then we choose a child of p_k^i to represent v_j^{i+1}: we choose, among the $2w$ children of p_k^i, a child which is the root of a subtree in I containing no representative of a vertex in the ith (previous) layer and also containing no representative (so far) of a vertex in layer $i+1$. (Since there are at most $2w$ nodes in layers i and $i+1$, the $2w$ children of p_k^i suffice.) This child is then p_j^{i+1}.

It remains to show that for any two consecutive layers, the distance in T between any pair of vertices contained in those two layers is equal to the distance in I between their representatives. Consider any two consecutive layers numbered i and $i+1$. The proof is by induction on i. The case of $i = 0$ is easy and the proof is omitted. Now consider $i > 0$. Notice that by the inductive hypothesis the claim is true if both vertices in the pair are taken from the ith layer. As we generate the representatives for the vertices in the $i+1$st layer, we check the distances between the representatives and the representatives of vertices in the ith layer, and the distances between their representatives and the representatives already created for vertices in layer $i+1$. Consider a particular vertex v_j^{i+1} of the $i+1$st layer. If the distance to its parent v_k^i in T is 0, then, as described above, we have $p_j^{i+1} = p_k^i$, which is the representative of its parent. Since p_k^i has already been considered in the current step of the induction, the claim trivially holds. If the distance between v_j^{i+1} and v_k^i is 1, the choice of p_j^{i+1} guarantees that its distance to any representative q of a vertex in layer $i+1$ which was already considered in the current step of the induction, or of a vertex in layer i, is exactly the distance between p_k^i and q, plus 1. Thus, the claim holds in this case as well.

Therefore we conclude that at each step, the distance traversed by the w-MSS server is equal to the distance traversed by the w-LTT searcher. We also conclude that the optimal costs for both instances are the same (since an optimal path for one induces a path for the other with the same cost). This completes the proof of the lemma. □

Lemmas 15 and 17 give the following result.

THEOREM 18. *For each w, strictly c_w-competitive, deterministic or randomized algorithms exist for w-MSS for all metric spaces with integral distances if and only if a c_w-competitive, deterministic or randomized algorithm, respectively, exists for w-LGT.*

8. Concluding remarks. An obvious open problem is to close the gap between the upper bound and the lower bound for deterministic and randomized layered graph traversal. Of special interest is the question of designing an efficient randomized traversal algorithm. In an earlier version of this paper, we conjectured that a polynomial upper bound is achievable by the use of randomization. Since then, this conjecture has been proven by Ramesh [Ram], who gives an $O(w^{13})$-competitive randomized algorithm. Ramesh has also reported improvements in the deterministic upper bounds (to $O(w^3 2^w)$) and in the randomized lower bounds (to a nearly quadratic bound). Burley [Bur] recently further improved the deterministic upper bound to $O(w2^w)$ via an algorithm for metrical service systems.

REFERENCES

[AMOT] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan, *Faster algorithms for the shortest path problem*, J. Assoc. Comput. Mach., 37 (1990), pp. 213–223.

[BCR] R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins, *Searching in the plane*, Inform. and Comput., 106 (1993), pp. 234–252.

[Bel] R. Bellman, *On the routing problem*, Quart. Appl. Math., 16 (1958), pp. 87–90.

[BBKTW] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson, *On the power of randomization in on-line algorithms*, Algorithmica, 11 (1994), pp. 2–14.

[BLS] A. Borodin, N. Linial, and M. Saks, *An optimal on-line algorithm for metrical task systems*, J. Assoc. Comput. Mach., 39 (1992), pp. 745–763.

[Bur] W. R. Burley, *Traversing layered graphs using the work function algorithm*, J. Algorithms, 20 (1996), pp. 479–511.

[CDRS] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir, *Random walks on weighted graphs and applications to on-line algorithms*, J. Assoc. Comput. Mach., 40 (1993), pp. 421–453.

[CL] M. Chrobak and L. Larmore, *Server Problems and On-Line Games*, DIMACS Workshop on On-Line Algorithms, February 1991.

[Dij] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[DP] X. Deng and C. H. Papadimitriou, *Exploring an unknown graph*, in Proc. 31st IEEE Annual Symposium on Foundations of Computer Science, 1990, pp. 355–361.

[FKLMSY] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, *Competitive paging algorithms*, J. Algorithms, 12 (1991), pp. 685–699.

[FF] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.

[FL] J. Friedman and N. Linial, *On convex body chasing*, Discrete Comput. Geom., 9 (1993).

[Flo] R. W. Floyd, *Algorithm* 97 *(shortest path)*, Comm. ACM, 5 (1962), p. 345.

[FRR] A. Fiat, Y. Rabani, and Y. Ravid, *Competitive k-server algorithms*, J. Comput. System Sci., 48 (1994), pp. 410–428.

[KRR] H. J. Karloff, Y. Rabani, and Y. Ravid, *Lower bounds for randomized k-server and motion-planning algorithms*, SIAM J. Comput., 23 (1994), pp. 293–312.

[MMS] M. S. Manasse, L. A. McGeoch, and D. D. Sleator, *Competitive algorithms for on-line problems*, J. Algorithms, 11 (1990), pp. 208–230.

[PY] C. H. Papadimitriou and M. Yannakakis, *Shortest paths without a map*, Theoret. Comput. Sci., 84 (1991), pp. 127–150.

[RS] P. Raghavan and M. Snir, *Memory versus randomization in on-line algorithms*, in Proc. 16th ICALP, 1989. Springer-Verlag, New York, pp. 687–703.

[Ram] H. Ramesh, *On traversing layered graphs on-line*, J. Algorithms, 18 (1995), pp. 480–512.

[ST] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

[Yao] A. C. C. Yao, *Probabilistic computations: Towards a unified measure of complexity*, in Proc. 18th IEEE Annual Symposium on Foundations of Computer Science, 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 463–470

ON MULTIRATE REARRANGEABLE CLOS NETWORKS*

D. Z. DU†, B. GAO‡, F. K. HWANG§, AND J. H. KIM¶

Abstract. In the multirate switching environment each (connection) request is associated with a bandwidth weight. We consider a three-stage Clos network and assume that each link has a capacity of one (after normalization). The network is rearrangeable if for all possible sets of requests such that each input and output link generates a total weight not exceeding one, there always exists a set of paths, one for each request, such that the sum of weights of all paths going through a link does not exceed the link capacity. The question is to determine the minimum number of center switches which guarantees rearrangeability. We obtain a lower bound of $11n/9$ and an upper bound of $41n/16$. We then extend the result for the three-stage Clos network to the multistage Clos network. Finally, we propose the weighted version of the edge-coloring problem, which somehow has escaped the literature, associated with our switching network problem.

Key words. multirate switching, rearrangeable, Clos network

AMS subject classifications. 94C15, 0515

PII. S0097539795284716

1. Introduction.

Let $C(n, m, r)$ denote a symmetric three-stage Clos network satisfying the following conditions.

1. Each of the *input* (first) and *output* (third) stage consists of r $n \times m$ crossbars. The *center* (second) stage consists of m $r \times r$ crossbars.
2. There exists one link between every center switch and every input (output) switch. No link exists between any other pair of switches.

A $C(2, 3, 4)$ is illustrated in Figure 1.

The inlets (outlets) of the input (output) switches are the *inputs (outputs)* of the network. Inputs and outputs are referred to as *external links*, while links between switches are referred to as *internal links*. When $C(n, m, r)$ is used in the multirate environment, each connection is associated with a weight which can be thought of as the bandwidth requirement of that connection. A link can carry any number of connections as long as the *load*, total weight of these connections, does not exceed the link capacity. It is commonly assumed that links have uniform capacity which is normalized to be unity.

A (connection) *request* is a triple (x, y, w) where x is an inlet, y an outlet, and w the weight. A *request frame* is a collection of requests such that the total weight of all requests in the frame involving a fixed inlet or outlet does not exceed unity. To discuss routing it is convenient to assume that all links are directed from left to right. Thus a *path* from an inlet to any outlet always consists of the sequence: an inlet link → an input switch → a link → a center switch → a link → an output switch → an outlet link. Furthermore, since the crossbar is assumed to be nonblocking with fan-in and fan-out properties, a request (x, y, w) is *routable* if and only if there exists a path

*Received by the editors April 17, 1995; accepted for publication (in revised form) March 19, 1996; published electronically July 28, 1998.
http://www.siam.org/journals/sicomp/28-2/28471.html
†Department of Computer Science, University of Minnesota, Minneapolis, MN 55455 (dzd@cs.umn.edu).
‡Lattice Semiconductor Corporation, Milpitas, CA 95035 (biao@lscjc.latticesemi.com).
§Department of Applied Mathematics, Chiaotung University, Hsinchu Taiwan 30050, Republic of China (fhwang@math.nctu.edu.tu).
¶Bell Laboratories, Lucent Technologies, Murray Hill, NJ 07974 (jhk@research.att.com).

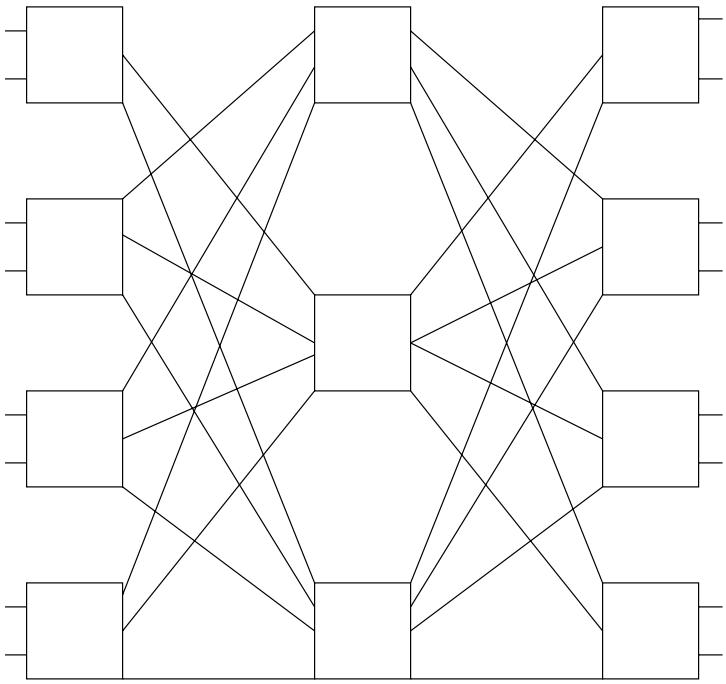


FIG. 1. $C(2,3,4)$.

from x to y such that every link on this path has unused capacity at least $1-w$ before carrying out this request. A request frame is routable if there exists a set of paths, one for each request, such that for every link the sum of weights of all requests going through it does not exceed unity. $C(n,m,r)$ is called *rearrangeable* if every request frame is routable.

Let $M(n,r)$ denote the minimum value of m such that $C(n,m,r)$ is rearrangeable for given n and r. In this paper we prove that $11n/9 \le M(n,r) \le 41n/16$. (Chung and Ross [2] have conjectured that a strictly nonblocking network for the classical circuit switching, hence $C(n,2n-1,r)$, is rearrangeable if the weights belong to a given finite set.) We also describe a weighted edge-coloring problem which is related to the multirate routing problem.

2. When weights are restricted. Let B and b denote the upper and lower bound of the weight of a request. Let $M_{(B,b)}(n,r)$ denote the corresponding $M(n,r)$ when B and b are given. Then $M(n,r)$ can also be interpreted as $M_{(1,0)}(n,r)$. Define a bipartite graph G with the input switches as one part, the output switches as the other part, and an edge with weight w between vertices X and Y for each request (x,y,w) where $x(y)$ is an inlet (outlet) of the switch $X(Y)$. The routing problem for $C(n,m,r)$ can then be formulated as an edge-coloring problem on G where the requirement is that the sum of weights of all edges of the same color at a vertex cannot exceed one. To see this, simply color each center switch with a distinct color and route a request through the switch with the same color.

Melen and Turner [5] gave an elegant routing algorithm, CAP, for the Beneš network. Here we adopt it for $C(n,m,r)$ to prove Lemma 1. Let $d(v)$ denote the degree of v in G.

LEMMA 1. $M_{(B,0)}(n,r) \le \frac{n-B}{1-B}$.

Proof. For each vertex v in G, order its $d(v)$ edges into $q(v) \equiv \lceil d(v)/m \rceil$ groups $g_1(v), \ldots, g_{q(v)}(v)$, where $g_1(v)$ consists of the m heaviest edges, $g_2(v)$ of the next m heaviest edges, and so on ($g_{q(v)}(v)$ may have less than m edges). Let G' be a bipartite graph obtained from G by splitting each vertex v into $q(v)$ vertices $v_1, \ldots, v_{q(v)}$ with edges $g_1(v), \ldots, g_{q(v)}(v)$, respectively. Then the maximum degree of a vertex in G' is upper bounded by m. Hence the edges of G' can be m-colored by the König theorem [3]. Note that an edge-coloring of G' induces an edge-coloring of G. Let $W(v,c)$ denote the set of weights of edges of color c at vertex v. Order the weights in $W(v,c)$ into $w_1(v,c) \ge w_2(v,c) \ge \cdots \ge w_{q(v)}(v,c)$, where $w_{q(v)}(v,c) = 0$ if $w(v,c)$ has only $q(v)-1$ edges. Then

$$w_1(v,c) \le B,\ w_{i+1}(v,c) \le w_i(v,c') \quad \text{for} \quad i = 1, \ldots q-1 \quad \text{and any } c' \ne c\ .$$

Define $w(v,c) = \sum_{i=1}^{q(v)} w_i(v,c)$. Then

$$w(v,c) - B \le w(v,c') \quad \text{for any} \quad c' \ne c\ .$$

Adding the above inequality over the $m-1$ colors other than c, plus the equality

$$w(v,c) - B = w(v,c) - B\ ,$$

we obtain

$$m(w(v,c) - B) \le n - B\ ,$$

or

$$w(v,c) \le \frac{n}{m} + \frac{m-1}{m} B\ .$$

Therefore $w(v,c) \le 1$ will be assured if

$$\frac{n}{m} + \frac{m-1}{m} B \le 1$$

or

$$m \ge \frac{n-B}{1-B}\ . \qquad \square$$

COROLLARY 1. $M_{(1/2,0)}(n,r) \le 2n-1$.

On the other hand, we can also use b to obtain an upper bound of $M_{(1,b)}(n,r)$.

LEMMA 2. $M_{(1,b)}(n,r) \le n\lfloor 1/b \rfloor$.

Proof. Since each inlet (outlet) can appear in at most $\lfloor 1/b \rfloor$ requests, the maximum degree of a vertex in G is upper bounded by $n\lfloor 1/b \rfloor$. Again, by the König theorem, G can be edge-colored in $n\lfloor 1/b \rfloor$ colors. $\square$

Note that Lemma 1 is not very useful when $B \to 1$, and Lemma 2 is not useful when $b \to 0$. Therefore, Lemma 1 and Lemma 2, individually, do not contribute to bounding $M(n,r)$. However, they can be combined to yield the following result.

THEOREM 1. $M(n,r) \le 3n-1$.

Proof. Partition the weights into two groups, those $> 1/2$ and those $\le 1/2$. We route these two groups through two disjoint sets of center switches. The first group has a lower bound $b > 1/2$. By Lemma 2, its routing requires n center switches. The

second group has an upper bound $B = 1/2$. By Corollary 1, its routing requires $2n-1$ center switches. Hence $n + (2n-1) = 3n-1$ center switches suffice. □

With more restrictive bounds, better results can be obtained. For example, Chung and Ross proved that a network which is rearrangeable for the classical circuit switching is multirate rearrangeable if all weights are equal. A corollary of this result is that a network which is strictly nonblocking for the classical circuit switching is multirate rearrangeable if the weights are either 1 or a constant b, $0 < b < 1$. Since $C(n, n, r)$ is rearrangeable and $C(n, 2n-1, r)$ strictly nonblocking [1] for the classical circuit switching, they are multirate rearrangeable when the weights meet the above conditions. We now generalize the results of Chung and Ross under more flexible weight conditions.

THEOREM 2. *A network which is rearrangeable for the classical circuit switching is multirate rearrangeable if all weights are in the interval* $(1/(k+1),\ 1/k]$ *for some positive integer* k.

Proof. Construct a bipartite graph H with the inputs and the outputs as the two parts of vertices, and an edge between u and v if there exists a request between u and v. Since each vertex can have at most k edges, the edges of H can be k-colored. Edges of a given color represent a matching between inputs and outputs. Since the network is assumed to be rearrangeable for the classical circuit switching, it can route any such matching (where each internal link carries at most one path). When we combine the routing of the k colors, then each internal link carries at most k paths, one in each color. Therefore, the load of an internal link is upper bounded by $k(1/k) = 1$. □

COROLLARY 2. *A network which is strictly nonblocking for the classical circuit switching is multirate rearrangeable if all weights not exceeding* $1-b$ *are in the interval* $(1/(k+1),\ 1/k]$, *where* $k = \lfloor 1/b \rfloor$ *for some* $0 < b \le 1/2$.

Proof. Each input (output) can be involved with at most one request with weight exceeding $1-b$. Since the network is strictly nonblocking for the classical circuit switching, it can route all these large weights. Delete these requests from the request frame and delete these paths from the network; the remaining network must still be strictly nonblocking, hence rearrangeable for the remaining inputs and outputs. Apply Theorem 2. □

By k-rate we mean that the set of weights consists of k distinct numbers.

COROLLARY 3. $C(n, kn, r)$ *is* k*-rate rearrangeable.*

Proof. Let $w_1, \ldots, w_k$ denote the k types of rate. Reserve n center switches for each type. □

Let $M^k(n, r)$ denote $M(n, r)$ conditional on k-rate. Clearly, $M^1(n, r) \ge n$, and by Theorem 2, $M^1(n, r) = n$. By stretching an example given by Chung and Ross for $n = 2$, we obtain the following theorem.

THEOREM 3. $M^2(n, r) \ge n + 1$ *for* $r \ge 3$.

Proof. Suppose n colors suffice. Label the inputs (outputs) of the ith input (output) switch by the set $\{(i-1)n+1, \ldots, in\}$. Consider a request frame containing the following requests: $\{(i, i, 1),\ 1 \le i \le 2n-1,\ i \ne n,\ (n, 2n, 1/2),\ (n, 3n, 1/2),\ (2n, 2n+1, 1)\}$. Since the $n-1$ weights $(i, i, 1)$, $1 \le i \le n-1$, must each take a different color, $(n, 2n, 1/2)$ and $(n, 3n, 1/2)$ must take the same color, say blue. Then the n unity weights from the second input switch must have distinct colors other than blue, which is impossible. □

THEOREM 4. $M^3(n, r) \ge 11n/9$ *for* $r \ge 3$.

Proof. Consider a bipartite graph with vertices $(X, Y, Z) \times (U, V)$ and the request frame $\{(X, U, 0.6)^{2n/3},\ (Y, U, 0.4)^{2n/3},\ (Z, U, 0.7)^{n/3},\ (X, V, 0.7)^{n/3},\ (Y, V, 0.7)^{2n/3}\}$,

where the exponent indicates the number of copies of the request in the (multi)set. Note that at vertex Y, the $2n/3$ 0.4-weights can come from $n/3$ inlets each having two-requests.

Consider vertex U and let f denote the proportion of 0.4-weights which are paired with 0.6-weights in coloring. Note that a 0.7-weight cannot be paired with any weight. The number of colors at U is at least $n+(2n/3-fn)/2$, where the latter term denotes the number of colors required by those 0.4-weights paired with each other (but not with the 0.6-weights). Note that none of those fn colors can appear at vertex V, or a conflict would have occurred either at vertex X or vertex Y. Hence at least $n+fn$ colors are required. Therefore

$$M^3(n,r) \geq \max\left\{n + \frac{2n/3 - fn}{2},\ n + fn\right\}$$
$$\geq \max\left\{n + \frac{2n/3 - 2n/9}{2},\ n + 2n/9\right\} = 11n/9\ . \qquad \square$$

3. The general weight case. We first prove a lemma.

LEMMA 3. *Suppose that all weights $> 1/f$, f an integer, are colored by a set C of $c \geq 2n$ colors. Then at most $\lceil (c-2)/f - c + 2n \rceil$ additional colors are needed which, together with C, color all weights $\leq 1/f$.*

Proof. Color the small weights (those $\leq 1/f$) by C until no more coloring is possible. Let S denote the bipartite graph on the uncolored small weights. We prove that the maximum degree of $S \leq c - 2 - (c-2n)f$. Consider vertex u. Let w be the minimum weight at u in S. Suppose w is between vertices u and v. Let $W_j(i)$ denote the weight of all edges in color $j \in C$ at vertex i. Then for each j,

$$W_j(u) + W_j(v) \geq \max\{W_j(u),\ W_j(v)\} > 1 - w\ ,$$

or w would be colored by j. Summing over all j,

$$W(u) + W(v) \equiv \sum_{j \in C} W_j(u) + \sum_{j \in C} W_j(v) > c(1-w)\ .$$

But

$$W(v) \leq n - w\ .$$

Hence

$$W(u) > c(1-w) - (n-w) = c - n - (c-1)w\ .$$

This implies that the total uncolored small weights involving u is at most

$$n - W(u) < (c-1)w - c + 2n\ .$$

But a small weight is lower bounded by w, hence the number of small weights at u is fewer than

$$c - 1 - \frac{c-2n}{w} \leq c - 1 - (c-2n)f \text{ for } c \geq 2n\ .$$

Since u is arbitrary, S has maximum degree $c - 2 - (c-2n)f$, and hence can be $\lceil c - 2 - (c-2n)f \rceil$-colored by the König theorem. But each color has at most one

edge at a vertex and the weight of that edge is at most $1/f$. Hence we can combine any f colors into one and the weight of each color at a vertex is still at most one. Therefore, the number of additional colors required is $\lceil (c-2)/f - c + 2n \rceil$. □

Let $d_G(v)$ denote the degree of a vertex v in G. A *spanning subgraph* of G is a subgraph with the same vertex-set as G (although a vertex can have zero degree). We quote a result from de Werra [6] (also see Lovász [4, p. 56]).

LEMMA 4. *Let G be any bipartite graph and suppose $k \geq 1$. Then G is the union of k edge-disjoint spanning subgraphs $G_1, \ldots, G_k$ such that*

$$\left\lfloor \frac{d_G(v)}{k} \right\rfloor \leq d_{G_i}(v) \leq \left\lceil \frac{d_G(v)}{k} \right\rceil \quad \textit{for each} \quad v \in G \ .$$

The subgraphs are required to be spanning so that the "union" operation is well defined.

THEOREM 5. *$C(n, m, r)$ is rearrangeable if $m \geq \lceil (41n - E_n)/16 \rceil$, where $E_n = 8, 5, 6, 3$ if $n \equiv 0, 1, 2, 3 \pmod 4$.*

Proof. Partition the weights into *large*: $w > 1/2$, *medium*: $1/4 < w \leq 1/2$, and *small*: $0 < w \leq 1/4$. Let $l_i(m_i)$ denote the number of large (medium) weights associated with vertex i. Then $l_i \leq n$ and $m_i \leq l_i + 3(n - l_i)$ since each external link having a large weight can have at most one medium weight, and each other external link can have at most three medium weights. For easier presentation, assume 4 divides n.

Consider the bipartite graph G defined in section 2. Let L denote the subgraph of G consisting of only edges of large weights and M consisting of medium weights. By Lemma 4, $M = M_1 \cup M_2$, where M_1 and M_2 are edge-disjoint spanning subgraphs of M and for $j = 1, 2, d_{M_j}(i) \leq \frac{3n}{2} - l_i$ for all vertices i in M.

Let $L' = L + M_1$. Then vertex i has degree at most $3n/2$ in L'. Hence edges in L' can be $3n/2$-colored. Furthermore, edges in M_2 can also be $3n/2$-colored. But since each edge in M_2 has weight at most $1/2$, we can combine any two colors into one without violating the condition that each color carries a load not exceeding one. In other words, M_2 can be $3n/4$-colored in our sense.

Let C denote the set of colors used in coloring L' and M_2. Then $|C| \leq 3n/2 + 3n/4 = 9n/4$. Add colors to C, if necessary, so that $|C| = 9n/4$. By Lemma 3, the small weights can be colored in at most ($f = 4$)

$$\left\lceil \frac{9n/4 - 2}{4} - (9/4 - 2)n \right\rceil = \left\lceil \frac{5n - 8}{16} \right\rceil$$

additional colors. Hence the total number of colors needed is

$$\frac{9n}{4} + \left\lceil \frac{5n - 8}{16} \right\rceil = \left\lceil \frac{41n - 8}{16} \right\rceil \ .$$

The case in which 4 does not divide n can be derived similarly. □

Note that

$$M(n, r) \geq M^k(n, r) \geq M^{k-1}(n, r) \text{ for } k \geq 2 \ .$$

Hence any lower bounds of $M^k(n, r)$, in particular, those obtained in section 2, are also lower bounds of $M(n, r)$.

4. Extensions. We first note that Theorems 1–5 are independent of r (Theorems 3 and 4 require $r \geq 3$); hence they hold even if the input stage and the output stage have different numbers of switches. Next we consider the case in which the number of inlets per input switch is different from the number of outlets per output switch. Namely, consider the asymmetric 3-stage Clos network $C(n_1, n_2, m, r_1, r_2)$ whose input stage consists of r_1 $n_1 \times m$ crossbars, whose center stage consists of m $r_1 \times r_2$ crossbars, and whose output stage consists of r_2 $m \times n_2$ crossbars. Define $n = \max(n_1, n_2)$. Without loss of generality, assume $n = n_1$. Add $n - n_2$ imaginary outlets to each output switch. Then $C(n_1, n_2, m, r_1, r_2)$ can be treated as $C(n, n, m, r_1, r_2)$, except the imaginary outlets do not appear in the request frame. Thus we have Theorem 6.

THEOREM 6. *$C(n_1, n_2, m, r_1, r_2)$ is multirate rearrangeable if $m \geq \lceil (41n - E_n)/16 \rceil$, where $n = \max\{n_1, n_2\}$.*

With Theorem 4 we can generalize our result on three-stage Clos networks to *multistage Clos networks*. A $(2k + 1)$-stage, $k \geq 2$, Clos network can be obtained from a $(2k - 1)$-stage Clos network by replacing a stage of $p \times q$ crossbars with a $C(n_1, n_2, m, r_1, r_2)$ such that $n_1 r_1 = p$ and $n_2 r_2 = q$.

THEOREM 7. *Suppose that we start with a multirate rearrangeable Clos network and then replace the crossbars only with multirate rearrangeable three-stage Clos networks. Then the resultant network is multirate rearrangeable.*

Note that the external links of the three-stage Clos networks replacing the crossbars are internal links of the multistage Clos network. This explains why we assume that both external and internal links have the same capacity in the three-stage Clos network model.

For applications in which the external links have a capacity larger than the internal link, let k denote the smallest integer not less than the external link capacity. By partitioning each request (x, y, w) into k copies of $(x, y, w/k)$, we can handle each copy by the results obtained for the uniform case. The corresponding results for the nonuniform case are obtained by multiplying by a factor of k.

5. The weighted edge-coloring problem. Let $G = (V, E, W)$ denote a weighted multigraph where each edge e is associated with a weight $w(e) \in W$, $b \leq w(e) \leq B$, and b and B are constants satisfying $0 < b \leq B \leq 1$. The edges are to be colored such that for each vertex v and each color c, the sum of weights of all edges incident to v with color c is at most unity. The problem is to minimize the number of colors. For $b = B = 1$, the weighted edge-coloring problem is reduced to the usual edge-coloring problem. Surprisingly, this very natural extension of the edge-coloring problem seems to have been neglected in the literature.

The notion of maximum degree plays an important role in the edge-coloring problem. In the weighted version, the counterpart is the maximum load of a vertex while the load of a vertex is the sum of weights of all edges incident to it.

The routing problem studied in this paper corresponds to a bipartite weighted edge-coloring problem where the maximum load is n. However, Theorems 1–4 assume some information from the external links which are not in G; i.e., the load of a vertex can be decomposed into n subsets each having a load at most unity. We now give a result on the bipartite weighted edge-coloring problem which does not assume this extraneous condition.

Let $W(n)$ denote the minimum number of colors required in the above bipartite weighted edge-coloring problem.

THEOREM 8. $W(n) \leq \lceil (17n - 5)/6 \rceil$.

Proof. For a given vertex i partition all its weights into l_i large weights: $w > 1/2$, m_i medium weights: $1/2 \geq w > 1/3$, and s_i small weights: $w \leq 1/3$. Let L, M, and S denote the bipartite graphs on the large, medium, and small weights, respectively. Then

$$\frac{l_i}{2} + \frac{m_i}{3} \leq n,$$

or equivalently,

$$m_i \leq 3n - \frac{3l_i}{2} .$$

By Lemma 4, $M = M_1 \cup M_2 \cup M_3$, where the M_i's are edge-disjoint spanning subgraphs of M such that $d_{M_j}(i) \leq \lceil n - l_i/2 \rceil$. Let L' denote the union of L, M_1, and M_2. Then the maximum degree of L' is at most

$$\max_i \left\{ l_i + 2\left(n - \frac{l_i}{2}\right) \right\} = 2n,$$

hence $2n$ colors suffice to color L'.

The maximum degree of M_3 is at most

$$\max_i \left\{ \left\lceil n - \frac{l_i}{2} \right\rceil \right\} \leq n ,$$

hence n colors suffice to color M_3. But since the weight of each edge in M_3 does not exceed $1/2$, we can combine any two colors into one, and the weight of a color at a vertex still does not exceed one. Hence $\lceil \frac{n}{2} \rceil$ colors suffice for M. Adding the colors in L' and M_3, we have

$$2n - 1 + \left\lceil \frac{n}{2} \right\rceil \leq \frac{5n-1}{2} .$$

By Lemma 4 ($f = 3$), at most

$$\left\lceil \frac{(5n-5)/2}{3} - \frac{5n-1}{2} + 2n \right\rceil = \left\lceil \frac{n-1}{3} \right\rceil$$

additional colors are needed. So the total number of colors is at most $\lceil \frac{17n-5}{6} \rceil$. ∎

Next we prove a lower bound of $W(n)$.

THEOREM 9. $W(n) \geq 2n - 1$.

Proof. Let vertex v have $2n-1$ edges of weight $n/(2n-1) > 1/2$. Then all of these $2n-1$ edges must have distinct colors. ∎

Acknowledgment. We thank a referee for many helpful comments.

REFERENCES

[1] V. E. BENEŠ, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, 1965.
[2] S.-P.CHUNG AND K. W. ROSS, *On nonblocking multirate interconnection networks*, SIAM J. Comput., 20 (1991), pp. 726–736.
[3] D. KÖNIG, *Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre*, Math. Ann., 77 (1916), pp. 453–465.
[4] L. LOVÁSZ, *Combinatorial Problems and Exercises*, 2nd ed., North Holland, Amsterdam, 1993.
[5] R. MELEN AND J. S. TURNER, *Nonblocking multirate networks*, SIAM J. Comput., 18 (1989), pp. 301–313.
[6] D. DE WERRA, *Balanced schedules*, Inform. J., 9 (1971), pp. 453–465.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 471–486

FINDING THE CONSTRAINED DELAUNAY TRIANGULATION AND CONSTRAINED VORONOI DIAGRAM OF A SIMPLE POLYGON IN LINEAR TIME*

FRANCIS CHIN† AND CAO AN WANG‡

Abstract. In this paper, we present an $\Theta(n)$ time worst-case deterministic algorithm for finding the constrained Delaunay triangulation and constrained Voronoi diagram of a simple n-sided polygon in the plane. Up to now, only an $O(n \log n)$ worst-case deterministic and an $O(n)$ expected time bound have been shown, leaving an $O(n)$ deterministic solution open to conjecture.

Key words. algorithm, computational geometry, constrained Delaunay triangulation, polygon, Voronoi diagram

AMS subject classifications. 68P05, 68Q20, 68Q25, 68U05

PII. S0097539795285916

1. Introduction.

Delaunay triangulation and *Voronoi diagram*, duals of one another, are two fundamental geometric constructs in computational geometry. These two geometric constructs for a set of points as well as their variations have been extensively studied [17, 3, 4]. Among these variations, Lee and Lin [15] considered two problems related to *constrained Delaunay triangulation*:[1]

(i) the Delaunay triangulation of a set of points constrained by a set of noncrossing line segments, and
(ii) the Delaunay triangulation of the vertices of a simple polygon constrained by its edges.

They proposed an $O(n^2)$ algorithm for the first problem and an $O(n \log n)$ algorithm for the second one. While the $O(n^2)$ upper bound for the first problem was later improved to $\Theta(n \log n)$ by several researchers [6, 21, 18], the upper bound for the second has remained unchanged and the quest for an improvement has become a recognized open problem [1, 3, 4].

Recently, there have been some results related to this open problem on the Delaunay triangulation of simple polygons. Aggarwal et al. [2] showed that the constrained Delaunay triangulation of a convex polygon can be constructed in linear time. Chazelle [5] presented a linear-time algorithm for finding an "arbitrary" triangulation of a simple polygon. Klein and Lingas showed that the aforementioned open problem for L_1 metrics can be solved in linear time [12], and this problem for the Euclidean metrics can be solved in expected linear time by a randomized algorithm [13]. These efforts all seem to point toward a linear solution to the Delaunay triangulation of simple polygons and support the intuition that the simple polygon problem is easier than the noncrossing line segment problem.

In this paper, we settle this open problem by presenting a deterministic linear-time worst-case algorithm. Our approach follows that of [13]:

*Received by the editors May 5, 1995; accepted for publication (in revised form) August 22, 1996; published electronically July 28, 1998. This work was partially supported by the RGC research grant HKU 439/94E and the NSERC grant OPG0041629.
http://www.siam.org/journals/sicomp/28-2/28591.html
†Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong (chin@cs.hku.hk).
‡Department of Computer Science, Memorial University of Newfoundland, St. John's, NFLD, Canada A1C 5S7 (wang@garfield.cs.mun.ca).
[1]Same as *generalized Delaunay triangulation* as defined in [15]

(i) first decomposing the given simple polygon into a set of simpler polygons, called *pseudonormal histograms* (*PNHs*),
(ii) constructing the constrained Delaunay triangulation of each normal histogram (NH), and
(iii) merging the constrained Delaunay triangulations of all these NHs to get the result.

In this three-step process, the first and third steps were shown to be possible in linear time, but the second step was done in expected linear time by a randomized algorithm. Our contribution is to show how this second step can be done in linear worst-case time deterministically.

The organization of the paper is as follows. In section 2, we review some definitions and known facts, which are related to our method. In section 3, we concentrate on how to construct the constrained Delaunay triangulation or constrained Voronoi diagram of a normal histogram in linear time. We conclude the paper in section 4.

2. Preliminaries. In this section,
(i) we define the constrained Delaunay triangulation and its dual, the constrained Voronoi diagram,
(ii) we define PNHs, and
(iii) to put our solution of how to construct the constrained Voronoi diagram of a PNH into perspective, we explain the approach taken to first partition any simple polygon into PNHs and then merge constrained Voronoi diagrams of these pseudodiagrams for the solution of the original polygon.

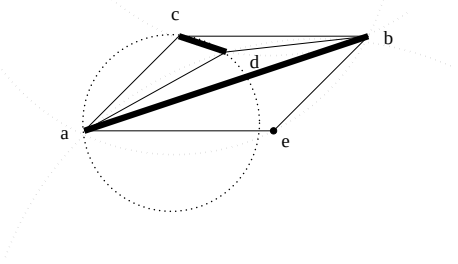
2.1. Constrained Delaunay triangulations and constrained Voronoi diagrams. The *constrained Delaunay triangulation* [15, 6, 21, 18] of a set of noncrossing line segments L, denoted by $CDT(L)$, is a triangulation of the endpoints S of L satisfying the following two conditions:
(i) the edge set of $CDT(L)$ contains L, and
(ii) the line segments in L are treated as obstacles and the interior of the circumcircle of any triangle of $CDT(L)$, say $\Delta ss's''$, does not contain any endpoint in S visible[2] to all vertices s, s', and s''.

Essentially, the constrained Delaunay triangulation is the Delaunay triangulation with the further constraint that the triangulation must contain a set of designated line segments. Figure 1a shows the constrained Delaunay triangulation of two obstacle line segments and a point (a degenerated line segment). In particular, if L forms a nonintersecting chain C, monotone with respect to a horizontal line l, we are only interested in the portion of $CDT(C)$ between C and l. If L forms a simple polygon P, we only consider the portion of $CDT(P)$ internal to P.

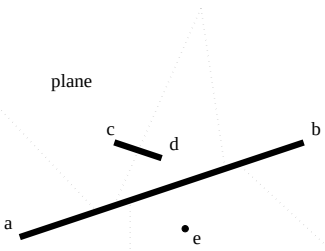
Given a set of line segments L, we can define the Voronoi diagram with respect to L as a partition of the plane into cells, one for each endpoint set S of L, such that a point p belongs to the cell of an endpoint v if and only if v is the closest endpoint visible from p. Figure 1b illustrates the corresponding Voronoi diagram for the set of line segments given in Figure 1a. Unfortunately, this Voronoi diagram is not the complete dual diagram of $CDT(L)$ [3]; i.e., some of the edges in $CDT(L)$ may not have a corresponding edge in this Voronoi diagram.

In [18, 16, 9], the proper dual for the constrained Delaunay triangulation has been defined as the *constrained (or bounded) Voronoi diagram* of L, denoted by $V_c(L)$.

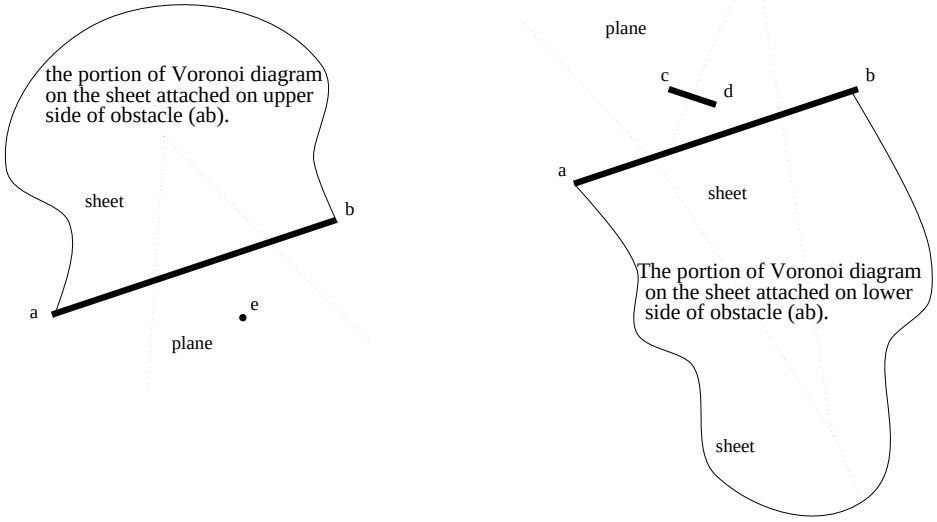
[2]Two points are *visible* to each other if the straight line joining them does not intersect any line segments in L.



(a) the constrained Delaunay triangulation of line segments (ab) and (cd), and point e, (L).



(b) Voronoi diagram w.r.t. L



(c) Constrained Voronoi diagram on 'extended' space.

FIG. 1. *Constrained Delaunay triangulation and constrained Voronoi diagram.*

Imagine two sheets or half-planes attached to each side of the obstacle line segments; for each sheet, there is a well-defined Voronoi diagram that is induced only by the endpoints on the other side of the sheet excluding the obstacle line segment attached to the sheet. The constrained Voronoi diagram extends the standard Voronoi diagram by including the Voronoi diagrams induced by the sheets, i.e., the extended Voronoi diagrams beyond both sides of each line segment in L. Figure 1c gives an example of $V_c(L)$, the Voronoi diagrams on the plane and on the two sheets of the obstacle line segment $\overline{ab}$. Note that the Voronoi diagrams on the two sheets of the obstacle line segment $\overline{cd}$ happened to be the same as the Voronoi diagram on the plane. With this definition of $V_c(L)$, there is a one-to-one duality relationship between edges in $V_c(L)$ and edges in $CDT(L)$. It was further proved in [18, 9] that the dual diagrams, $CDT(L)$ and $V_c(L)$, can be constructed from each other in linear time. For simplicity, we omit the word "constrained" over Voronoi diagrams in this paper as all the Voronoi diagrams are deemed to be constrained unless they are explicitly stated to be standard Voronoi diagrams.

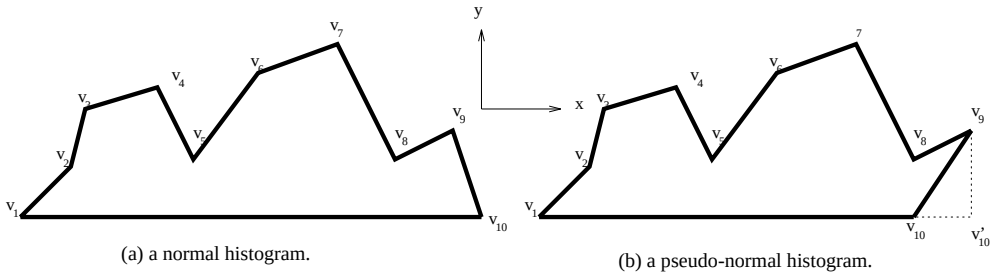


FIG. 2. *NH and PNH.*

2.2. PNHs. A NH [8] is a monotone polygon with respect to one of its edges, called the *bottom edge*, such that all the vertices of the polygon lie on the same side of the line extending the bottom edge (Figure 2a gives an example). A PNH [13] with a bottom edge e is a simple polygon which, by adding at most one right-angle triangle flush with e, can be transformed into a NH whose bottom edge is the extension of e by the colinear edge of the triangle. Intuitively, a PNH can be viewed as a NH missing one of its bottom corners; i.e., a PNH can be transformed into a NH by adding a right-angle triangle at its bottom[3] (Figure 2b).

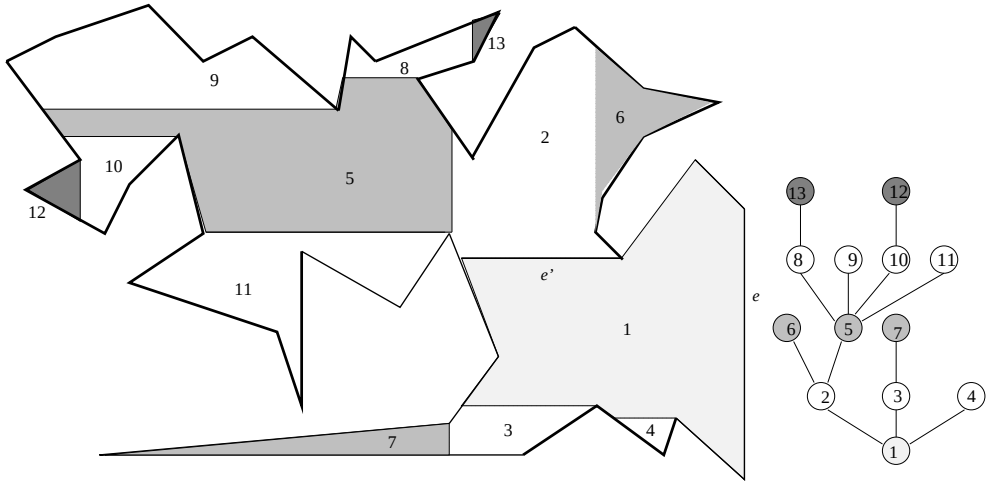


FIG. 3. *A decomposition of P into a tree of PNHs.*

2.3. Decomposition of a simple polygon into PNHs. Figure 3 illustrates how a polygon P is decomposed into 13 PNHs. PNH_1 is associated with the vertical bottom edge e missing its upper bottom corner; PNH_2, associated with the horizontal bottom edge e', is missing its left bottom corner, etc.

A simple polygon P with n vertices can be decomposed into PNHs in $O(n)$ time according to [13] when provided with what are known as the *horizontal* and *vertical visibility maps* of P (Figure 4), which in turn can be obtained in linear time according to [5]. A *diagonal* of P is a line segment joining two vertices of P and lying entirely inside P, while a *chord* of P is a line segment which

[3]Our definition of PNH is different from that given in [13], in which a PNH might be missing both bottom corners. Following the same approach as given in [13], decomposition of a simple polygon into PNHs is also possible with our definition.

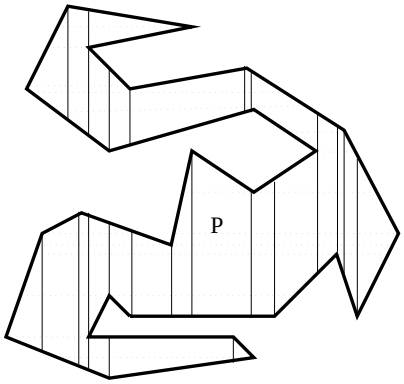


FIG. 4. *Horizontal and vertical visibility maps of simple polygon P.*

(i) lies entirely inside P,
(ii) is parallel to the designated bottom edge, and
(iii) joins a vertex and a boundary point of P (such a boundary point is called a *pseudovertex*).

The *horizontal visibility map* of a simple polygon P is the set of horizontal chords associated with every vertex of P. Note that every vertex of P is associated with at most two chords, and the horizontal visibility map partitions P into a number of horizontal trapezoids (*horizontal trapezoidal decomposition* or *horizontal trapezoidation*). The *vertical visibility map* can be defined similarly.

The decomposition of P into PNHs starts with an arbitrary edge e of P as the bottom edge of the first PNH. The interior of the PNH refers to the part of P that is illuminated by the parallel light emanating into PNH perpendicular to e from its pseudobottom edge $e \cup e_s$, where e_s is the edge (if any) incident to e at an interior angle between 90° and 180°, i.e., the hypotenuse of the missing right-angle triangle. The boundary edges of the PNH that are not edges of P will be the bottom edges in the next step.

The decomposition of P can then be represented by a tree such that each tree node is a PNH and each tree edge represents the adjacency of two PNHs sharing a chord. Consider the example as given in Figure 3: PNH_1, with an edge of P as its bottom edge, is classified as the root. For each edge in PNH_1 which is not an edge of P, we regard it as the bottom edge for a son of PNH_1. PNH_2, PNH_3, and PNH_4 are sons of PNH_1 whose bottom edges are all horizontal, whereas PNH_2 is on one side facing PNH_3 and PNH_4, which are on the other. Similarly, the grandsons of PNH_1 are those with vertical bottom edges adjacent to sons of PNH_1, etc.

2.4. Merging the Voronoi diagrams of PNHs. It has been proved [13] that a Voronoi cell in $V_c(P)$ of a vertex in a PNH would not share any boundary edge with a Voronoi cell of a vertex in another PNH as long as these two vertices are not shared by these two PNHs, and these two PNHs are
(i) at the same depth not facing each other, or
(ii) with their corresponding depths at least two apart.

The Voronoi diagram of a PNH is first merged with the extended Voronoi diagram of its parent, then with those of its sons on one side, and finally with those of the remaining sons on the other. Condition (ii) ensures that the extended Voronoi diagram of its parent will not share any boundary with those of its sons, and thus, only those of

its neighbors (sons and parent) have to be considered in the construction of the part of the Voronoi diagram $V_c(P)$ in a PNH. Condition (i) ensures that merging the Voronoi diagram of a PNH with the extended ones of its sons will trace the bisectors between sites of the PNH at most twice (once for sites on each side) [10, 11, 14, 17, 19]. So, the merging of Voronoi diagrams at each PNH can be done in time linearly proportional to the total size of the PNH and all its sons. Thus, the Voronoi diagram $V_c(P)$ can be obtained in time linearly proportional to the size of P by merging the Voronoi diagram of each PNH together with the extended ones of its neighbors.

In order to find $V_c(P)$ in deterministic linear time, what remains to be solved is the efficient construction of the constrained Voronoi diagram of a PNH. In [13], a randomized algorithm is introduced to find the Voronoi diagram of a NH in expected linear time. The Voronoi diagram of the corresponding PNH can then be obtained by removing the bottom vertex from the Voronoi diagram of this NH, and this can be done in time linearly proportional to the size of the NH. In the next section, we shall concentrate our effort to design a linear-time deterministic algorithm for constructing the Voronoi diagram of a NH.

3. Finding the constrained Voronoi diagram of an NH. Given a normal histogram H with a horizontal bottom edge e, H is decomposed recursively into a tree, say T_I, of smaller normal histograms called *influence normal histograms* (INH), where a node of T_I corresponds to an INH and an edge of T_I indicates an adjacency between two INHs. A formal definition of INH with an algorithmic method of construction will be given in section 3.2. In Figure 5, node 0 (the root INH) is $(v_1, v_3', v_3, v_3'', v_5, v_6, v_7, v_8, v_9,\ v_{10}, v_{12}', v_{12}, v_{13}, v_{13}',\ v_{25}, v_{25}',\ v_{28}, v_{28}', v_{33}, v_{34}, v_{35})$.

Nodes 1–6 form the second level and are sons of node 0.

Node 1 $= (v_3', v_2, v_3)$,
Node 2 $= (v_3, v_4, v_3'')$,
Node 3 $= (v_{12}', v_{11}, v_{12})$,
Node 4 $= (v_{13}, v_{14}, v_{14}', v_{13}')$,
Node 5 $= (v_{25}, v_{26}, v_{27}, v_{25}')$, and
node 6 $= (v_{28}, v_{30}', v_{30}, v_{32}', v_{32}, v_{28}')$.

Nodes 7–9 form the third level with

Node 7 $= (v_{14}, v_{15}, v_{16}, v_{17}, v_{17}',\ v_{19}, v_{20},\ v_{21}, v_{22}, v_{23}, v_{24}, v_{14}')$,
Node 8 $= (v_{30}', v_{29}, v_{30})$,
Node 9 $= (v_{32}', v_{31}, v_{32})$.

Node 10 $= (v_{17}, v_{18}, v_{17}')$ is on the fourth level.

The decomposition ensures that the portion of Voronoi diagram $V_c(H)$ in each INH can only be affected by its own vertices and the vertices of its sons and nothing beyond. In general, the Voronoi cells of $V_c(H)$ associated with vertices of an INH might cross its bottom edge and share edges with Voronoi cells associated with vertices of its parent, but not with those of its brothers or its grandparents. Similarly, the Voronoi cells of an INH would not share any boundary with those of its grandsons. This property implies that, should the Voronoi diagrams of the INHs ($V_c(INH)$) be given, the repeated merging of the Voronoi diagrams of the adjacent INHs can be done in time linearly proportional to the sum of their sizes.

Let $V(p)$ denote the Voronoi cell associated with vertex p in a Voronoi diagram. A point p in a normal histogram H is called an *influence point* if the Voronoi cell $V(p)$ in $V_c(H \cup \{p\})$ will cross H's bottom edge e. The set of influence points is called the *influence region* (IR) with respect to bottom edge e. Consider Figure 5: the IR of H with respect to $\overline{v_1 v_{35}}$ (the bottom edge e) is the region enclosed by $\widehat{v_1 v_5}$, $\overline{v_5 v_6}$, $\overline{v_6 v_7}$, $\overline{v_7 v_8}$, $\overline{v_8 v_9}$, $\overline{v_9 v_{10}}$, $\widehat{v_{10} v_{25}}$, $\widehat{v_{25} v_{28}}$, $\widehat{v_{28} v_{34}}$, $\overline{v_{34} v_{35}}$, and $\overline{v_{35} v_1}$, where $\overline{xy}$ and $\widehat{xy}$ represent,

respectively, the straight line and the arc joining vertices x and y. The *root* (or *root INH*) of T_I is defined as the NH enclosing all influence points of H and consisting of all horizontal trapezoids that intersect the IR. In other words, the root INH would contain all its horizontal chords which will intersect the IR of H, i.e., the *smallest* NH containing IR in the sense that the INH is bound above by the lowest horizontal chords which do not intersect IR. As an example, the root INH is indicated by the unshaded region in Figure 5. Let us now consider the part of H excluding the root INH, which consists of zero or more disjoint polygons. Each polygon is also a NH with a chord as its bottom edge. As given in Figure 5, H is decomposed into a root INH and six other NHs, i.e., the NHs above chords $\overline{v_3'v_3}$, $\overline{v_3v_3''}$, $\overline{v_{12}'v_{12}}$, $\overline{v_{13}v_{13}'}$, $\overline{v_{25}v_{25}'}$, and $\overline{v_{28}v_{28}'}$. For example, the NH above chord $\overline{v_{28}v_{28}'}$ is $(v_{28}, v_{29}, v_{30}, v_{31}, v_{32}, v_{28}')$. The decomposition can be recursively applied to each of these NHs.

Since any node of T_I does not contain the influence points of its parent by the definition of INH, the Voronoi cell associated with a vertex in any node of T_I could not cross the bottom edge of its parent. Thus, the part of $V_c(H)$ within the root can be formed by merging the Voronoi diagram of the root INH with those of its sons. As the Voronoi cells associated with the internal vertices of an INH never share any edges with the Voronoi cells of its brother INH (Theorem 1), the merging can be performed in $O(m_0 + \sum_{i=1}^{s} m_i)$ time, where m_0 is the number of vertices of the root, s is the number of its sons, and m_i is the number of vertices of its ith son.

THEOREM 1. *Let v_1 be a vertex of INH_1 with bottom edge $\overline{u_1w_1}$ and v_2 be a vertex of INH_2 with bottom edge $\overline{u_2w_2}$. Assume that INH_1 and INH_2 are brothers in T_I, $v_1 \neq u_1$, $v_1 \neq w_1$, $v_2 \neq u_2$, and $v_2 \neq w_2$. Then, the Voronoi cell of v_1 will never share any point with the Voronoi cell of v_2.*

Proof. Without loss of generality, assume that $\overline{u_1w_1}$ is on the left-hand side of $\overline{u_2w_2}$; i.e., $u_1 < w_1 \leq u_2 < w_2$ according to their x-coordinates. We show that there does not exist a point p in H

(i) that is equidistant to v_1 and v_2, and
(ii) for which there exists no other vertex in H closer to p than v_1 and v_2.

Assume p exists. Since p is in H, p has to lie directly under $\overline{u_1w_1}$ in order to be closer to v_1 than to u_1 or w_1, i.e., $u_1 < p < w_1$. Similarly, p has to lie directly under $\overline{u_2w_2}$, i.e., $u_2 < p < w_2$. Obviously, p cannot simultaneously satisfy both conditions. □

For example, the Voronoi cell of v_{14} in INH_4 never shares any point with the Voronoi cell of v_{26} or v_{27} in INH_5. Let $M(n)$ denote the merging time for constructing $V_c(H)$ with $\mid H \mid = n$ when provided with the Voronoi diagram of every INH in T_I. Then, we have $M(n) = k(m_0 + \sum_{i=1}^{s} m_i) + \sum_{i=1}^{s} M(n_i)$, where k is a constant and n_i is the number of vertices of the ith subtree. As $n = m_0 + \sum_{i=1}^{s} n_i$, we can show that $M(n) = k(2n - m_0)$ by induction. Thus, the total merging time is $O(n)$. Note that in the above calculation, the pseudovertices are also counted. As n is at most thrice the actual number of vertices of H (as each vertex of H might associate with at most two pseudovertices), the total merging time is still linearly proportional to the actual number of vertices of H.

In the following sections, we shall prove the properties of the IR and the INH which allow us to do efficient merging and identification.

3.1. IR. Let H_V be a subpolygon of NH H, consisting of the bottom edge of H and all those vertices of H with the property that their associated Voronoi cells in $V_c(H)$ cross the bottom edge of H. H_V can also be viewed as the maximum subsequence of the vertices of H having this property. As H is a NH, H_V will also be a

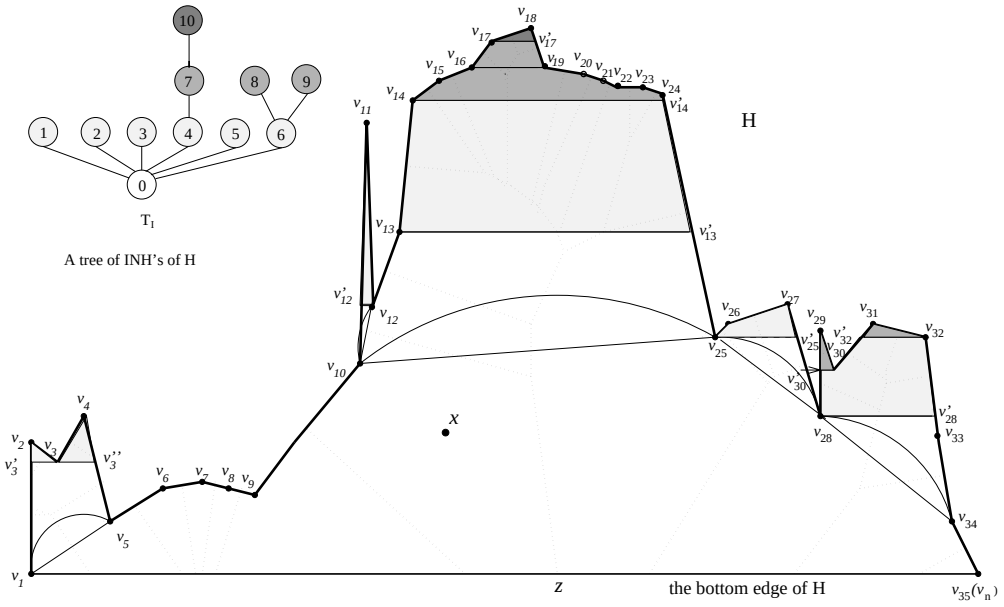


FIG. 5. *Decomposition of H into $INHs$ and T_I.*

NH sharing the same bottom edge as H. Let us consider the example given in Figure 5 again, in which H_V is indicated by the sequence of vertices $(v_1, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{25}, v_{28}, v_{34}, v_{35})$.

LEMMA 1. *All points in H_V are influence points.*

Proof. By the definition of H_V, the bisector of two adjacent vertices (except the two vertices of the bottom edge) of H_V, which forms part of the $V_c(H)$, always crosses the bottom edge of H. In other words, H_V are partitioned by these bisectors into cells, each of which is associated with one of its vertices. These cells resemble the Voronoi cells of $V_c(H)$. In fact, each of these cells in $V_c(H_V)$ always includes its corresponding Voronoi cell in $V_c(H)$. These bisectors also partition the bottom edge into segments according to their closest vertices in H or H_V. It is sufficient to prove this lemma by showing that given any point x in H_V, there always exists a point on the bottom edge which is closer to x than to any vertex in H; i.e., $V(x)$, the Voronoi cell of x, in $V_c(H \cup \{x\})$ would cross the bottom edge. Let x be a point in H_V, in particular, in a Voronoi cell $V(u)$, corresponding to vertex u in $V_c(H_V)$. Furthermore, let the extended line of $\overline{ux}$ intersect the boundary of this Voronoi cell $V(u)$ at y, which may be a point on the bottom edge or a point on a bisector. If y is on the bottom edge, let z be y; otherwise let z be the intersection point of that bisector and the bottom edge. An example is given in Figure 5, where x is in $V(v_{10})$, i.e., $u = v_{10}$. As $\angle uxz > 90°$, z is always closer to x than to u by the triangular property. It is easy to see that z is closer to x than to any other vertices in H, thus z is a point on the bottom edge that belongs to $V(x)$; i.e., $V(x)$ crosses the bottom edge. Hence, x belongs to the IR. □

In general, the IR includes some regions not belonging to H_V. Let b be a boundary edge of H_V. If b is also an edge of H, then b must be an edge of the IR, e.g., $\overline{v_5v_6}$, $\overline{v_6v_7}$, $\overline{v_7v_8}$, etc. in Figure 5. However, if $b = \overline{uw}$ is a diagonal of H, then the IR must include some region of H above b and below the circular arc $\widehat{uw}$, where $\widehat{uw}$ is part of

the semicircle above the bottom edge. (The semicircle is uniquely defined by boundary points u and w and center c, where c is the intersection point of the bottom edge and the perpendicular bisector of u and w.) Let O_{uw} denote the region in H above b (i.e., outside H_V) and below the circular arc $\widehat{uw}$, and let $O_{v_1v_5}$, $O_{v_{10}v_{25}}$, $O_{v_{25}v_{28}}$, and $O_{v_{28}v_{34}}$ be such examples in Figure 5. Then we have the following theorem.

THEOREM 2. *$IR = (\cup_{\overline{uw}\in D} O_{uw}) \cup H_V$, where D is the set of edges of H_V which are diagonals of H.*

Proof. By Lemma 1, we need to prove $IR - H_V = \cup_{\overline{uw}\in D} O_{uw}$ (as $H_V \cap O_{uw} = \phi$). Consider a point p in H, but not in H_V. Then, point p must lie above an edge $\overline{uw}$ of H_V which is a diagonal of H. On one hand, if point $p \in O_{uw}$, then b_{up} and b_{pw} will cross the bottom edge before intersecting each other, where b_{xy} denotes the perpendicular bisector of vertices x and y, and thus p belongs to the IR; i.e., $IR - H_V \supseteq \cup_{\overline{uw}\in D} O_{uw}$. On the other hand, if $p \notin O_{uw}$, then b_{up} and b_{pw} will intersect each other above the bottom edge, and thus p does not belong to the IR; i.e., $IR - H_V \subseteq \cup_{\overline{uw}\in D} O_{uw}$. □

COROLLARY. *Assume a NH H and let $H_V = (v_0, v_1, \ldots, v_n)$. Then the IR with respect to H can be defined by keeping the sequence of vertices of H_V and by replacing all diagonals $\overline{v_iv_{i+1}}$ of H in the sequence of H_V by an arc $\widehat{v_iv_{i+1}}$.* □

3.2. INH. As the root INH is the smallest NH containing the IR, an INH would contain all the edges of the IR, in particular, those edges of H_V (Theorem 2) which are also edges of H (e.g., $\overline{v_5v_6}$, $\overline{v_6v_7}$, $\overline{v_7v_8}$, etc. in Figure 5). As O_{uw} is part of the IR for every $\overline{uw} \in D$ (Theorem 2), the remaining edges of an INH above $\overline{uw}$ would be those chords and edges of H enclosing O_{uw}. Thus, we define H_B *above* $\overline{uw}$ as the smallest NH containing O_{uw}, which consists of edges (or parts of edges) of H and the lowest horizontal chord which do not intersect the IR. For example, as in Figure 5, the H_B's are $(v_1, v_3', v_3, v_3'', v_5)$, $(v_{10}, v_{12}', v_{12}, v_{13}, v_{13}', v_{25})$, $(v_{25}, v_{25}', v_{28})$, and $(v_{28}, v_{28}', v_{33}, v_{34})$.

Now, we can have a precise description of an INH. There are two types of vertices in an INH, the vertices of H_V and the vertices of H_B's, with one H_B for each edge in D. Thus, any vertex in an INH that is not in H_V will be in H_B, and the endpoints of any edge in D will be vertices in both H_V and H_B. In the following, we shall describe the properties of H_B and H_V and show that the Voronoi diagram of an INH can be constructed in linear time.

A *monotonic histogram* is an NH such that if the bottom edge is on the x-axis, then the x-coordinates of the vertices along the boundary are monotonically nondecreasing, and the y-coordinates of the vertices (except the last vertex) along the boundary are monotonically nondecreasing or nonincreasing. A *bitonic histogram* is a composition of two monotone histograms such that the x-coordinates of the vertices along the boundary are monotonically nondecreasing, and the y-coordinates of the vertices along the boundary are first monotonically nondecreasing on one side and then monotonically nonincreasing on the other.

LEMMA 2. *H_B is bitonic.*

Proof. Since H_B is the smallest NH enclosing O_{uw}, all its internal horizontal chords will intersect with O_{uw}; i.e., all vertices of H_B, except possibly the top vertex and its associated pseudovertex (vertices), should be horizontally visible from O_{uw}. As H_B consists of only edges (or parts of edges) and chords of H, all edges of H_B should be monotonically nondecreasing in the x- and y-coordinates on one side and monotonically nondecreasing in the x-coordinate but monotonically nonincreasing in the y-coordinate on the other. Thus, H_B is bitonic. □

LEMMA 3. *The Voronoi diagrams of H_B and H_V can be constructed in linear time.*

Proof. It is shown in [8] that the Voronoi diagram of a monotonic histogram can be constructed in linear time. Because H_B can be partitioned into two monotonic histograms by the vertical line through its highest vertex or edge (Lemma 2), the Voronoi diagrams of two such monotonic polygons can be merged in linear time [20, 13]. Thus, $V_c(H_B)$ can be found in linear time. As far as H_V is concerned, because the vertices on the boundary of H_V are in sorted order according to their x-coordinates (property of the NH), the extended Voronoi diagram below the bottom edge can be found in linear time [2]. Since all the Voronoi cells in H_V must cross the bottom edge, the Voronoi diagram of H_V can be constructed in linear time from its extended Voronoi diagram below the bottom edge. □

Note that in the construction of the Voronoi diagrams of H_B and H_V, all the pseudovertices are ignored. Thus, the resulting Voronoi diagrams do not contain any Voronoi cell of pseudovertices. This approach is different from that proposed in [13], which requires the removal of the Voronoi cells of pseudovertices.

The following lemma shows that the Voronoi diagrams of two H_B's cannot affect each other.

LEMMA 4. *Assume an INH with its attached H_B's, and let x and y be two vertices not belonging to H_V but in two different H_B's. Then the Voronoi cells, $V(x)$ and $V(y)$, cannot share any point in $V_c(H_V)$.*

Proof. As x and y are vertices in two different H_B's but not in H_V, x and y must be separated by some vertex z in H_V. By the definition of H_V, the Voronoi cell $V(z)$ must cross the bottom edge. Thus, $V(x)$ cannot share any point with $V(y)$ above the bottom edge. □

THEOREM 3. *The Voronoi diagram of an INH can be constructed in time linearly proportional to its size.*

Proof. By Lemma 3, the Voronoi diagrams of H_V and H_B's can be constructed in time linearly proportional to their sizes. Since each H_B shares an edge with H_V, the Voronoi diagrams of each H_B and H_V can be merged in time proportional to the number of Voronoi edges shared by them [20, 13]. As different H_B's do not interfere with each other (Lemma 4), the total merging time is linearly proportional to the number of Voronoi edges shared by H_B's and H_V, i.e., the size of the INH. □

3.3. Region identification. In this section, we shall present an algorithm which identifies the INH in a NH in time linearly proportional to the size of the INH. Chazelle's linear-time algorithm [5] is first applied to the NH to obtain its horizontal visibility map (Figure 6). By the property of a normal histogram, H can be further represented by a *partition tree* T_P, in which each tree node represents a chord in the map and each tree edge represents the adjacency of two chords. Let $n(v)$ denote the chord(s) associated with vertex v of H. If there are two chords in $n(v)$, $n^L(v)$ and $n^R(v)$ denote the left chord and right chord, respectively. With this partition tree T_P, the INH to be identified can be represented as a rooted subtree of T_P.[4] For example, the INH indicated by the shaded area can be represented by the rooted subtree as marked in Figure 6. The algorithm to identify the INH is based on tree traversal. In order to achieve linear time complexity, only those tree nodes relevant to the INH will be traversed. Thus, one of the key steps in the tree traversal is the pruning condition, i.e., under what conditions the traversal of a subtree can be terminated. The other key step is the identification of the vertices of H_V so that we can partition the INH

[4] A rooted subtree of T has the property that the root of T is also the root of the subtree.

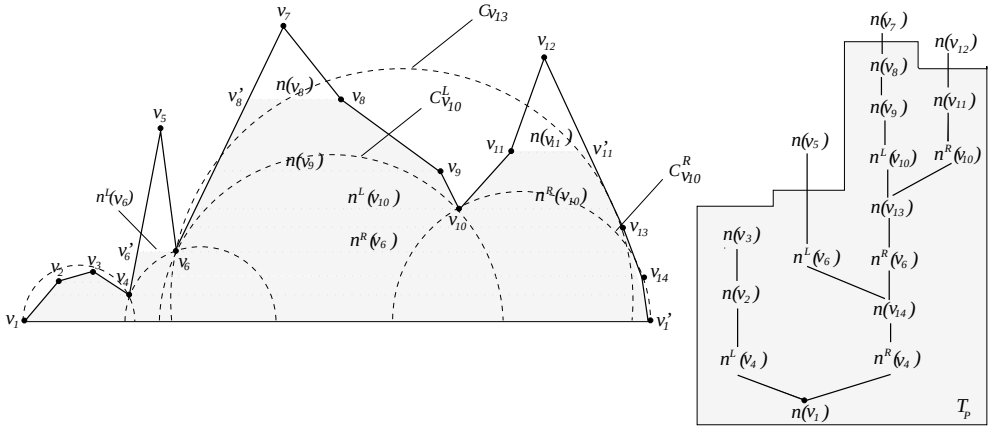


FIG. 6. *An INH and its tree* T_P.

into H_B's and H_V for further constructing of Voronoi diagrams, as described in the previous section.

The following lemma gives a necessary and sufficient condition for a vertex v of H to be a vertex of H_V. Based on Theorem 2 and the definition of H_B, we can ensure that a visited vertex, that is not a vertex of H_V, will be a vertex of H_B.

LEMMA 5. *For any vertex v of H, v is a vertex of H_V if and only if v can be touched by a circle centered at the bottom edge and empty of other vertices of H.*

Proof. The center of such a circle is a point closer to vertex v than to other vertices of H. As Voronoi cells are simply connected, the Voronoi cell $V(v)$ will intersect or cross the bottom edge. The lemma follows directly from the definition of H_V. □

Without loss of generality, assume the parent of $n(v)$ intersects the IR and $n(v)$ is being visited on the traversal of T_P. Based on Lemma 5, vertex v is tested and classified into one of the following three types: (i) a vertex in H_V, (ii) a vertex in H_B (if not in H_V), or (iii) a potential vertex.

A *potential vertex* is such a vertex that can be touched by a circle centered at the bottom edge and empty of any vertices of H *on or below* $n(v)$; i.e., the Voronoi cell of a potential vertex would extend across the bottom edge if no vertex above $n(v)$ will affect this Voronoi cell. However, since we have not examined any vertex above $n(v)$ yet, we cannot rule out the possibility that v is in H_B.

Assume that all circles in the following discussion will be centered at the bottom edge. Let C_v denote the largest circle that crosses chord $n(v)$ and whose interior does not contain any vertex on or below $n(v)$. If vertex v is associated with two chords, then C_v^L and C_v^R denote such circles that cross $n^L(v)$ and $n^R(v)$, respectively. Let us first study some properties of C_v, which can be used to determine the IR above $n(v)$ by considering only the histogram studied so far, i.e., the histogram below $n(v)$ (note that C_v^L and C_v^R also have these properties).

LEMMA 6. *If C_v exists, then*

(i) *C_v must touch vertices to the left and right of its centers, or is centered at a nonvertex endpoint of the bottom edge, i.e., pseudovertex, and*

(ii) *if v is the left (right) endpoint of $n(v)$, then the center of C_v must be to the right (left) of v.*

Proof. (i) If C_v exists, then C_v is unique. This is because, by the x-monotonic property of H, there is no vertex below the bottom edge (e) and between the intervals

of $n(v)$; i.e., the largest circle centered at e cannot be bounded by such vertices. Then, either the largest circle centered at e must touch only one vertex above e and hence will be centered at the (nonvertex) endpoint of e, or this circle must touch two or more vertices above e on both sides of its center. In either case, this largest circle is the C_v if it crosses $n(v)$. (ii) By contradiction, assume that the center of C_v is to the left of v, which is the left endpoint of $n(v)$; then C_v would not cross $n(v)$ and cannot exist. □

The following lemma will give a sufficient condition for a vertex to be in H_V.

LEMMA 7.

(i) *If both C_v^L and C_v^R exist, then vertex v must be in H_V.*

(ii) *If C_v^L (C_v^R) does not exist, then the traversal of the subtree above $n^L(v)$ ($n^R(v)$) is terminated.*

Proof. (i) The existence of both C_v^L and C_v^R implies the existence of an empty circle with center at the bottom edge and lying entirely below $n(v)$. Thus v must be in H_V by Lemma 5. (ii) Since there does not exist any vertex above $n^L(v)$ that can affect the IR, the tree traversal can be terminated. □

Let us consider an example as given in Figure 6 to show how these properties can be applied to classify v. Initially C_{v_1} is the circle that touches v_1 and is centered at v_1'. The next vertex to be visited is v_4. Since both $C_{v_4}^L$ and $C_{v_4}^R$ exist, v_4 is in H_V (Lemma 7). When the tree traversal of T_P at $n^R(v_4)$ is continued, vertices v_{14}, v_6, and v_{13} will be visited and classified as potential vertices.

In order to construct C_v for each v during the tree traversal T_P, the potential vertices previously identified are kept in two stacks, L and R. Except possibly for their bottom vertices, L contains the "left" potential vertices (i.e., left endpoints of the corresponding chords) while R contains the "right" ones. For example, after v_6 and v_{13} are visited, L and R contain $[v_4, v_6)$ and $(v_{13}, v_{14}]$, respectively, with v_6 and v_{13} being their top vertices. The following lemma gives the properties of stacks L and R.

LEMMA 8. *With respect to the histogram on or below the chord $n(v)$ studied so far,* (i) *stacks L and R contain the vertices whose Voronoi cells cross the bottom edge in order, and* (ii) *the largest empty circle C_v is determined by the top vertex of L and the top vertex of R.*

Proof. (i) First, we have to show that the y-coordinates of the vertices in L are monotonically increasing while those in R are monotonically decreasing. Without loss of generality, assume the contrary, that L contains a vertex v whose y-coordinate is lower than that of its precedent vertex t. Then it is impossible for t to be the left endpoint of a chord. As the vertices in L and R are potential vertices visited according to their y-coordinates, their Voronoi cells must cross the bottom edge in order.

(ii) As L and R contain the "left" and "right" potential vertices (i.e., left and right endpoints of the chords), the largest circle C_v must touch the top vertices of L and R, which are the rightmost left endpoint and the leftmost right endpoints, respectively. □

We shall describe the construction of $C_{v'}$ and the tree traversal algorithm of T_P. Assume v' is the next vertex to be visited after v.

Case 1. $n(v')$ does not exist or does not intersect C_v; the tree traversal is terminated/pruned at $n(v')$ and all vertices in L and R become vertices in H_V. Subtrees rooted at such $n(v')$ (if they exist) are pruned because their corresponding INHs do not contain the IR. The pruned subtrees represent smaller NHs needed to be processed recursively.

For example, in Figure 6, the tree traversal is terminated at $n(v_8)$ and $n(v_{11})$ as they do not intersect C_{v_9} and $C^R_{v_{10}}$, respectively. Vertex v_6, previously in L, becomes a vertex in H_V. In Figure 5 the pruned portion $(v_{28}, v_{29}, v_{30}, v_{31}, v_{32}, v_{28'})$ is an NH with $\overline{v_{28}v_{28'}}$ (the pruned chord) as the bottom edge to be processed recursively.

Case 2. $n(v')$ intersects C_v; the tree traversal will continue to visit $n(v')$'s son(s).

(a) v' is outside C_v (i.e., $b_{uv'}$ and $b_{v'w}$ intersect each other above the bottom edge where u and w are top vertices of L and R, respectively). $- v'$ will not be in H_V and must be in H_B; stacks L and R remain unchanged. Vertex v_9 in Figure 6 is such an example.

(b) v' is inside C_v (i.e., $b_{uv'}$ and $b_{v'w}$ cross the bottom edge before intersecting each other), and $- v'$ may be closer to some point of the bottom edge than vertices in L and R. The Voronoi cell of v' crosses the bottom edge and may crowd out the Voronoi cells of some vertices in L and R. If the largest circle determined by the next-to-top vertex of L and v' does not contain the top vertex of L (i.e., $b_{u'u}$ intersects $b_{uv'}$ above the bottom edge where u' is the next-to-top vertex of L), then pop the top vertex of L and assign it as a vertex in H_B. Vertices of L are popped until its top vertex remains in L; Lemma 8 guarantees that all vertices beneath also remain in L. Stack R is handled similarly. For example, in Figure 6, v_{13} is popped when v_{10} is visited.

  (i) If either $C^L_{v'}$ or $C^R_{v'}$ exists but not both, v' is pushed onto stack $L(R)$ if v' is the left (right) endpoint of that chord. The tree traversal is continued at $n(v')$ using the new empty circle $C_{v'}$, new stack $L(R)$, and old stack $R(L)$. Vertices v_6 and v_{13} are such examples which are pushed onto their corresponding stacks L and R when $n(v_6)$ and $n(v_{13})$ are visited.

  (ii) If both $C^L_{v'}$ and $C^R_{v'}$ exist, v' must be a vertex in H_V by Lemma 7. The tree traversal will continue at $n^L(v')$, where $C^L_{v'}$, old stack L, and new stack R' containing v' alone will be used for further vertex classification. Similarly, the traversal at $n^R(v')$ will use $C^R_{v'}$, old stack R, and a new stack L' containing only v'. Vertex v_{10} in Figure 6 is such an example.

For visualizing the above algorithm, Figure 7 gives a walk-through of the example in Figure 6.

3.4. Complexity analysis. Our method for constructing the constrained Voronoi diagram of a simple polygon P mainly relies on the efficiency of the identification of the INHs from an NH. Since the identification for different INHs is executed recursively, we shall only consider the root INH of an NH.

As described previously, when we traverse tree T_P of an NH to identify an INH, we visit each vertex of the INH exactly once. Those vertices which have not been visited in the traversal of T_P cannot belong to the root INH. Therefore, we only need to show that each visited vertex is tested in constant time in order to classify it as a vertex in H_V or in H_B.

Let us consider a vertex v. In the test, v can be classified into one of the following three types: (i) $v \epsilon H_V$, (ii) $v \epsilon H_B$, and (iii) v is a potential vertex.

For type (i), v is stored in the list of vertices representing H_V.

For type (ii), v is stored in the list of vertices corresponding to a particular H_B and vertex v will never be tested again. Note that each vertex in H_V or potential vertex is associated with a separate list H_B of vertices. If the potential vertex, separating the two lists of vertices corresponding to two H_B's, has been determined to be in H_B, then these two lists of vertices will be concatenated together.

For type (iii), v is stored in the left or right stack and could be repeatedly tested

Initially $L = [v_1)$, $R = (\ \ ]$, $H_V = (v_1)$
visiting v_4: insert v_4 into H_V $\quad / * case2b(ii) * /$
left chord $n^L(v_4)$, $L = [v_1)$, $R = (v_4]$
visiting v_2: chord $n(v_2)$, $L = [v_1, v_2)$, $R = (v_4]$ $\quad / * case2b(i) * /$
visiting v_3: traversal terminated, insert v_2, v_3 into H_V $\quad / * case1 * /$
right chord $n^R(v_4)$, $L = [v_4)$, $R = (\ \ ]$
visiting v_{14}: chord $n(v_{14})$, $L = [v_4)$, $R = (v_{14}]$ $\quad / * case2b(i) * /$
visiting v_6: $\quad / * case2b(i) * /$
left chord $n^L(v_6)$, traversal pruned, process NH above $n^L(v_6)$ recursively
$/ * case1 * /$
right chord $n^R(v_6)$, $L = [v_4, v_6)$, $R = (v_{14}]$
visiting v_{13}: chord $n(v_{13})$, $L = [v_4, v_6)$, $R = (v_{13}, v_{14}]$ $\quad / * case2b(i) * /$
visiting v_{10}: insert v_{10} in H_V $\quad / * case2b(ii) * /$
left chord $n^L(v_{10})$, $L = [v_4, v_6)$, $R = (v_{10}]$
visiting v_9: $H_B(-, v_{10}) = (v_9, v_{10})$ $\quad / * case2a * /$
visiting v_8: traversal pruned, insert v_6 into H_V $\quad / * case1 * /$
$H_B(v_6, v_{10})$ = concatenate (v_6, v'_8, v_8) and $H_B(-, v_{10}))$
$= (v_6, v'_8, v_8, v_9, v_{10})$
process NH above $n(v_8)$ recursively
right chord $n^R(v_{10})$, $L = [v_{10})$, pop v_{13} from R and assign v_{13}
as an element in H_B,
$R = (v_{14}]$, H_B(-, v_{14})=(v_{13}, v_{14})
visiting v_{11}: traversal pruned, $\quad / * case1 * /$
$H_B(v_{10}, v_{14})$ = concatenate $(v_{10}, v_{11}, v'_{11})$ and $H_B(-, v_{14}))$
$= (v_{10}, v_{11}, v'_{11}, v_{13}, v_{14})$
process NH above $n(v_{11})$ recursively
Finally $H_V = (v_1, v_2, v_3, v_4, v_6, v_{10}, v_{14})$

FIG. 7. *Walk-through of the example in Figure* 6.

when the descendants of v are visited. However, once vertex v is identified to be a vertex in H_V or H_B, v will never be tested again. Thus, we can argue that the time for visiting a vertex is constant when amortized over a sequence of tests. To see this, our analysis assumes that one unit credit should have been assigned to each potential vertex in L and R. For each vertex v of H in the bottom-up sweep of T_P, two unit credits of work are needed for each test: one for carrying the test itself, i.e., either assigning v as a vertex in H_V, H_B or a potential vertex in L or R; the other unit credit is assigned to the vertex should it be identified as a potential vertex. The test on a vertex in L or R to determine whether or not it has to be reassigned to H_B or H_V will be paid by the unit credit associated with the vertex. This either happens once for the checked vertex (which is accounted to it) or this test stops at a vertex which still cannot be reassigned and the test stop condition is accounted to the vertex again.

It is not difficult to see that linked lists can be used to keep track of the vertices in H_V and H_B's. In particular, insertion and concatenation operations on H_V and H_B's can be executed in constant time. The time complexity analysis for the construction of Voronoi diagrams of INH, NH, and P is obvious, as described in the previous sections. We shall conclude the above analysis by the following theorem.

THEOREM 4. *$CDT(P)$ can be found in $\Theta(|\ P\ |)$ time for simple polygon P.*

4. Concluding remarks. In this paper, we presented a deterministic algorithm for finding the constrained Delaunay triangulation of a simple polygon with n sides in $\Theta(n)$ time in the worst case. This may be one of the few linear-time algorithms for nonarbitrary triangulation of a simple polygon.

In the definition of Delaunay triangulation, we can check whether a triangulation is Delaunay by studying vertices within local proximity. It should not be surprising that the Delaunay triangulation and the constrained Voronoi diagram of a simple polygon can be done in linear time after being given Chazelle's horizontal visibility map, which links vertices within proximity together. The horizontal visibility maps are helpful to decompose the polygon into components such that the "divide and conquer" approach can be applied. However, if the decomposition of the polygon into components is not carefully done, "interaction" of the Voronoi diagrams of the components may be more than linear (even quadratic time). From Theorem 1, the partition of the polygon into components by chords has the advantage that the Voronoi diagrams of the components at the same level would not interact with each other; i.e., horizontal interaction can be reduced. Moreover, because of the property of H_V, interaction of Voronoi diagrams of components at different levels can also be confined; i.e., vertical interaction can be eliminated.

With our linear-time algorithm, the following related problems can also be solved efficiently:

(1) all nearest (mutual visible) neighbors of the vertices of a simple polygon [13],

(2) a shortest diagonal of a simple polygon [13],

(3) a largest inscribing circle of vertices of a simple polygon [12],

(4) the nearest vertex from a query point [13],

(5) finding $DT(S)$ if the Euclidean minimum spanning tree for a point set S is given [1],

(6) finding standard Voronoi diagram for S' if the Voronoi diagram of a point set S is known [1], where $S' \subset S$.

By treating edges and vertices of a single polygon as sites for the Voronoi diagram, we can apply ideas similar to those given in this paper to find the medial axis of a simple polygon in linear time [7].

Acknowledgment. The authors would like to thank Bethany Chan, Siu-Wing Cheng, and the anonymous referees for their patience in reading the first draft of this paper and their comments in improving the readability of the paper.

REFERENCES

[1] A. Aggarwal (1988), *Computational Geometry*, MIT Lecture Notes 18.409, MIT, Cambridge, MA.

[2] A. Aggarwal, L. Guibas, J. Saxe, and P. Shor (1989), *A linear time algorithm for computing the Voronoi diagram of a convex polygon*, Discrete Comput. Geom., 4, pp. 591–604.

[3] A. Aurenhammer (1991), *Voronoi diagrams: A survey*, ACM Computing Surveys, 23, pp. 345-405.

[4] M. Bern and D. Eppstein (1992), *Mesh Generation and Optimal Triangulation*, Technical Report, Xerox PARC, Palo Alto, CA.

[5] B. Chazelle (1991), *Triangulating a simple polygon in linear time*, Discrete Comput. Geom., 6, pp. 485–524.

[6] P. Chew (1987), *Constrained Delaunay triangulation*, in Proc. 3rd ACM Symposium on Comp. Geometry, pp. 213–222.

[7] F. Chin, J. Snoeyink, and C. A. Wang (1995), *Finding the medial axis of a simple polygon in linear time*, in Proc. 6th International Symposium ($ISAAC'95$), Lecture Notes in Comput. Sci. 1004, Springer-Verlag, New York, pp. 382–391.

[8] H. Djidjev and A. Lingas (1991), *On computing the Voronoi diagram for restricted planar figures*, in Lecture Notes in Comput. Sci. 519, Springer-Verlag, New York, pp. 54–64.
[9] B. Joe and C. Wang (1993), *Duality of constrained Delaunay triangulation and Voronoi diagram*, Algorithmica, 9, pp. 142–155.
[10] D.G. Kirkpatrick (1979), *Efficient computation of continuous skeletons*, in Proc. 20th IEEE Symposium on Foundations of Computer Science, pp. 18–27.
[11] R. Klein (1989), *Concrete and Abstract Voronoi Diagrams*, Lecture Notes in Comput. Sci. 400 Springer-Verlag, New York.
[12] R. Klein and A. Lingas (1992), *A linear time algorithm for the bounded Voronoi diagram of a simple polygon in L_1 metrics*, in Proc. 8th ACM Symposium on Comp. Geometry, pp. 124–133.
[13] R. Klein and A. Lingas (1993), *A linear time randomized algorithm for the bounded Voronoi diagram of a simple polygon*, in Proc. 9th ACM Symposium on Comp. Geometry, pp. 124–133.
[14] D. T. Lee (1982), *Medial axis transformation of a planar shape*, IEEE Trans. Pat. Anal. Mach. Int., PAMI-4(4), pp. 363–369.
[15] D. Lee and A. Lin (1986), *Generalized Delaunay triangulations for planar graphs*, Discrete Comput. Geom., 1, pp. 201–217.
[16] A. Lingas (1987), *A space efficient algorithm for the constrained Delaunay triangulation*, in Lecture Notes in Control and Inform. Sci. 113, Springer-Verlag, New York, pp. 359–364.
[17] F. P. Preparata and M. I. Shamos (1985), *Computational Geometry - An Introduction*, Springer-Verlag, New York.
[18] R. Seidel (1988), *Constrained Delaunay Triangulations and Voronoi Diagrams with Obstacles*, Rep. 260, IIG-TU Graz, Austria, pp. 178–191.
[19] M. I. Shamos and D. Hoey (1975), *Closest point problems*, in Proc. 16th IEEE Symposium on the Foundations of Computer Science, pp. 151–162.
[20] C. Wang (1993), *Efficiently updating the constrained Delaunay triangulations*, BIT, 33, pp. 176–181.
[21] C. Wang and L. Schubert (1987), *An optimal algorithm for constructing the Delaunay triangulation of a set of line segments*, in Proc. 3rd ACM Symposium on Comp. Geometry, pp. 223–232.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 487–510

RECONSTRUCTING ALGEBRAIC FUNCTIONS FROM MIXED DATA*

SIGAL AR†, RICHARD J. LIPTON†, RONITT RUBINFELD‡, AND MADHU SUDAN§

Abstract. We consider a variant of the traditional task of explicitly reconstructing algebraic functions from black box representations. In the traditional setting for such problems, one is given access to an unknown function f that is represented by a black box, or an oracle, which can be queried for the value of f at any input. Given a guarantee that this unknown function f is some nice algebraic function, say a polynomial in its input of degree bound d, the goal of the reconstruction problem is to explicitly determine the coefficients of the unknown polynomial. All work on polynomial interpolation, especially sparse ones, are or may be presented in such a setting. The work of Kaltofen and Trager [*Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators*, in Proc. 29th Ann. IEEE Symp. on Foundations of Computer Science, 1988, pp. 296–305], for instance, highlights the utility of this setting, by performing numerous manipulations on polynomials presented as black boxes.

The variant considered in this paper differs from the traditional setting in that our black boxes represent several algebraic functions $f_1, \ldots, f_k$, where at each input x, the box arbitrarily chooses a subset of $f_1(x), \ldots, f_k(x)$ to output and we do not know which subset it outputs. We show how to reconstruct the functions $f_1, \ldots, f_k$ from the black box, provided the black box outputs according to these functions "often." This allows us to *group* the sample points into sets, such that for each set, all outputs to points in the set are from the same algebraic function. Our methods are robust in the presence of a small fraction of arbitrary errors in the black box.

Our model and techniques can be applied in the areas of computer vision, machine learning, curve fitting and polynomial approximation, self-correcting programs, and bivariate polynomial factorization.

Key words. Bezout's theorem, error correcting codes, multivariate polynomials, PAC learning, noisy interpolation, polynomial factoring, polynomial interpolation

AMS subject classifications. 68Q25, 68Q40, 68Q60

PII. S0097539796297577

1. Introduction.

Suppose you are given a large set of points in the plane and you are told that an overwhelming majority of these points lie on one of k different algebraic curves of some specified degree bound D (but you are not told anything else about the curves). Given the parameters k and D, your task is to determine or "reconstruct" these algebraic curves, or alternatively, to *group* the points into sets, each of which is on the same degree D curve. Related versions of this problem may also be of interest, such as extensions to higher dimensions, and a setting where instead of the points being given in advance, one is allowed to make queries of the form "what

*Received by the editors January 22, 1996; accepted for publication (in revised form) November 18, 1996; published electronically July 28, 1998.

http://www.siam.org/journals/sicomp/28-2/29757.html

†Department of Computer Science, Princeton University, Princeton, NJ 08544 (shr@cs.princeton.edu, rjl@cs.princeton.edu). The research of the first author was supported by Dept. of Navy grant N00014-85-C-0456, NSF PYI grant CCR-9057486, and a grant from MITL. Part of this research was done while the second author was at Matsushita Information Technology Labs.

‡Department of Computer Science, Cornell University, 5137 Upson Hall, Ithaca, NY 14853 (ronitt@cs.cornell.edu). Part of this research was done while the author was at Hebrew University and Princeton University. This research was supported by DIMACS, NSF-STC88-09648, ONR Young Investigator award N00014-93-1-0590, and grant 92-00226 from the United States–Israel Binational Science Foundation (BSF).

§Laboratory for Computer Science, MIT, Cambridge, MA 02139 (madhu@lcs.mit.edu). Part of this work was done while this author was at U.C. Berkeley and IBM's Thomas J. Watson Research Center, supported in part by NSF grant CCR 88-96202.

is the value of one of the curves at point x?" (The answer to such a query will not specify which of the k curves was used to compute the value.)

Solutions to this fundamental problem have applications to:

- the grouping problem in computer vision,
- computational learning theory,
- curve fitting over discrete domains,
- simple algorithms for polynomial factorization,
- self-correcting programs.

Computer vision. Consider a computer vision system for a robot that picks parts out of a bin. The input to the system contains an intensity map of the scene. The robot can distinguish between the parts by extracting edges from the image. Current edge detection algorithms use discretized differential operators to extract edges (e.g., [30], [10]). These algorithms produce output consisting of a bit map, where for every image point (x, y), the bit value of the point, $e(x, y)$, is set to 1 if and only if this point lies on an edge. For many vision applications it is then desired to connect between neighboring points to achieve a more compact representation of the edge map. This problem, known as "the grouping problem," is complicated by the fact that the parts are cluttered, they may be nonconvex, and they may contain holes. No polynomial time algorithm has been found for this problem.

Under the assumption that the edges of the parts are given by piecewise algebraic curves, and that the edge detection process produces results which are free of precision error, our algorithm transforms edge maps into piecewise polynomial curves in polynomial time. The second assumption is unrealistic in real computer vision applications. However, we feel that it suggests an interesting approach which should be studied further.

Computational learning theory. Our mechanism can be used to extend some well-known results on learning boolean functions to the setting of learning real-valued functions. Here is a specific instance of such a situation: in the study of economics, the price–demand curve is often considered to be well described by an algebraic function (e.g., $f(x) = c/x$ or $f(x) = -a \cdot x + b$). However, it is also the case that this curve may change [23]. In particular, there may be several unknown price–demand curves which apply in various situations: one may correspond to the behavior found when the country is at war, a second may apply after a stock market crash, and a third behavior may be found after a change in the tax structure. Some of the factors that determine which curve applies may be obvious, but others may occur because of more subtle reasons. The task of learning the price–demand relationship may be decomposed into the two subtasks of first determining the unknown curves, and then learning what determines the move from one curve to another. Our algorithm gives a solution for the first task.

We consider the Valiant model of PAC learning [36], in which a concept is learnable if there is an efficient algorithm that is able to find a good approximation to the concept on the basis of sample data. In general, our results imply that any function on input x and boolean attributes $(y_1, \ldots, y_m)$ which uses $(y_1, \ldots, y_m)$ to select f_i from a set of polynomial functions $f_1, \ldots, f_k$ and then computes and outputs $f_i(x)$ can be learned, as long as the selector function can be learned.

For example, a *polynomial-valued decision list* given by a list of terms (conjuncts of literals) $(D_1, \ldots, D_k)$ over boolean variables $y_1, \ldots, y_n$, and a list of univariate polynomials $(f_1, \ldots, f_{k+1})$ in a real variable x, represents a real-valued function f as follows:

$$f(x, y_1, \ldots, y_n) = f_i(x),$$

where i is the least index such that $D_i(y_1, \ldots, y_n)$ is true.

If the terms are restricted to being conjunctions of at most c literals, we call it a *polynomial-valued c-decision list*. This is an extension of the *boolean decision list* model defined by Rivest in [32], where the polynomials f_i are restricted to being the constants 0 or 1.

In [32], Rivest shows that the class of boolean c-decision lists is learnable in polynomial time. Using our techniques, in combination with Rivest's algorithm, we can extend this result to show that the class of polynomial-valued c-decision lists can be learned in polynomial time. The only technical point that needs to be made is as follows: Rivest gives an algorithm for producing a decision list that is consistent with the random examples and then argues using an Occam argument (see Blumer et al. [8]) that any hypothesis that is consistent with the labels of the random examples is a good hypothesis (i.e., computes a function that is usually equal to the target function). Our techniques in combination with Rivest's algorithm yield a consistent hypothesis, but since our hypothesis is not a boolean function, we must use the work of Haussler [22] to see that a consistent hypothesis is a good hypothesis.

Independent of our work, Blum and Chalasani [6] also consider a model of learning from examples where the examples may be classified according to one of several different concepts. In their model an adversary controls the decision of which concept would be used to classify the next example. Under this model they study the task of learning boolean-valued concepts such as k-term DNFs and probabilistic decision lists.

Curve fitting problems over discrete domains. A typical curve fitting problem takes the following form: given a set of points $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ on the plane, give a simple curve that "fits" the given points. Depending on the exact specification of the "fit," the problem takes on different flavors: for instance, if the curve is to pass close to all the points, then this becomes a *uniform approximation* problem (see text by Rivlin [33]), while if the curve is supposed to pass through *most* of the points, then it resembles problems from coding theory. Here, we consider a problem that unifies the above two instances over *discrete domains*. For example, given a set of m points, with integer coordinates, we show in section 4.1 how to find a polynomial with integer coefficients that is Δ-close to all but an ϵ fraction of the points (if such a polynomial exists), where ϵ need only be less than $1/2$ (provided m is larger than $\frac{(4\Delta+1)d}{1-2\epsilon}$).

Reducing bivariate factoring to univariate factoring. In [4] Berlekamp gave a randomized polynomial time algorithm for factoring univariate polynomials over finite fields. Kaltofen [24] and Grigoriev and Chistov [18] show that the problem of bivariate factoring can also be solved in polynomial time by a reduction to univariate factoring, using somewhat deep methods from algebra. Our techniques in section 4.2 give a simple method to reduce the problem of factoring bivariate polynomials to that of factoring univariate polynomials over finite fields in the special case when the bivariate polynomial splits into factors which are monic and of constant degree in one of the variables. Though the results are not new, nor as strong as existing results, the methods are much simpler than those used to obtain the previously known results.

Self-correcting programs. One motivation for this work comes from the area of self-correcting programs introduced independently in [7] and [28]. For many functions, one can take programs that are known to be correct on *most* inputs and apply a simple transformation to produce a program that is correct with high probability on *each* input. But how bad can a program be and still allow for such a transformation?

There is previous work addressing this question when the functions in question are polynomials (see, for example, [28], [13], [14]). When the program is not known to be correct on most inputs, the definition of self-correction needs to be modified, since the program can toggle between two seemingly correct functions. Our methods give self-correctors that work when the error of the program is such that it answers according to one of a small number of other algebraic functions. An algebraic decision tree may contain a small number of branches, in which all subtrees are intended to compute the same function but are computed separately for purposes of efficiency. The algebraic decision tree program might err in some of the subtrees and compute the wrong algebraic function. Our self-correctors output a small number of candidates for the correct function.

One particular situation where this is useful is in the computation of the permanent of a matrix, over a finite field. Results of Cai and Hemachandra [9], when used in combination with our results, imply that if there is an efficient program which computes the permanent correctly on a nonnegligible fraction of the input and computes one of a small number of other algebraic functions on the rest of the inputs, then the permanent can be computed efficiently everywhere.

1.1. The *k*-algebraic black box model. We consider the following black box reconstruction problem, which is general enough to model all of the aforementioned problems. We think of the black box as "containing" k functions, $f_1, \ldots, f_k$, where f_i is an "algebraically well-behaved" function. For instance, each f_i could be a polynomial of degree at most d, and on every input x the black box outputs $f_i(x)$ for some $i \in [k]$. (Here and throughout this paper, the notation $[k]$ stands for the set of integers $\{1, \ldots, k\}$.) Relating to the problem discussed in the first paragraph of the introduction, this corresponds to the case where for every value of an x-coordinate there is at least one point that has that value. We now present this definition formally, starting with the standard black box model ($k = 1$).

DEFINITION 1. *A* black box B *is an oracle representing a function from a finite domain* $\mathcal{D}$ *to a range* $\mathcal{R}$.

There are two kinds of domains that will be of interest to us. One is a finite subset H of a (potentially infinite) field F. The second is an n-dimensional vector space over a finite field F. In both cases the range will be the field F.

Previous research on black box reconstruction focused on the following: assuming that B is one of a special class of functions (for example, that B is a degree d polynomial), determine an explicit representation of B. In our model, there may be more than one output that is valid for each input. More specifically, we give the following definition.

DEFINITION 2. *A black box* B *mapping a finite subset* H *of a field* F *to* F *is a* (k, d)-polynomial black box *if there exist polynomials* $f_1, \ldots, f_k : F \to F$ *of degree at most* d *such that for every input* $x \in H$, *there exists* $i \in \{1, \ldots, k\}$, *such that* $B(x) = f_i(x)$. *In such a case, we say that the functions* $f_1, \ldots, f_k$ *describe the black box* B.

Our first *black box reconstruction problem* is:

> Given a (k, d)-polynomial black box B, find a set of functions $f_1, \ldots, f_k$ that describe B.

The definition of a (k, d)-polynomial black box can be generalized to situations involving noise as follows.

DEFINITION 3. *For* $\epsilon \in [0, 1]$ *and for a finite subset* H *of a field* F, *a black box* $B : H \to F$ *is an* ϵ-noisy (k, d)-polynomial black box *if there exist polynomials*

$f_1, \ldots, f_k : F \to F$ of degree at most d and a set $S \subset H$, with $|S| \geq (1-\epsilon)|H|$, such that for every input $x \in S$, there exists $i \in \{1, \ldots, k\}$ such that $B(x) = f_i(x)$. In such a case, the functions $f_1, \ldots, f_k$ are said to describe B.

The notion of the reconstruction problem generalizes to the noisy case in the obvious way. We now attempt to generalize the problem to allow the black box to compute other algebraic functions, such as $B(x) = \sqrt{x}$, etc. This part is somewhat more technical, so we now introduce some new terminology.

DEFINITION 4. *For positive integers d_x, d_y and indeterminates x, y, the $\{(d_x, x), (d_y, y)\}$-*weighted degree *of a monomial $x^i y^j$ is $id_x + jd_y$. The $\{(d_x, x), (d_y, y)\}$-weighted degree of a polynomial $Q(x, y)$ is the maximum over all monomials in Q (i.e., the monomials with nonzero coefficients in Q) of their $\{(d_x, x), (d_y, y)\}$-weighted degree.*

We now introduce the notion of an algebraic box and show how it relates to the earlier notion of a polynomial black box.

DEFINITION 5. *For a finite subset H of a field F, a black box $B : H \to F$ is a* (k, d)-algebraic black box *if there exists a polynomial $Q(x, y)$ of $\{(1, x), (d, y)\}$-weighted degree at most kd, such that for every input $x \in H$, the output y of the black box satisfies $Q(x, y) = 0$. In such a case, we say that the polynomial Q describes B.*

For example, if $B(x) = \sqrt{x}$, the polynomial $Q(x, y) = (y^2 - x)$ satisfies the requirement of the definition and describes B. The $\{(1, x), (d, y)\}$-weighted degree of Q is $2d$.

PROPOSITION 6. *If B is a (k, d)-polynomial black box then B is also a (k, d)-algebraic black box.*

Proof. Let B be a (k, d)-polynomial black box and let $f_1, \ldots, f_k$ describe it. Then the polynomial $Q(x, y) \stackrel{\text{def}}{=} \prod_{i=1}^{k} (y - f_i(x))$ describes B and has $\{(1, x), (d, y)\}$-weighted degree at most kd. □

The (k, d)-algebraic black box reconstruction problem is:

> Given a (k, d)-algebraic box B, find the polynomial Q of $\{(1, x), (d, y)\}$-weighted degree at most kd which describes it.

The definition and proposition can be extended easily to the ϵ-noisy case.

All the above definitions generalize to a case where the input is an n-dimensional vector over F and the black box is computing n-variate functions. In particular, we have the following definition.

DEFINITION 7. *For a finite field F, an n-variate black box $B : F^n \to F$ is a* (k, d)-polynomial black box *if there exist polynomials $f_1, \ldots, f_k : F^n \to F$ of total degree at most d such that for every input $(x_1, \ldots, x_n) \in F^n$ there exists $i \in \{1, \ldots, k\}$ such that $B(x_1, \ldots, x_n) = f_i(x_1, \ldots, x_n)$.*

DEFINITION 8. *For a finite field F, an n-variate black box $B : F^n \to F$ is a* (k, d)-algebraic black box *if there exists a polynomial $Q(x_1, \ldots, x_n, y)$ of $\{(1, x_1), \ldots, (1, x_n), (d, y)\}$-weighted degree at most kd such that for every input $(x_1, \ldots, x_n) \in F^n$, the output y of the black box satisfies $Q(x_1, \ldots, x_n, y) = 0$.*

Again the reconstruction problems are defined correspondingly. In this paper we attempt to solve all such reconstruction problems. Notice that this problem is not well defined if there exist multiple solutions, say Q and $\tilde{Q}$, such that both Q and $\tilde{Q}$ describe the black box. Much of the work is done in establishing conditions under which any $\tilde{Q}$ that describes the black box gives a meaningful answer.

1.2. Previous work and our results. The setting where the black box represents a *single* polynomial or rational function, without noise, is the classic interpolation problem and is well studied. Efficient algorithms for sparse multivariate polyno-

mial interpolation are given by Zippel [40, 41], Grigoriev, Karpinski, and Singer [21], Ben-Or [2] and Ben-Or and Tiwari [3], and for sparse rational functions by Grigoriev and Karpinski GK and Grigoriev, Karpinski, and Singer [20].

The case where the black box represents a single function with some noise has also been studied previously. Welch and Berlekamp [39, 5] (see also [14]) show how to reconstruct a univariate polynomial from a $(\frac{1}{2} - \delta)$-noisy $(1, d)$-polynomial black box and Coppersmith [11], Gemmell et al. [13], and Gemmell and Sudan [14] show how to do the same for multivariate polynomials. All the above-mentioned results require, however, that the field size be at least polynomially large in $\frac{d}{\delta}$. The conditions are required to ensure that there is a *unique* degree d polynomial describing the black box on a $\frac{1}{2} + \delta$ fraction of the inputs. Reconstructing functions from a black box representing more than one function, or when the function it represents is not guaranteed to be unique, seems to be a relatively unexplored subject. The works of Goldreich and Levin [15] and Kushilevitz and Mansour [26] are the only exceptions we know of. Both papers study the reconstruction of n-variate linear functions (i.e., homogenous polynomials of degree 1) from a $\frac{1}{2} - \delta$-noisy black box over GF(2). In this case, there can be up to $O(\frac{1}{\delta}^2)$ polynomials representing the black box and they reconstruct all such polynomials.

The main result in this paper is an algorithm for reconstructing algebraic functions describing noisy algebraic black boxes, which works when the black box satisfies certain conditions. To see why the result needs to have some conditions on the black box, consider the following example. Suppose the black box is described by the polynomial $(x^2 + y^2 - 1)(x + y - 1)$. But suppose that for every x the black box always outputs a y from the unit circle (and never according to the line $x + y - 1 = 0$). Then clearly the reconstruction algorithm has no information to reconstruct the line $x + y - 1$. The condition imposed on the black box essentially addresses this issue. We describe the result for univariate ϵ-noisy (k, d)-polynomial black boxes. We present a randomized algorithm which takes as input a parameter $p > \epsilon$ and with high probability outputs a list of all polynomials f_i which describe the black box on more than p fraction of the input, provided $(p - \epsilon)|H| > kd$. (This condition amounts to saying that the black box must output according to f_i sufficiently often.) The running time of the algorithm is a polynomial in k, d, and $\frac{1}{(p-\epsilon)}$. This result is presented along with generalizations to univariate noisy algebraic black boxes in section 2.

To reconstruct a univariate polynomial, we sample the black box on a small set of inputs and construct a bivariate polynomial $\tilde{Q}(x, y)$ which is zero at *all* the sample points. Then we use bivariate polynomial factorization to find a factor of the form $(y - f(x))$. If it exists, $f(x)$ then becomes our candidate for output. We show that if the number of points (x, y) such that $y = f(x)$ is large in the sample we chose, then $y - f(x)$ has to be a factor of any $\tilde{Q}$ which all the sample points satisfy.

Our results do not generalize immediately to multivariate polynomials. Among other factors, one problem is that an n-variate polynomial of degree d has $\binom{n+d}{d}$ coefficients, which is exponential in n (or d). This seems to make the problem inherently hard to solve in time polynomial in n and d. However, we bypass this, once again using the idea of black boxes. Instead of trying to reconstruct the multivariate polynomial explicitly (i.e., by determining all its coefficients), we allow the reconstruction algorithm to reconstruct the polynomial implicitly, i.e., by constructing a black box which computes the multivariate polynomial. If the polynomial turns out to be sparse then we can now use any sparse interpolation algorithm from [3, 20, 21, 40] to reconstruct an explicit representation of the polynomials in time polynomial in n, d and the

number of nonzero coefficients. On the other hand, by using the techniques of [25] we can also continue to manipulate the black boxes as they are for whatever purposes.[1]

We now describe our result for reconstructing multivariate polynomials. We present a randomized algorithm which takes as input a parameter p and with high probability reconstructs probabilistic black boxes for all the polynomials $f_1, \ldots, f_k$ describing a noisy (k, d)-black box over a finite field F, provided the noisy (k, d)-black black box satisfies the following conditions. (1) Every polynomial is represented on at least a p fraction of the inputs (i.e., for every i, $\Pr_{\hat{x} \in F^n}[B(\hat{x}) = f_i(\hat{x})] \geq p$). (2) The finite field F over which the black box works is sufficiently large ($|F|$ should be polynomially large in $k, d, \frac{1}{(p-\epsilon)}$). The running time of the algorithm is polynomial in k, d, and $\frac{1}{(p-\epsilon)}$. The main technique employed here is a randomized reduction from the multivariate to the univariate case. We note that the solution obtained here for the multivariate case differs in several aspects from the solution for the univariate case. First, this algorithm does not extend to the case of finite subsets of infinite fields. Second, it needs to make sure that all the polynomials are well represented in the black box. The latter aspect is a significant weakness, and getting around this is an open question.

Subsequent work. One of the main questions left open by this paper is the problem of reconstructing all degree d polynomials that agree with an arbitrary black box on ϵ fraction of the inputs. Some recent work has addressed this question. Goldreich, Rubinfeld, and Sudan [16] give an algorithm to (explicitly) reconstruct all n-variate degree d polynomials agreeing with a black box over F on an ϵ fraction of the inputs, provided $\epsilon \geq 2\sqrt{d/|F|}$. Their algorithm runs in time $O((n, \frac{1}{\epsilon})^{\mathrm{poly}(d)})$, which is exponential in d. Their algorithm generalizes the earlier mentioned solution of Goldreich and Levin [15]. For the case of univariate polynomials, Sudan [35] has given a polynomial time algorithm which can find all degree d polynomials agreeing with a black box on ϵ fraction of the domain, provided $\epsilon \geq 2\sqrt{d/|F|}$. The main contribution in [35] is a simple observation which shows that m input/output pairs from any black box can be thought of as the output of a $(O(\sqrt{m}), 1)$-algebraic black box. Using this observation, Lemma 18 of this paper is applied to reconstruct all polynomials of low degree which describe the black box on an ϵ fraction of the inputs. Finding a similar solution for the multivariate cases remains open (some cases are addressed by [35], but the problem is not completely resolved). A second question that is left open is the task of solving the (k, d)-polynomial black box problem over the reals, where the points are not provided to infinite precision. This is the true problem underlying the application to computer vision. While the ideas in this paper do not immediately apply to this question, some variants (in particular, the variant employed in [35]) seem promising and deserve to be investigated further

In other related work, Rubinfeld and Zippel [34] have employed the black box reconstruction problem and build on the techniques presented in this paper to present a modular approach to the polynomial factorization problem. While the application presented in this paper (in section 4.2) is to a restricted subclass of the bivariate factorization problem, the work of [34] finds an application to the general multivariate factorization problem.

[1] The idea of manipulating multivariate polynomials and rational functions represented by black boxes was proposed by Kaltofen and Trager in [25]. They show that it is possible to factor and compute gcd's for polynomials given by such a representation and to separate the numerator from the denominator of rational functions given by such a representation.

1.3. Organization. The rest of this paper is organized as follows. In section 2, we describe our results for univariate polynomials, rational functions, and other algebraic functions. In section 3, we consider extensions of the reconstruction problem to the case of multivariate polynomials. Finally, in section 4, we describe several applications of our work.

2. Univariate black boxes. In this section we consider the univariate reconstruction problem for (noisy) (k, d)-polynomial and algebraic black boxes. We describe the general format of our results with the example of a (k, d)-polynomial black box described by $f_1, \ldots, f_k$. We present a solution in the form of an algorithm which takes m input/output pairs $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ of the black box and attempts to reconstruct the polynomials $f_1, \ldots, f_k$ from this set of input/output pairs. In order to reconstruct a small set of polynomials which includes a specific polynomial f_i, the algorithm (obviously) needs to find sufficiently many points (x_j, y_j) such that $y_j = f_i(x_j)$ ($d + 1$ such points are needed). We present complementary bounds, showing that if the number of points on f_i is sufficiently large, then the output is guaranteed to include f_i. We then show how some simple sampling of the black box (either by exhaustively sampling all points from the domain H or by picking a random sample of x_j's chosen independently and uniformly at random from H) yields a collection of input/output pairs which satisfies the required condition, provided H is large enough and the fraction of inputs on which B's output is described by f_i is large enough.

2.1. An intermediate model. As a first step towards solving the algebraic reconstruction problem, we consider the case where the black box outputs *all* of $f_1(x), \ldots, f_k(x)$ on any input x. We refer to this as a (k, d)-*total polynomial black box*. These are output in arbitrary order, which is not necessarily the same for each x. We further assume that there are no errors in the output. Thus the reconstruction problem we wish to solve may be stated formally as follows.

Given: Positive integers k and d, a field F, and a black box $B = (B_1, \ldots, B_k)$, where $B_i : F \to F$ with the property that there exist polynomials $f_1, \ldots f_k$ of degree at most d over F, such that for every $x \in F$, the multisets $\{B_1(x), \ldots, B_k(x)\}$ and $\{f_1(x), \ldots, f_k(x)\}$ are identical.

Problem: Find $f_1, \ldots, f_k$.

We reduce the problem of extracting the polynomials to that of bivariate polynomial factorization. The main idea underlying this reduction is the following: on input x, if the (k, d)-total polynomial black box outputs $\{y_1, \ldots, y_k\}$, we know that $\forall j \in [k]$, $\exists i \in [k]$ such that $y_j = f_i(x)$. Therefore, each input/output pair $(x; y_1, \ldots, y_k)$ of the black box satisfies the relation:

$$\sum_j \prod_i (y_j - f_i(x)) = 0.$$

Our aim will be to construct a related polynomial which will enable us to recover the f_i's.

Consider the functions $\sigma_j : F \mapsto F$, $j \in [k]$ defined as

$$\sigma_j(x) \stackrel{\text{def}}{=} \sum_{S \subset [k], |S| = j} \prod_{i \in S} f_i(x)$$

(these are the *primitive symmetric* functions of $f_1, \ldots, f_k$).

Observe that $\sigma_j(x)$ can be evaluated at any input x using the given (k,d)-total polynomial black box, using the identity

$$\sigma_j(x) = \sum_{S \subset [k], |S|=j} \prod_{i \in S} B_i(x).$$

Furthermore this computation can be performed in time in $O(k \log k \log\log k)$ using a fast Fourier transform (see survey article by von zur Gathen [12, pp. 320–321]). Observe further that σ_j is a polynomial of degree at most jd. Hence evaluating it at $jd+1$ points suffices to find all the coefficients of this polynomial (if the black box outputs every $f_i(x)$ for every x).

Now consider the following bivariate polynomial, in x and a new indeterminate y:

$$Q(x,y) \stackrel{\text{def}}{=} y^k - \sigma_1(x)y^{k-1} + \cdots + (-1)^k \sigma_k(x).$$

From the explicit representation of the σ_i's, we can also compute an explicit representation of Q. But now notice that Q can equivalently be written as:

$$Q(x,y) = \prod_{i=1}^{k} (y - f_i(x)).$$

(The equivalence follows from the definition of the σ_j's.) Therefore, to recover the f_i's, all we have to do is find the factors of the bivariate polynomial Q. Bivariate factorization can be done efficiently over the rationals [18, 24, 29] and can be done efficiently (probabilistically) over finite fields [17, 24].

We now summarize our algorithm. The input to the algorithm is $kd+1$ distinct elements $\{x_1, \ldots, x_{kd+1}\}$ from the the field F, and a set $\{y_{1,j}, \ldots, y_{k,j}\}$ for every $j \in [kd+1]$ representing the output of the black box B on input x_j.

1. Evaluate $\sigma_j(x_i)$ for every $j \in [k]$ and every $i \in [kd+1]$.
2. Interpolate for the coefficients of $\sigma_j(x)$ and let $\sigma_{j,l}$ be the coefficient of x^l in σ_j.
3. Let $Q(x,y)$ be the polynomial $\sum_{j=0}^{k} \sum_{l=0}^{jd} (-1)^j \sigma_{j,l} x^l y^{k-j}$.
4. Factor Q into its irreducible factors. This will yield $Q(x,y) = \prod_{i=1}^{k} (y - g_i(x))$.
5. Output the polynomials $g_1, \ldots, g_k$.

The arguments leading to this algorithm prove its correctness and we have the following lemma.

LEMMA 9. *Let $\{(x_j; (y_{j,1}, \ldots, y_{j,k}))\}_{j=1}^{kd+1}$ be the input/output pairs of a (k,d) total polynomial black box B over a field F on $kd+1$ distinct inputs. Then there exists a randomized algorithm whose running time is polynomial in k, d which explicitly reconstructs the set of polynomials $\{f_1, \ldots, f_k\}$ which describe B.*

Since the only condition on the x_j's is that they be distinct, it is easy to get a total polynomial reconstruction algorithm from the above lemma and thus we get the following theorem.

THEOREM 10. *Let $f_1, \ldots, f_k$ be degree d polynomials over $\mathcal{Q}$ (the rationals) or a finite field F of cardinality at least $kd+1$. Given a black box B which on input x outputs the multiset $\{f_1(x), \ldots, f_k(x)\}$ (in arbitrary order), there exists an algorithm which queries the black box on $kd+1$ distinct inputs and reconstructs the polynomials that describe the black box. The algorithm is deterministic when the polynomials are over $\mathcal{Q}$ and probabilistic when the polynomials are over some finite field.*

2.2. (k, d)-polynomial black boxes. We now build on the methods of the previous section to reconstruct information from a (k, d)-polynomial black box which outputs the value of *one* of k univariate polynomials $f_1, \ldots, f_k$, on every input. Our method extends immediately to two more general cases:

1. (k, d)-algebraic black boxes,
2. noisy (polynomial and algebraic) black boxes.

The generalizations are dealt with in the next section.

The problem we wish to solve is formally stated as follows.

Given: Positive integers k and d, a field F, a finite set $H \subseteq F$, and a black box $B : H \to F$ with the property that there exist polynomials $f_1, \ldots f_k$ of degree at most d over F, such that for every $x \in H$, $B(x) \in \{f_1(x), \ldots, f_k(x)\}$.

Problem: Find $f_1, \ldots, f_k$.

Our solution for this problem is based on the solution of the previous subsection. The critical observation is that the polynomial Q produced by the algorithm of the previous section always satisfied the property $Q(x, y) = 0$ for any input x to the black box and where y is any element of the output set of the black box on input x. We will try to construct a polynomial Q in two variables as in the previous section, satisfying the property that if $y = B(x)$ is the output of the black box on input x, then $Q(x, y) = 0$. However, we will not be able to construct the polynomials $\sigma_i(x)$ as in the previous case. Hence, we will abandon that part of the algorithm and directly try to find any polynomial $\tilde{Q}$ such that $\tilde{Q}(x, y) = 0$ on all the sampled points. We will then use the factors of this polynomial to determine the f_i's as in the previous section. Thus our algorithm is summarized as follows.

The input to the algorithm is m distinct pairs of elements $\{(x_1, y_1), \ldots, (x_m, y_m)\}$.

1. Interpolate to find a set of coefficients $\tilde{q}_{lj}$ of the polynomial

$$\tilde{Q}(x, y) = \sum_{l=0}^{k} \sum_{j=0}^{dl} \tilde{q}_{lj} y^{k-l} x^j$$

   that satisfies $\tilde{Q} \not\equiv 0$ and $\tilde{Q}(x_i, y_i) = 0$ for $i \in [m]$.
2. Factor the polynomial $\tilde{Q}$ and if it has any factors of the form $(y - g(x))$, output g as a candidate polynomial.

Notes. The important step above is step 1, which involves finding a nontrivial solution to a homogenous linear system. First we need to make sure this system has at least *one* solution. This is easy since $Q(x, y) = \prod_i (y - f_i(x))$ is such a solution. However, the solution in such a step need not necessarily be unique and we will simply find *any* solution to this system and show that it suffices, under certain conditions, for step 2. In what follows, we shall examine the conditions under which the output will include a certain polynomial f_i.

LEMMA 11. *For a set $\{(x_j, y_j) | j \in [m]\}$ of m distinct pairs from $F \times F$, if $\tilde{Q}$ is a bivariate polynomial of $\{(1, x), (d, y)\}$-weighted degree kd satisfying*

$$\forall\ j \in [m],\ \tilde{Q}(x_j, y_j) = 0$$

and f is a univariate polynomial of degree d satisfying

$$|\{j | f(x_j) = y_j\}| > kd,$$

then the polynomial $(y - f(x))$ divides the polynomial $\tilde{Q}(x, y)$.

Proof. Let $S \stackrel{\text{def}}{=} \{j | f(x_j) = y_j\}$. Notice that for distinct $j_1, j_2 \in S$, $x_{j_1} \neq x_{j_2}$, or else the pairs $(x_{j_1}, f(x_{j_1}))$ and $(x_{j_2}, f(x_{j_2}))$ are not distinct.

Consider the univariate polynomial $\tilde{Q}_f(x) \equiv \tilde{Q}(x, f(x))$. For all indices $j \in S$ we have that $\tilde{Q}(x_j) = 0$. Furthermore, $\tilde{Q}_f(x)$ is a polynomial of degree at most kd in x. Hence if $\tilde{Q}_f$ is zero at $|S| > kd$ places, then it must be identically zero, implying that $(y - f(x))|\tilde{Q}(x, y)$. □

The lemma above guarantees that under certain circumstances, the factors of $\tilde{Q}(x, y)$ do give useful information about the f_i's. The effect is summarized in the following lemma.

LEMMA 12. *Let $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ be m distinct pairs of elements which are the input/output pairs of a (k, d)-polynomial black box B described by polynomials $f_1, \ldots, f_k$. If there exists an $i \in [k]$ such that $|\{j | y_j = f_i(x_j)\}| > kd$, then a set of at most k polynomials $\{g_1, \ldots, g_k\}$ that includes f_i can be found in time polynomial in m, k, and d.*

Remark. Notice that Lemma 12 is a strict strengthening of Lemma 9.

To finish the analysis of the algorithm we need to determine how to sample the black box B so as to get enough points according to f_i. Let $p_i \stackrel{\text{def}}{=} \Pr_{x \in H}[B(x) = f_i(x)]$ and $\delta > 0$ be the confidence parameter. Let $M = \frac{4}{p_i}(kd + \ln \frac{2}{\delta})$. The strategy for picking the points $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ depends on $|H|$. If $|H| \geq \frac{2}{\delta}\binom{M}{2}$, then we let $m = M$ and pick m elements $x_1, \ldots, x_m$ independently and uniformly at random from H. Lemma 35 in the appendix (shown using a simple combination of Chernoff bounds and the "birthday problem analysis") shows that the sampled points are all distinct and satisfy $|\{j : B(x_j) = f_i(x_j)\}| > kd$ with probability at least $1 - \delta$. Thus in this case we will use $\{(x_1, B(x_1)), \ldots, (x_m, B(x_m))\}$ as the input to the algorithm described above. If, on the other hand, $|H|$ is not large enough, then we will simply sample every point in H (i.e., the input set will be $\{(x, B(x)) | x \in H\}$), implying in particular that $m = |H|$, and in this case the algorithm described above will include f_i as part of its output provided $p_i|H| > kd$. Notice that in both cases the running time of the algorithm is polynomial in M, which is in turn bounded by some polynomial in $k, d, \frac{1}{p_i}, \frac{1}{\delta}$. Furthermore, by choosing $\delta' = \delta/k$ and a threshold parameter p and running the algorithm above with confidence parameter δ', we find that the algorithm above recovers, with confidence $1 - \delta$, every polynomial f_i, such that $p_i \geq p$. The running time is still a polynomial in $k, d, \frac{1}{p}, \frac{1}{\delta}$. This yields the following theorem.

THEOREM 13. *Let B be a (k, d)-polynomial black box, mapping a finite domain H to a field F, described by polynomials $f_1, \ldots, f_k$. For $i \in [k]$, let $p_i \stackrel{\text{def}}{=} \Pr_{x \in H}[B(x) = f_i(x)]$. There exists an algorithm which takes as input a confidence parameter $\delta > 0$ and a threshold $p > 0$, runs in time $\text{poly}(k, d, \frac{1}{p}, \frac{1}{\delta})$ and makes calls to the black box B, and with probability at least $1 - \delta$ reconstructs a list of at most k polynomials which includes all polynomials f_i such that $p_i \geq p$, provided $p > \frac{kd}{|H|}$.*

2.3. (k, d)-algebraic black boxes. The algorithm of section 2.2 extends immediately to the case of algebraic black boxes. Here, by definition, the input/output pair of the black box, (x, y), satisfies an algebraic relation of the form $Q(x, y) = 0$. We can attempt to find a polynomial $\tilde{Q}$ which satisfies $\tilde{Q}(x, y) = 0$ for all the sampled points by interpolation (step 3 in the algorithm of section 2.2).

As in the previous section, it will not be possible to guarantee that the output we produce will be exactly Q. For instance, if $Q(x, y) = (x^2 + y^2 - 1)(x + y - 1)$, but all the points actually come from the unit circle, then the algorithm has no information to

point to the line $x+y-1=0$. Thus, as in the previous section, we will only attempt to find those parts of the curve that describe significant portions of output of the black box. More precisely, if $Q(x,y)$ factors into irreducible factors $Q_1(x,y),\ldots,Q_l(x,y)$ and we know that many points satisfy, say, $Q_1(x_j,y_j)=0$, then we would like Q_1 to be one of the outputs of the algorithm.

The proof that this is indeed the case is slightly more complicated than in the previous subsection. We will use a version of Bezout's theorem ([38, Theorem 3.1]). Essentially, Bezout's theorem states that two algebraic curves in the plane cannot intersect in infinitely many points, unless they are identical. The theorem gives an explicit bound on the number of points where two curves of degree d_1 and d_2 may meet. Bezout's bound is slightly weaker than the one we wish to prove for the case of (k,d)-algebraic black boxes, so we prove our lemma from first principles.

Before going on to the next lemma we review a couple of standard definitions from algebra (cf. [38]).

DEFINITION 14. *Given univariate polynomials* $P(y)=\sum_{i=0}^{d_1}\alpha_i y^i$ *and* $Q(y)=\sum_{j=0}^{d_2}\beta_j y^j$ *over some domain* F*, let* $M(P,Q)$ *be the* $(d_1+d_2)\times(d_1+d_2)$ *matrix given as follows:*

$$M(P,Q)=\begin{bmatrix}
\alpha_0 & \alpha_1 & \cdots & \alpha_{d_1-1} & \alpha_{d_1} & 0 & \cdots & 0 & 0 & 0 & \cdots & 0\\
0 & \alpha_0 & \cdots & \alpha_{d_1-2} & \alpha_{d_1-1} & \alpha_{d_1} & \cdots & 0 & 0 & 0 & \cdots & 0\\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & \alpha_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{d_1}\\
\beta_0 & \beta_1 & \cdots & \beta_{d_1-1} & \beta_{d_1} & \beta_{d_1+1} & \cdots & \beta_{d_2-1} & \beta_{d_2} & 0 & \cdots & 0\\
0 & \beta_0 & \cdots & \beta_{d_1-2} & \beta_{d_1-1} & \beta_{d_1} & \cdots & \beta_{d_2-2} & \beta_{d_2-1} & \beta_{d_2} & \cdots & 0\\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\
0 & 0 & \cdots & \beta_0 & \beta_1 & \beta_2 & \cdots & \beta_{d_2-d_1} & \beta_{d_2-d_1+1} & \beta_{d_2-d_1+2} & \cdots & \beta_{d_2}
\end{bmatrix}.$$

The resultant *of the polynomials* P *and* Q*, denoted* $\mathrm{Res}(P,Q)$*, is the determinant of* $M(P,Q)$*. For multivariate polynomials* $P(x_1,\ldots,x_n;y)$ *and* $Q(x_1,\ldots,x_n,y)$ *their resultant with respect to* y *is defined similarly by viewing* P,Q *as polynomials in* y *with coefficients from the ring of polynomials in* $x_1,\ldots,x_n$*. We define the matrix* $M_y(P,Q)$ *as above and its determinant is the resultant* $\mathrm{Res}_y(P,Q)$*.*

LEMMA 15. *For a set of points* $\{(x_1,y_1),\ldots,(x_m,y_m)\}$*, with the* x_j*'s being distinct, if* $\tilde{Q}(x,y)$ *and* $Q_1(x,y)$ *are polynomials of* $\{(1,x),(d,y)\}$*-weighted degree at most* kd *and* $k_1 d$*, respectively, satisfying the properties* (1) $\forall\ j\in[m]$, $\tilde{Q}(x_j,y_j)=0$ *and* (2) $|\{j|Q_1(x_j,y_j)=0\}|>kk_1d$*, then the polynomials* $Q_1(x,y)$ *and* $\tilde{Q}(x,y)$ *share a nonconstant common factor.*

Proof. Consider the resultant $R_y(x)$ of the polynomials $\tilde{Q}(x,y)$ and $Q_1(x,y)$ with respect to y. Observe that the resultant is a polynomial in x. The following claim bounds the degree of this polynomial.

CLAIM 16. $R_y(x)$ *is a polynomial of degree at most* k_1kd*.*

Proof. The determinant of the matrix $M_y(\tilde{Q},Q_1)$ is given by

$$\sum_{\pi}\prod_{i=1}^{k+k_1}\mathrm{sign}(\pi)(M_y(\tilde{Q},Q_1))_{i\pi(i)},$$

where π ranges over all permutations from $[k+k_1]$ to $[k+k_1]$ and $\mathrm{sign}(\pi)$ denotes the sign of the permutation. We will examine every permutation $\pi:[k+k_1]\to[k+k_1]$

and show that the degree of the term $\prod_{i=1}^{k+k_1}(M_y(\tilde{Q},Q_1))_{i\pi(i)}$ (viewed as a polynomial in x) is at most kk_1d. This will suffice to show that the determinant is a polynomial of degree at most kk_1d.

Let d_{ij} denote the degree of the entry $(M_y(\tilde{Q},Q_1))_{ij}$. Observe that, by the definition of the resultant, $d_{ij} \leq (i+k-j)d$ for $i \leq k_1$ and $d_{ij} \leq (i-j)d$ for $i \geq k_1$. (Here we consider the polynomial 0 as having degree $-\infty$.) Thus the degree of the term $\prod_{i=1}^{k+k_1}(M_y(\tilde{Q},Q_1))_{i\pi(i)}$ is given by

$$\begin{aligned}\sum_{i=1}^{k+k_1} d_{i\pi(i)} &= \sum_{i=1}^{k_1} d_{i\pi(i)} + \sum_{i=k_1+1}^{k+k_1} d_{i\pi(i)} \\ &\leq \sum_{i=1}^{k_1}(i+k-\pi(i))d + \sum_{i=k_1+1}^{k+k_1}(i-\pi(i))d \\ &= \sum_{i=1}^{k+k_1} id - \sum_{i=1}^{k+k_1}\pi(i)d \;+\; kk_1d \\ &= kk_1d.\end{aligned}$$

This concludes the proof. □

It is well known that the resultant of two polynomials is zero if and only if the polynomials share a common factor (cf. [38, Chapter 1, Theorem 9.3]). We will show that $R_y(x)$ is identically zero, and this will suffice to prove the lemma. We show this part in the next claim by showing that $R_y(x)$ has more zeroes than the upper bound on its degree.

CLAIM 17. *For every j such that $\tilde{Q}(x_j,y_j) = Q_1(x_j,y_j) = 0$, $R_y(x_j) = 0$.*

Proof. Fix x_j and consider the polynomials $\tilde{q}(y) = \tilde{Q}(x_j,y)$ and $q_1(y) = Q_1(x_j,y)$. Now $R_y(x_j)$ gives the resultant of the polynomials $\tilde{q}(y)$ and $q_1(y)$. Now we know that $\tilde{q}(y_j) = q_1(y_j) = 0$, implying that $(y-y_j)$ is a common factor of $\tilde{q}$ and q_1. Therefore, the resultant of $\tilde{q}$ and q_1 must be zero, implying $R_y(x_j) = 0$. □

Since the above holds for any factor Q_i of Q, we have the following lemma.

LEMMA 18. *Let B be a (k,d)-algebraic black box described by a bivariate polynomial Q with no repeated nonconstant factors. Let $Q_1,\ldots,Q_l$ be the irreducible factors of Q of $\{(1,x),(d,y)\}$-weighted degree $k_1d,\ldots,k_ld$, respectively. Given m pairs of elements $\{(x_1,y_1),\ldots,(x_m,y_m)\}$ which are the input/output pairs of B on m distinct inputs, if there exists an $i \in [k]$ such that $|\{j|Q_i(x_j,y_j) = 0\}| > k_ikd$, then a set of at most k polynomials $\{\tilde{Q}_1,\ldots,\tilde{Q}_k\}$ that includes Q_i can be found in time polynomial in m, k, and d.*

Remark. For a set of pairs $\{(x_1,y_1),\ldots,(x_m,y_m)\}$, with distinct x_j's, Lemma 18 is a strengthening of Lemma 12. Unfortunately, the proof as shown above does not extend to the case where the pairs are distinct, but the x_j are not. Due to this limitation, Lemma 18 does not even cover the case of Lemma 9.

Once again, using a sampling method similar to that used for Theorem 13, we get the following theorem.

THEOREM 19. *Let B be a (k,d)-algebraic black box described by a polynomial Q with distinct irreducible factors $Q_1,\ldots,Q_l$ such that the $\{(1,x),(d,y)\}$-weighted degree of Q is at most kd and that of Q_i is at most k_id. Further, let $p_i = \Pr_{x\in H}[Q_i(x,B(x)) = 0]$. There exists a randomized algorithm which takes as input a confidence parameter $\delta > 0$ and a threshold $p > 0$, runs in time* $\mathrm{poly}(k,d,\frac{1}{p},\frac{1}{\delta})$, *makes calls to the*

black box B, and with probability at least $1-\delta$ reconstructs a list of at most k bivariate polynomials which includes every polynomial Q_i such that $p_i/k_i \geq p$, provided $p|H| > kd$.

2.4. ϵ-noisy black boxes. Finally we extend the reconstruction algorithms of the previous section to the case when the black boxes are allowed to output noise on an ϵ fraction of the inputs from H. As usual, the basic algorithm will be to find a polynomial $\tilde{Q}(x,y)$ which is zero on all the input-output pairs of the black box. However, we will have to do something about the noisy points which do not lie on any nice algebraic curve. We adapt an algorithm of Welch and Berlekamp [39, 5] (see also [14]) to handle this situation.

Say we sample the black box B in m points $x_1,\ldots,x_m$ and the black box outputs $y_1,\ldots,y_m$ according to some (unknown) polynomial Q in all but m' locations. Say that these locations are given by $E=\{j|Q(x_j,B(x_j))\neq 0\}$. We use the fact that there exists a nonzero polynomial $W(x)$ of degree at most m' which is zero when $x=x_j$ for $j\in E$. Indeed $W(x)=\prod_{j\in E}(x-x_j)$ is such a polynomial. Let $Q^*(x,y)=Q(x,y)\cdot W(x)$. Then $Q^*(x_j,y_j)$ is zero for all $j\in[m]$. Thus we can modify the algorithm of the previous section to try to find Q^*. This algorithm is summarized as follows.

The input to the algorithms is m pairs of elements $\{(x_1,y_1),\ldots,(x_m,y_m)\}$ with distinct x_j's.

1. Interpolate to find a set of coefficients q_{lj} of the polynomial

$$\tilde{Q}(x,y)=\sum_{l=0}^{k}\sum_{j=0}^{dl+m'} q_{lj}y^{k-l}x^j$$

   that satisfies $\tilde{Q}(x_i,y_i)=0$ for $i\in[m]$. /* The parameter m' will be specified later. */
2. Factor the polynomial $\tilde{Q}$ and output all its irreducible factors.

Let $Q_1(x,y),\ldots,Q_l(x,y)$ be the irreducible factors of the unknown polynomial $Q^*(x,y)$ describing the black box B. We focus on the factor Q_1. Let the $\{(1,x),(d,y)\}$-weighted degree of Q_1 be k_1d. The following two lemmas essentially show that if the fraction of points (x_i,y_i) for which $Q_1(x_i,y_i)=0$ is sufficiently larger than k_1 times the fraction of noise, then we can reconstruct the polynomial Q_1.

LEMMA 20. *For a set of points $\{(x_j,y_j)|j\in[m]\}$, if $\tilde{Q}(x,y)$ and $Q_1(x,y)$ are polynomials of $\{(1,x),(d,y)\}$-weighted degree at most $kd+m'$ and k_1d, respectively, satisfying the properties (1) $\forall\ j\in[m]$, $\tilde{Q}(x_j,y_j)=0$ and (2) $|\{j|Q_1(x_j,y_j)=0\}|>k_1(kd+m')$, then the polynomials $Q_1(x,y)$ and $\tilde{Q}(x,y)$ share a nonconstant common factor.*

Proof. The proof is a straightforward modification of the proof of Lemma 15. The only change is in Claim 16, where the bound on the degree of the resultant $\mathrm{Res}_y(\tilde{Q},Q_1)$ goes up to $k_1(kd+m')$, because the degree of the nonzero entries in the first k_1 columns goes up by m'. □

LEMMA 21. *Let B be a (k,d)-algebraic black box described by a bivariate polynomial Q with no repeated nonconstant factors. Let $Q_1,\ldots,Q_l$ be the irreducible factors of Q of at most $\{(1,x),(d,y)\}$-weighted degree $k_1d,\ldots,k_ld$, respectively. Given $m'\leq m$ and m pairs $\{(x_1,y_1),\ldots,(x_m,y_m)\}$, which are the input/output pairs of B on distinct inputs, if there exists an $i\in[k]$ such that $|\{j|Q_i(x_j,y_j)=0\}|>k_i(m'+kd)$ and $|\{j|Q(x_j,y_j)\neq 0\}|\leq m'$, then a set of at most k bivariate polynomials that includes Q_i can be found in time polynomial in m, k, and d.*

Lemma 36 of the appendix ensures that if $M = \frac{k_i kd}{p-k_i\epsilon} + \frac{16}{(p-k_i\epsilon)^2}\ln\frac{3}{\delta}$ and $|H| \geq \frac{3}{\delta}\binom{M}{2}$, then a sample of M elements $\{x_1, \ldots, x_M\}$ chosen independently and uniformly at random from F satisfies the following three properties.

1. The x_j's have no repeated elements.
2. There are (strictly) less than $((\epsilon + p_i/k_i)/2)M - kd/2$ values of j such that $Q(x_j, B(x_j)) \neq 0$.
3. There are at least $((p_i+k_i\epsilon)/2)M+k_ikd/2$ values of j such that $Q_i(x_j, B(x_j)) = 0$.

Thus if $|H| \geq \frac{3}{\delta}\binom{M}{2}$, then we randomly choose $m = M$ points from H and use $\{(x_1, B(x_1)), \ldots, (x_m, B(x_m))\}$ and $m' = ((\epsilon + p_i/k_i)/2)M - kd/2 - 1$ as input to the algorithm described above. If, on the other hand, H is small, then we use all $\{(x, B(x))|x \in H\}$ as the input set and use $m' = \epsilon|H|$ as input to our algorithm. In the latter case, Q_i is guaranteed to be part of the output if $(p_i - k_i\epsilon)|H| > kk_id$. This yields the following theorem.

THEOREM 22. *Let B be an ϵ-noisy (k, d)-algebraic black box described by a polynomial Q with no repeated nonconstant factors. Further, let $Q_1, \ldots, Q_l$ be the (distinct) irreducible factors of Q and let $p_i \stackrel{\text{def}}{=} \Pr_{x\in H}[Q_i(x, B(x)) = 0]$. There exists an algorithm which takes as input $\delta, p > 0$, runs in time $\text{poly}(k, d, \frac{1}{p-\epsilon}, \frac{1}{\delta})$, and with probability at least $1 - \delta$ reconstructs a list of at most k bivariate polynomials which includes every Q_i such that $p_i/k_i \geq p$, provided $(p - \epsilon)|H| > kd$.*

3. Multivariate black boxes. In this section, we extend Theorem 22 to multivariate polynomial black boxes over finite fields. The methods of section 2, i.e., those based on trying to find the coefficients of polynomials simultaneously, do not seem to extend directly to the general multivariate case. This is due to the possibly large *explicit* representation of the function extracted from the black box, which makes it inefficient to work with. Instead, we use techniques of pairwise independent sampling to reduce the problem to a univariate situation and then apply Theorem 22 to the new univariate problem. We start by summarizing the problem.

Given: An n-variate ϵ-noisy (k, d)-polynomial black box $B : F^n \to F$. That is, there exist n-variate polynomials $f_1, \ldots, f_k$ of total degree at most d such that

$$\Pr_{\hat{x}\in F^n}[\exists i \in [k] \text{ s.t. } B(\hat{x}) = f_i(\hat{x})] \geq 1 - \epsilon,$$

and furthermore, each f_i is well represented in B; i.e.,

$$\forall\ i \in [k] \Pr_{\hat{x}\in F^n}[B(\hat{x}) = f_i(\hat{x})] \geq p > \epsilon.$$

Problem: Construct k black boxes computing the functions $f_1, \ldots, f_k$.

Notice that we have changed the problem from that of the previous section in several ways. First, we no longer ask for an explicit representation of f_i, but allow for implicit representations. This is a strengthening of the problem, since explicit representations may be much longer than implicit ones, and thus allow a reconstruction algorithm much more time than we do. For instance, if the reconstructed function is a sparse multivariate polynomial, then we can use any of the sparse multivariate polynomial interpolation algorithms given in [3, 20, 21, 40] to recover explicit representations of the reconstructed functions, in running time which is polynomial in the number of nonzero coefficients rather than the total number of possible coefficients. A second change from the problem of the previous section is that we expect all the

polynomials $f_1, \ldots, f_k$ to be well represented in the black box B. This is a weakening of the problem, and we do not know how to get around it.

The outline of the method we use to solve the above problem is as follows. Consider first the slightly simpler problem: given B and an input $\hat{b} \in F^n$, find the multiset $\{f_1(\hat{b}), \ldots, f_k(\hat{b})\}$. This we solve by a reduction to a univariate version of the reconstruction problem. Now a solution to this problem does not immediately suffice to yield a solution to the n-variate reconstruction problem as described above. This is because the solution produces a multiset of values $\{y_1, \ldots, y_k\}$ for which we do not know which y_j corresponds to f_i. We want the black boxes to always output according to the same polynomial consistently.

In order to solve this problem, we introduce the notion of a *reference point* $\hat{r} \in F^n$, which will have the property that the value of the k different polynomials will be all distinct on this point. We will then use a more general reduction to the univariate problem which will allow us to reconstruct a set of pairs $\{(y_1, z_1), \ldots, (y_k, z_k)\} = \{(f_1(\hat{b}), f_1(\hat{r})), \ldots, (f_k(\hat{b}), f_k(\hat{r}))\}$. This, along with the property of the reference point, allows us to order the points consistently for all inputs $\hat{b}$. We now go into the details.

3.1. Reference points. The following definition of a reference point is motivated by the above discussion. We wish to consider the polynomial $Q(x_1, \ldots, x_n; y) = \prod_{i=1}^{k}(y - f_i(x_1, \ldots, x_n))$, and want to ensure that at the reference point $\hat{r}$, $f_i(r_1, \ldots, r_n) \neq f_j(r_1, \ldots, r_n)$, whenever $i \neq j$. One way to test for this is to see if the polynomial $p(y) \stackrel{\text{def}}{=} Q(r_1, \ldots, r_n; y)$ has any repeated nonconstant factors. This will be our definition of a reference point. Note that this definition is general enough to apply also to polynomials Q which do not factor linearly in y.

DEFINITION 23. *For a multivariate polynomial $Q(x_1, \ldots, x_n; y)$ that has no nonconstant repeated factors, a* reference point *is an element $\hat{r} = (r_1, \ldots, r_n)$ of F^n such that the univariate polynomial $p(y) \stackrel{\text{def}}{=} Q(r_1, \ldots, r_n; y)$ has no repeated nonconstant factors.*

The next lemma will show that a random point is likely to be a reference point for any given polynomial Q, provided the field size is large compared to the degree of the polynomial Q. We will need one more notion which is standard in algebra.

DEFINITION 24. *The discriminant of a univariate polynomial $Q(y)$, denoted Δ, is $\mathrm{Res}(Q, Q')$ where Q' is the derivative of Q with respect to y. The discriminant of a multivariate polynomial $Q(x_1, \ldots, x_n; y)$ with respect to y, denoted $\Delta(x_1, \ldots, x_n)$, is defined to be $\mathrm{Res}_y(Q, Q')$ where Q' is the derivative of Q with respect to y. (Formally, the derivative of a monomial $q_i y^i$ is $(q_i + \cdots + q_i)y^{i-1}$, where the summation is of i q_i's. The derivative of a polynomial is simply the sum of the derivatives of the monomials in it.)*

The above definition is motivated by the following well-known fact: a polynomial p (over any unique factorization domain) has repeated nonconstant factors if and only if it shares a common factor with its derivative (cf. [27, Theorem 1.68]). From the well-known fact about resultants, this extends to saying that a polynomial has repeated nonconstant factors if and only if its discriminant is zero.

LEMMA 25. *For a polynomial $Q(x_1, \ldots, x_n; y)$ of $\{(1, x_1), \ldots, (1, x_n), (d, y)\}$-weighted degree at most kd with no repeated nonconstant factors, a random point $\hat{r} \in F^n$ is a reference point with probability at least $1 - \frac{k(k-1)d}{|F|}$.*

Proof. Let $\Delta(x_1, \ldots, x_n)$ be the discriminant of Q with respect to y. Notice that Q' is a polynomial of $\{(1, x_1), \ldots, (1, x_n), (d, y)\}$-weighted degree at most $(k-1)d$.

Thus, as in Claim 16, we can show that $\Delta(x_1, \ldots, x_n)$ is a polynomial in $x_1, \ldots, x_n$ of degree at most $k(k-1)d$. Since Q has no repeated factors, $\Delta(x_1, \ldots, x_n)$ is not identically zero. Thus for a random point $\hat{r} \in F^n$, the probability that $\Delta(\hat{r}) = 0$ is at most $k(k-1)d/|F|$. But observe that $\Delta(\hat{r})$ is the discriminant of the univariate polynomial $p(y) \stackrel{\text{def}}{=} Q(r_1, \ldots, r_n, y)$, and if $\Delta(\hat{r}) \neq 0$, then $\hat{r}$ is a reference point. □

3.2. Reduction to the univariate case. We now consider the case where we are given a black box B, described by polynomials $f_1, \ldots, f_k$, and two points $\hat{a}, \hat{b} \in F^n$, and we wish to find a set of k pairs $\{(y_1, z_1), \ldots, (y_k, z_k)\}$ such that for every $i \in [k]$, there exists some $j \in [k]$ such that $(y_j, z_j) = (f_i(\hat{a}), f_i(\hat{b}))$. We solve this problem by creating a univariate reconstruction problem and then using Theorem 22 to solve this problem. This reduction builds upon a method of [14], which in turn builds upon earlier work of [1, 13].

We create a univariate "subdomain," more precisely, a function $D : F \to F^n$, such that the image of the domain, $\text{Im}(D) \stackrel{\text{def}}{=} \{D(t) | t \in F\}$, satisfies the following properties.

1. $\hat{a}$ and $\hat{b}$ are contained in $\text{Im}(D)$.
2. The restriction of a polynomial $\tilde{Q}(x_1, \ldots, x_n, y)$ of $\{(1, x_1), \ldots, (1, x_n), (d, y)\}$-weighted degree kd to $\text{Im}(D)$, i.e., the function $\tilde{Q}^D(t, y) \stackrel{\text{def}}{=} \tilde{Q}(D(t), y)$, is a bivariate polynomial of $\{(1, t), (3d, y)\}$-weighted degree $3kd$.
3. $\text{Im}(D)$ resembles a randomly and independently chosen sample of F^n of size $|F|$. In particular, with high probability, the fraction of points from $\text{Im}(D)$ where the black box responds with $f(x_1, \ldots, x_n)$ is very close to the fraction of points from F^n where the black box responds with f.

For a finite field F, with $|F| > 3$, $D = (D_1, \ldots, D_n)$, where $D_i : F \to F$ is constructed by picking vectors $\hat{c} = (c_1, \ldots, c_n)$ and $\hat{d} = (d_1, \ldots, d_n)$ at random from F^n and setting $D_i(t) = a_i + c_i t + d_i t^2 + (b_i - c_i - d_i - a_i)t^3$ for $i \in [n]$. By construction, it is immediately clear that the "subdomain" D satisfies properties (1) and (2) listed above. The following lemma shows that it also satisfies property (3) above.

LEMMA 26. *For sets $S_1, \ldots, S_k, E \subset F^n$, let $p_i \stackrel{\text{def}}{=} |S_i|/|F^n|$ and $\epsilon \stackrel{\text{def}}{=} |E|/|F^n|$ and let $\gamma > 0$. Then*

$$\Pr_{\hat{c},\hat{d}}[\exists i \in [k] \text{ s.t. } |\text{Im}(D) \cap S_i|/|F| \leq p_i - \gamma/2 \text{ or } |\text{Im}(D) \cap E|/|F| \geq \epsilon + \gamma/2] \leq \frac{k+1}{\gamma^2(|F|-2)}.$$

Proof. Observe that the set of points $\{D(t) | t \in F \setminus \{0, 1\}\}$ constitutes a pairwise independent sample of points chosen uniformly at random from F^n. The lemma now follows from a standard application of Chebyshev bounds. □

Thus we obtain the following algorithm (tuned for confidence parameter $\delta = 1/3$). The algorithm is given a threshold p.

1. Pick $\hat{c}, \hat{d}$ at random from F^n.
2. Let $D(t) = (D_1(t), \ldots, D_n(t))$ be given by $D_i(t) = a_i + c_i t + d_i t^2 + (b_i - a_i - c_i - d_i)t^3$, and let $B' : H \to F$ be the black box given by $B'(t) = B(D(t))$, where $H = F - \{0, 1\}$.
3. Reconstruct all univariate polynomials $g_1, \ldots, g_k$ of degree at most $3d$ describing B', for threshold p and confidence $1 - 1/6$.

4. Output $\{(g_1(0), g_1(1)), \ldots, (g_k(0), g_k(1))\}$.

Let $\gamma = \sqrt{\frac{6(k+1)}{(|F|-2)}}$. By Lemma 26 we know that the above algorithm finds a univariate domain D, s.t. at most an $\epsilon + \gamma/2$ fraction of the points on the domain are "noisy" and every polynomial is represented on at least a $p_i - \gamma/2$ fraction of the domain, with probability at least $1-1/6$. Thus if $p_i \geq p$ for every i, and $(p-\epsilon-\gamma)(|F|-2) > 3kd$, then the univariate reconstruction algorithm is guaranteed to find all the f_i's, with probability at least $1 - 1/6$. The condition on $|F|$ above can be simplified (somewhat) to $(p-\epsilon)|F| > 3kd + \sqrt{6(k+1)|F|} + 2$, and under this condition the algorithm above returns $\{(f_1(0), f_1(1)), \ldots, (f_k(0), f_k(1))\}$ correctly with probability at least $2/3$. Notice that by repeating $\log \frac{1}{\delta}$ times and outputting the majority answer (i.e., the set that is output most often), we can boost the confidence up to $1 - \delta$, for any $\delta > 0$. This yields the following lemma.

LEMMA 27. *Given an ϵ-noisy n-variate polynomial black box B described by polynomials $f_1, \ldots, f_k$ of degree d, s.t.*

$$\forall i \in [k], \Pr_{\hat{x} \in F^n} [B(\hat{x}) = f_i(\hat{x})] \geq p > \epsilon,$$

there exists a randomized algorithm that takes as input $\delta, p > 0$ and $\hat{a}, \hat{b} \in F^n$, runs in time $\mathrm{poly}(n, k, d, \frac{1}{(p-\epsilon)}, \log \frac{1}{\delta})$, *and outputs the set of k ordered pairs $\{(f_1(\hat{a}), f_1(\hat{b})), \ldots, (f_k(\hat{a}), f_k(\hat{b}))\}$ with probability at least $1 - \delta$ provided $(p-\epsilon)|F| > 3kd + 2 + \sqrt{6(k+1)|F|}$.*

3.3. Putting it together. We are now ready to describe the algorithm for solving the multivariate reconstruction problem. The algorithm has a preprocessing stage where it sets up k black boxes, and a query processing stage where it is given a query point $\hat{a} \in F^n$ and the black boxes compute $f_i(\hat{a})$.

Preprocessing Stage: Given: Oracle access to a black box B described by polynomials $f_1, \ldots, f_k$. Parameters k, ϵ, p, and δ.

Step 1: Pick $\hat{r}$ at random and $\hat{b}$ at random.

Step 2: Reconstruct, with confidence $1 - \delta$, the set $\{(f_1(\hat{r}), f_1(\hat{b})), \ldots, (f_k(\hat{r}), f_k(\hat{b}))\}$ using the algorithm of section 3.2.

Step 3: If the multiset $\{f_1(\hat{r}), \ldots, f_k(\hat{r})\}$ has two identical values, then output "failure". Else pass the reference point $\hat{r}$ and the values $f_1(\hat{r}), \ldots, f_k(\hat{r})$ to the Query Processing Stage.

Query Processing Stage: Given: Oracle access to a black box B, $\hat{b} \in F^n$, and parameters k, d, p, and δ. Additionally, reference point $\hat{r}$ and values $v_1, \ldots, v_k$ passed on by the Preprocessing Stage.

Step 1: Reconstruct with confidence δ the set $\{(f_1(\hat{r}), f_1(\hat{b})), \ldots, (f_k(\hat{r}), f_k(\hat{b}))\}$ using the algorithm of section 3.2.

Step 2: If the set $\{f_1(\hat{r}), \ldots, f_k(\hat{r})\}$ equals the set $\{v_1, \ldots, v_k\}$ then reorder the indices so that $f_i(\hat{r}) = v_i$ for every $i \in [k]$. If the sets are not identical, then report "failure".

Step 3: For every $i \in [k]$, the black box B_i outputs $f_i(b)$.

This yields the following theorem.

THEOREM 28. *Let B be an ϵ-noisy n-variate (k, d)-polynomial black box s.t.*

$$\Pr_{\hat{x} \in F^n} [\exists i \in [k], \text{ s.t. } B(\hat{x}) = f_i(\hat{x})] \geq 1 - \epsilon$$

$$\text{and } \forall i \in [k] \Pr_{\hat{x} \in F^n} [B(\hat{x}) = f_i(\hat{x})] \geq p > \epsilon.$$

Then, if $(p-\epsilon)|F| > 3kd+2+\sqrt{6(k+1)|F|}$, *there exists a randomized algorithm that takes as input a confidence parameter* δ *and with probability* $1-\delta$ *produces* k *black boxes* B_j *such that for every* $i \in [k]$ *there exists* $j \in [k]$ *s.t. for every input* $\hat{b} \in F^n$, *the black box* B_j *computes* $f_i(\hat{b})$ *with probability* $1-\delta$.

4. Applications. In this section, we describe the application of our techniques to curve fitting and bivariate polynomial factorization.

4.1. Curve fitting problems over discrete domains. In this subsection, we study the curve fitting problem over discrete domains. Given a set of m points, with integer coordinates, we show how to find a polynomial with integer coefficients that is Δ-close to all but an ϵ fraction of the points (if such a polynomial exists), where ϵ need only be less than $1/2$ (provided m is larger than $\frac{(4\Delta+1)d}{1-2\epsilon}$). Over Z_p (or over the integers) the problem can be formulated as follows.

Given: m pairs of points, $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ and ϵ, such that there exists a polynomial f, of degree at most d, such that for all but ϵm values of j in $[m]$

$$\exists i \in [-\Delta, \Delta] \text{ s.t. } y_j = f(x_j) + i$$

Problem: Find such an f.

Consider $f_i(x) \stackrel{\text{def}}{=} f(x) + i$, where $i \in [-\Delta, \Delta]$. Notice that all but an ϵ fraction of the points are described by the polynomial f_i's. Thus the above problem could be thought of as a reconstruction problem for an ϵ-noisy $(2\Delta+1, d)$-polynomial black box reconstruction problem. Lemma 21 can now be applied to this set of points to get the following result.

CLAIM 29. *If there exists an* i *such that the number of points for which* $y = f_i(x)$ *is strictly more than* $\epsilon m + kd$, *then we can find a small set of polynomials which includes* f_i.

The weakness of the above procedure is that it can only be guaranteed to succeed if ϵ is smaller than $\frac{1}{2\Delta+2}$, since only then can we guarantee the existence of an i such that the polynomial $f_i(x)$ is represented more often than the noise in the input set. We now present a variation of the above method which gets around this weakness and solves the curve fitting problem for strictly positive values of ϵ (independent of Δ) and in fact works for ϵ arbitrarily close to $1/2$.

The idea is that we can artificially decrease the influence of the *bad* points. To do this, we look at the following set of points: $\{(x_1^i, y_1^i), \ldots, (x_m^i, y_m^i)\}_{i=-\Delta}^{\Delta}$, where $x_j^i = x_j$ and $y_j^i = y_j - \Delta + i$. (From each point in the original sample, we generate $2\Delta+1$ points, by adding and subtracting up to Δ to the y coordinate of each point.) We show that these points represent the output of a (k, d)-algebraic black box for $k = \epsilon m + (4\Delta+1)d$.

Observe that the following conditions hold for the $(2\Delta+1)m$ points constructed above.

- There exists a polynomial $Q(x, y)$ of $\{(1, x), (d, y)\}$-weighted degree at most $\epsilon m + (4\Delta+1)d$ such that $Q(x, y) = 0$ for all the points. This is the polynomial: $Q(x, y) = W(x) \cdot \Pi_{i=-2\Delta}^{2\Delta} (y - f_i(x))$, where $f_i(x) = f(x) + i$ and $W(x)$ is the polynomial satisfying $W(x_j) = 0$ if $f_i(x_j) \neq y_j$ for any $i \in [-\Delta, \Delta]$. (Notice that the degree of W is ϵm.)
- At least $(1-\epsilon)m$ of the points satisfy $y = f(x)$. This is because for every point in the original (x_j, y_j) such that y_j is within Δ of $f(x_j)$ (and there were $(1-\epsilon)m$ such points), one of the new (x_j^i, y_j^i) pairs satisfies $y_j^i = f(x_j^i)$.

LEMMA 30. *Given m points $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, integer d, and $\epsilon < \frac{1}{2}$, there exists a polynomial time algorithm that can find all polynomials f of degree d such that f is Δ-close to all but an ϵ fraction of the points (x_j, y_j), provided $m > \frac{(4\Delta+1)d}{1-2\epsilon}$.*

Proof. Find a polynomial $\tilde{Q}(x, y)$ such that $\tilde{Q}(x, y) = 0$ for all the points $\{(x_j^i, y_j^i)\}_{j=1, i=-\Delta}^{m, \Delta}$ such that the degree of $\tilde{Q}$ is at most $\epsilon m + (4\Delta + 1)d$. If $\epsilon m + (4\Delta + 1)d < (1 - \epsilon)m$, then by Lemma 11 we know that for every candidate function f which forms an (ϵ, Δ) fit on the given points, $(y - f(x))$ divides $\tilde{Q}$. Thus factoring $\tilde{Q}$ will give us all the candidates. ∎

4.2. Reducing bivariate factoring to univariate factoring. In section 2.1, we saw how to reduce the problem of reconstructing total polynomial black boxes to the problem of factoring bivariate polynomials. In the specific case of univariate polynomial black boxes over finite fields, we will also reduce the reconstruction problem to that of factoring univariate polynomials into their irreducible factors. As an interesting consequence, we describe a simple way of reducing the problem of factoring special bivariate polynomials over finite fields to the problem of factoring univariate polynomials.

We first show how to reduce the reconstruction problem to that of factoring univariate polynomials. Suppose we have a black box which on input x outputs the (unordered) set $\{f_1(x), \ldots, f_k(x)\}$, where the f_i's are univariate polynomials, each of degree at most d. Sampling from the black box and interpolating, we can find the polynomial $f(x) = \prod_{i=1}^{k} f_i(x)$ explicitly (in terms of its coefficients). If somehow we could guarantee that at least one of the f_i's is irreducible, we could factor t to find f_i. Such a guarantee is not available, but we simulate it via randomization.

Let $\alpha(x) \in F[x]$ be a random degree d polynomial. We can convert the given set of sample points so that on each input x we have the (still unordered) set $\{g_1(x), \ldots, g_k(x)$: $g_i(x) = f_i(x) + \alpha(x)\}$. Each of the polynomials g_i is a random degree d polynomial (but they are not necessarily independent). We then use the fact that random polynomials over finite fields have a reasonable chance of being irreducible.

LEMMA 31 ([27, p. 84]). *The probability $P_q(d)$ that a random polynomial of degree d is irreducible over F_q is at least $\frac{1}{d}(1 - \frac{1}{q-1})$.*

We can thus interpolate (after sampling at $kd + 1$ points) and explicitly compute $g(x) = \prod_{i=1}^{k} g_i(x)$. We factor g into irreducible factors $r_1, \cdots, r_l$. For each factor r_j of g, we verify whether or not $r_j - \alpha$ is a candidate for one of the f_i's by checking that it evaluates to one of the outputs of the black box B on all the sampled points. By Lemma 31 we know that with nonnegligible probability g_i is irreducible and if this happens, we find g_i as one of the factors of g (i.e., as one of the r_j's). Subtracting α from g_i gives us f_i, which will pass the candidacy verification.

LEMMA 32. *If a degree d polynomial p agrees with one of the outputs of the black box on $kd + 1$ different x's, then p agrees with one of the outputs of the black box on all x's.*

Proof. If p agrees with one of the outputs of the black box on $kd + 1$ different x's, then by the pigeonhole principle there is a polynomial f_i which agrees with p on at least $d + 1$ different x's. Thus $p \equiv f_i$. ∎

Thus, no r_j which is not equal to one of the g_i's will pass the candidacy verification. By repeating this procedure enough times and outputting all the candidates, we can reconstruct all the polynomials $\{f_1, \ldots, f_k\}$. Straightforward analysis shows that the expected number of times that we need to repeat the process (choose random α) is $O(k/P_q(d))$. Refining the analysis, we can show that $O(\ln k/P_q(d))$ times suffice.

From the above, we get the following algorithm for finding the monic linear factors of a bivariate polynomial $Q(x, y)$.

```
program Simple Factor
repeat O( ln k/P_q(d) ) times
   pick a random degree d polynomial
         α(x) over F
   factor Q(x, α(x))
   for every factor g(x) of Q(x, α(x))
      if (y + g(x) − α(x)) divides Q(x, y)
         output (y + g(x) − α(x))
end
```

CLAIM 33. *Given a bivariate polynomial $Q(x, y)$, over a finite field F, of total degree at most kd, the algorithm* **Simple Factor** *finds all the linear and monic factors of $Q(x, y)$.*

We next extend this mechanism and apply the reconstruction mechanism of section 2.2 to the problem of finding the factors of $Q(x, y)$ which are monic and of constant degree in y. Our mechanism tries to isolate some factor $A(x, y)$ of $Q(x, y)$ of the form

$$A(x, y) = y^c + a_{c-1}(x)y^{c-1} + \cdots + a_0(x),$$

where the a_i's are polynomials in x of degree at most d (and c is a constant).

Let $Q(x, y)$ be a polynomial of $\{(1, x), (d, y)\}$-weighted degree kd. For each $i \in [c]$ we construct a program P_i which is supposed to be a (K, d)-algebraic box for a_i, for some $K \leq 2ikd\binom{k}{c}$. We then use our reconstruction procedure (Theorem 22) to produce, for each $i \in [c]$, a list of at most K polynomials which contains a_i. This, in turn, gives a set of at most K^c polynomials in x and y which contains $A(x, y)$. $A(x, y)$ can be isolated from this set by exhaustive search. The running time of this algorithm is thus some polynomial in $(kd)^{c^2}$.

The program P_i for a_i works as follows on input x_1.

- P_i constructs the polynomial $Q_{x_1}(y) \equiv Q(x_1, y)$ (which is a polynomial in y) and factors Q_{x_1}.
- Let S be the set of factors of Q_{x_1}. (S contains polynomials in y.)
- Let S^c be the set of polynomials of degree c obtained by taking products of polynomials in S.
- P_i picks a random polynomial f in S^c and outputs the coefficient of y^i in f.

We now show that P_i is a $(2ikd\binom{k}{c}, d)$-algebraic black box described by some polynomial $Q^*(x, y)$ such that $y - a_i(x)$ divides $Q^*(x, y)$. Let $Q(x, y) = q_k(x)y^k + q_{k-1}y^{k-1} + \cdots + q_0(x)$. Over the algebraic closure of the quotient ring of polynomials in x, $Q(x, y)$ factors into linear factors in y; let this factorization be

$$Q(x, y) = q_k(x)(y - b_1(x))(y - b_2(x)) \cdots (y - b_k(x)).$$

(The $b_i(x)$ are some functions of x, but not necessarily polynomials.) For $T \subset [k]$, $|T| = c$, let $\sigma_{T,i}(x) \stackrel{\text{def}}{=} \sum_{S \subset T, |S|=i} \prod_{l \in S} b_l(x)$. Notice that the function $a_i(x)$ that we are interested in is actually $\sigma_{T,i}(x)$ for some T. Notice further that the output of the program P_i is always $\sigma_{T,i}(x)$ for some T (though this T is some arbitrary subset of $[k]$). Thus the input/output pairs (x, y) of the program P_i always satisfy $\prod_{T \subset [k], |T|=c}(y - \sigma_{T,i}(x)) = 0$. Unfortunately, $\sigma_{T,i}(x)$ need not be a polynomial in x. So we are not done

yet. We will show that $Q^*(x,y) \stackrel{\text{def}}{=} (q_k(x,y)^N) \prod_{T \subset [k], |T|=c} (y - \sigma_{T,i}(x))$ is actually a polynomial in x and y of $\{(1,x),(d,y)\}$-weighted degree at most Ki, where $N = i\binom{k}{c}$. To see this, consider the coefficient of y^j in $Q^*(x,y)$. This is $q_k(x)^N$ times some polynomial in $b_1(x), \ldots, b_k(x)$, denoted $g_j(b_1(x), \ldots, b_k(x))$. By the definition of Q^*, we notice that g_j is a symmetric polynomial in $b_1(x), \ldots, b_k(x)$ of degree at most $i\binom{k}{c}$. We now invoke the "fundamental theorem of symmetric polynomials" [27, pp. 29–30], which states that a symmetric polynomial of degree D in variables $z_1, \ldots, z_k$ can be expressed as a polynomial of degree at most D in the *primitive* symmetric functions in $z_1, \ldots, z_k$. In our case this translates into saying that there exists some polynomial $\tilde{g}_j$ of degree at most N s.t. $g_j(b_1(x), \ldots, b_k(x)) = \tilde{g}_j(\frac{q_{k-1}(x)}{q_k(x)}, \ldots, \frac{q_0(x)}{q_k(x)})$ (since the primitive symmetric functions in $b_1(x), \ldots, b_k(x)$ are actually $\frac{q_{k-1}(x)}{q_k(x)}, \ldots, \frac{q_0(x)}{q_k(x)}$). Thus we find that the coefficient of y^j in $Q^*(x,y)$ is a polynomial in x of degree at most $ikd\binom{k}{c}$. The claimed bound on the degree of Q^* now follows easily.

Thus we get the following lemma.

LEMMA 34. *Given a polynomial $Q(x,y)$ of $\{(1,x),(d,y)\}$-weighted degree kd, there is an algorithm that runs in time polynomial in $(kd)^{c^2}$ which finds all factors of Q that are monic and of degree c in y.*

Appendix A.

LEMMA 35. *Given $S \subset H$, with $|S|/|H| = p$, if $|H| \geq \frac{2}{\delta}\binom{M}{2}$, and $M \geq \frac{2}{p}(kd + \ln \frac{2}{\delta})$, then the probability that M points $x_1, \ldots, x_M$ chosen uniformly at random from H turn out to be distinct and satisfy $|\{j | x_j \in S\}| \geq kd$ is at least $1 - \delta$.*

Proof. First observe that for a given $i \neq j$, the probability that $x_i = x_j$ is exactly $1/|H|$. Thus the probability that there exists i, j s.t. $x_i = x_j$ is at most $\binom{M}{2}/|H| \leq \delta/2$.

Now, for the second part, we use Chernoff bounds (in particular, we use a bound from [31, Theorem 4.2]). Let X denote the number of elements j s.t. $x_j \in S$. Let $\mu = Mp$ denote the expected value of X. Then $\Pr[X \leq K] \leq \exp(-(\mu - K)^2/2\mu)$. Plugging in the values of $\mu = 2(kd + \ln \frac{2}{\delta})$ and $K = kd$ and simplifying, we find that $\exp(-(\mu - K)^2/2\mu) \leq \delta/2$.

Thus the probability that either of the above two events happens is bounded by at most δ as desired. ∎

LEMMA 36. *Given $S, E \subset H$, with $|S|/|H| = p$ and $|E|/|H| = \epsilon$ if $|H| \geq \frac{3}{\delta}\binom{M}{2}$, and $M \geq \frac{k_1 kd}{p - k_1\epsilon} + \frac{16}{(p-k_1\epsilon)^2} \ln \frac{3}{\delta}$, then the probability that M points $x_1, \ldots, x_M$ chosen uniformly at random from H turn out to be distinct and satisfy $|\{j | x_j \in S\}| > M(p + k_1\epsilon)/2 + k_1 kd/2$ and $|\{j | x_j \in E\}| < M(p/k_1 + \epsilon)/2 - kd/2$ is at least $1 - \delta$.*

Proof. First we argue as above that the probability that the x_j's are not distinct is at most $\delta/3$.

Now for the second part we again use Chernoff bounds (this time we use the bounds from [31, Theorems 4.2 and 4.3]). Let X denote the number of elements j s.t. $x_j \in S$. The above-mentioned bounds translate to show that the probability that the number of points from S is less than $Mp - \lambda\sqrt{Mp}$ is at most $\exp(-\lambda^2/2)$. Similarly, the probability that the number of points from E turns out to be more than $M\epsilon + \lambda\sqrt{M\epsilon}$ is bounded by at most $\exp(-\lambda^2/4)$. Each of these probabilities is at most $\delta/3$ if we choose $\lambda = 2\sqrt{\ln \frac{3}{\delta}}$. The lemma now follows from the fact that for the chosen value of M, we have that $M\epsilon + \lambda\sqrt{M} < M(p/k_1 + \epsilon)/2 - kd/2$ and $Mp + \lambda\sqrt{M} > M(p + k_1\epsilon)/2 + k_1 kd/2$. ∎

Acknowledgments. We are very grateful to Avi Wigderson for asking a question that started us down this line of research and for helpful discussions. We are very grateful to Umesh Vazirani for his enthusiasm, his valuable opinions and suggestions, and the time that he spent with us discussing this work. We thank Ronen Basri, Oded Goldreich, and Mike Kearns for their comments on the writeup of this paper. We thank Joel Friedman for technical discussions about questions related to the topics of this paper. We also thank the anonymous referees for their extensive reports and for catching many blatant as well as subtle errors from an earlier version of this paper.

REFERENCES

[1] D. Beaver and J. Feigenbaum, *Hiding instance in multioracle queries*, in Proc. ACM Symposium on Theoretical Aspects of Computer Science, 1990.
[2] M. Ben-Or, *Probabilistic algorithms in finite fields*, in Proc. 22nd IEEE Symposium on Foundations of Computer Science, 1981, pp. 394–398.
[3] M. Ben-Or and P. Tiwari, *A deterministic algorithm for sparse multivariate polynomial interpolation*, in Proc. 20th ACM Symposium on Theory of Computing, 1988, pp. 301–309.
[4] E. Berlekamp, *Factoring polynomials over large finite fields*, Math. Comp., 24 (1970), p. 713.
[5] E. Berlekamp, *Bounded distance +1 soft-decision Reed-Solomon decoding*, in IEEE Trans. Inform. Theory, 42 (1996), pp. 704–720.
[6] A. Blum and P. Chalasani, *Learning switching concepts*, in Proc. 5th Annual ACM Workshop on Computational Learning Theory, 1992, pp. 231–242.
[7] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595.
[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, *Occam's razor*, Inform. Process. Lett., 24 (1987), pp. 377–380.
[9] J. Y. Cai and L. Hemachandra, *A note on enumerative counting*, Inform. Process. Lett., 38 (1991), pp. 215–219.
[10] J. Canny, *Finding edges and lines in images*, Artificial Intelligence Laboratory Report, AI-TR-720, MIT, Cambridge, MA, 1983.
[11] D. Coppersmith, Personal communication to Ronitt Rubinfeld, Fall 1990.
[12] J. von zur Gathen, *Algebraic complexity theory*, Ann. Rev. Comput. Sci., 3 (1988), pp. 317–347.
[13] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson, *Self-testing/correcting for polynomials and for approximate functions*, in Proc. 23rd ACM Symposium on Theory of Computing, 1991, pp. 32–42.
[14] P. Gemmell and M. Sudan, *Highly resilient correctors for polynomials*, Inform. Process. Lett., 43 (1992), pp. 169–174.
[15] O. Goldreich and L. Levin, *A hard-core predicate for any one-way function*, in Proc. 21st ACM Symposium on Theory of Computing, 1989.
[16] O. Goldreich, R. Rubinfeld, and M. Sudan, *Learning polynomials with queries: The highly noisy case*, in Proc. 36th IEEE Symposium on Foundations of Computer Science, 1995, pp. 294–303.
[17] D. Grigoriev, *Factorization of polynomials over a finite field and the solution of systems of algebraic equations*, Zap. Nauchn. Sem. Lenningradskogo Otdel. Mat. Inst. Steklov. AN SSSR, 137 (1984), pp. 20–79 (in translation).
[18] D. Grigoriev and A. Chistov, *Fast decomposition of polynomials into irreducible ones and the solution of systems of algebraic equations*, Soviet Math. Dokl., 29 (1984), pp. 380–383.
[19] D. Grigoriev and M. Karpinski, *Algorithms for Sparse Rational Interpolation*, ICSI Technical Report TR-91-011, ICSI, Berkeley, CA, 1991.
[20] D. Grigoriev, M. Karpinski, and M. F. Singer, *Interpolation of Sparse Rational Functions without Knowing Bounds on Exponents*, Report No. 8539-CS, Institut für Informatik der Universität Bonn, 1989.
[21] D. Grigoriev, and M. Karpinski, and M. F. Singer, *Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields*, SIAM J. Comput., 19 (1990), pp. 1059–1063.
[22] D. Haussler, *Decision theoretic generalizations of the PAC model for neural net and other learning applications*, Inform. and Comput., 100 (1992), pp. 78–150.
[23] J. Henderson and R. Quandt, *Microeconomic Theory*, McGraw Hill, New York, 1958, 1971.

[24] E. Kaltofen, *A polynomial-time reduction from bivariate to univariate integral polynomial factorization*, in 23rd Annual IEEE Symposium on Foundations of Computer Science, 1982, pp. 57–64.
[25] E. Kaltofen and B. Trager, *Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators*, in 29th Annual IEEE Symposium on Foundations of Computer Science, 1988, pp. 296–305.
[26] E. Kushilevitz and Y. Mansour, *Learning decision trees using the Fourier spectrum*, in Proc. 23rd ACM Symposium on Theory of Computing, 1991.
[27] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*, Cambridge University Press, UK, 1986.
[28] R. Lipton, *New directions in testing*, in Distributed Computing and Cryptography, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 2, AMS, Providence, RI, 1991, pp. 191–202.
[29] A. K. Lenstra, H. W. Lenstra, and L. Lovasz, *Factoring polynomials with rational coefficients*, Math. Ann., 261 (1982), pp. 515–534.
[30] D. Marr and E. Hildreth, *Theory of edge detection*, Proc. Roy. Soc. London, B207 (1980), pp. 187–217.
[31] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, UK, 1995.
[32] R. Rivest, *Learning decision lists*, Machine Learning, 2 (1987), pp. 229–246.
[33] T. J. Rivlin, *An Introduction of the Approximation of Functions*, Dover, New York, 1969.
[34] R. Rubinfeld and R. Zippel, *A new modular interpolation algorithm for factoring multivariate polynomials*, in Proc. Algorithmic Number Theory Symposium, 1994.
[35] M. Sudan, *Decoding of Reed Solomon codes beyond the error-correction bound*, J. Complexity, 13 (1997), pp. 180–193.
[36] L. Valiant, *A theory of the learnable*, Comm. ACM, 27 (1984), pp. 1134–1142.
[37] B. L. Van der Waerden, *Algebra*, Vol. 1, Frederick Ungar Publishing Co., Inc., p. 82.
[38] R. J. Walker, *Algebraic Curves*, Princeton University Press, Princeton, NJ, 1950.
[39] L. Welch and E. Berlekamp, *Error Correction of Algebraic Block Codes*, U.S. Patent Number 4,633,470, issued December 1986.
[40] R. E. Zippel, *Probabilistic algorithms for sparse polynomials*, in Proc. European Conference on Symbolic and Algebraic Manipulation '79, Lecture Notes in Comput. Sci. 72, Springer-Verlag, New York, 1979, pp. 216–226.
[41] R. E. Zippel, *Interpolating polynomials from their values*, J. Symbol. Comput., 9 (1990), pp. 375–403.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 511–524

OPTIMAL BROADCAST WITH PARTIAL KNOWLEDGE*

BARUCH AWERBUCH†, ISRAEL CIDON‡, SHAY KUTTEN§, YISHAY MANSOUR¶, AND DAVID PELEG‖

Abstract. This work is concerned with the problem of broadcasting a large message efficiently when each processor has partial prior knowledge about the contents of the broadcast message. The partial information held by the processors might be out of date or otherwise erroneous, and consequently, different processors may hold conflicting information. Tight bounds are established for broadcast under such conditions, and applications of the broadcast protocol to other distributed computing problems are discussed.

Key words. distributed algorithms, topology update, error correcting code, universal hash, self stabilization

AMS subject classifications. 94A, 94B, 68Q22

PII. S0097539795279931

1. Introduction.

1.1. Motivation. Many tasks in distributed computing deal with concurrently maintaining the "view" of a common object in many separate sites of a distributed system. This object may be the topology of a communication network (in which case the view is a description of the underlying network graph), or certain resources held at the system sites (in which case the view is an inventory listing the resources held at each site), or even a general database. The objects considered here are dynamic in nature, and are subject to occasional changes (e.g., a link fails, a resource unit is consumed or released, a database record is modified). It is thus necessary to have an efficient mechanism for maintaining consistent and updated views of the object at the different sites.

One obvious algorithm for maintaining updated views of a distributed object is the *full broadcast* algorithm. This algorithm is based on initiating a broadcast of the entire view of the object whenever a change occurs. Due to the possibility of message pipelining, the time complexity of this algorithm is relatively low. On the other hand, this algorithm might be very wasteful in communication, since the object may be rather large.

Consequently, it is clear that a successful consistency maintenance strategy should strive to utilize the fact that the processors already have a correct picture of "most"

*Received by the editors January 10, 1995; accepted for publication (in revised form) April 10, 1996; published electronically July 28, 1998.

http://www.siam.org/journals/sicomp/28-2/27993.html

†The Johns Hopkins University, Baltimore, MD 21218 and MIT Laboratory for Computer Science (baruch@blaze.cs.jhu.edu). This research was supported by Air Force contract TNDGAFOSR-86-0078, ARPA/Army contract DABT63-93-C-0038, ARO contract DAAL03-86-K-0171, NSF contract 9114440-CCR, DARPA contract N00014-J-92-1799, and a special grant from IBM.

‡Faculty of Electrical Engineering, Technion, Haifa 32000, Israel (cidon@ee.technion.ac.il).

§Department of Industrial Engineering and Management, Technion, Haifa 32000, Israel and IBM T.J. Watson Research Center, Yorktown Heights, NY 10598 (kutten@ie.technion.ac.il).

¶Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (mansour@math.tau.ac.il).

‖Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel (peleg@wisdom.weizmann.ac.il). This research was supported in part by an Allon Fellowship, by a Walter and Elise Haas Career Development Award, and by a Bantrell Fellowship.

of the object and need to be informed of relatively few changes. Viewed from this angle, the problem can be thought of as having to broadcast the entire view of the object, while taking advantage of prior partial knowledge available to the processors of the system.

On the other extreme there is the *incremental update* strategy, in which only "necessary" information is transmitted. This strategy is at the heart of the algorithms suggested for handling the topology update problem [ACK90, MRR80, SG89, BGJ$^+$85]. Unfortunately, it is not obvious how to employ information pipelining with this method, as demonstrated in the sequel. This increases the time complexity.

The purpose of this work is to study the problem of updating a distributed database, under minimal assumptions. That is, we do not assume any initial coordination and allow only a small amount of space. Under such conditions, we look for efficient solutions to the problem with respect to communication and time overheads. In this setting, it turns out that the main bottleneck of the database update problem can be characterized as a fairly simple "communication complexity" problem, called *broadcast with partial knowledge*.

1.2. The model and the problem. The *broadcast with partial knowledge* problem can be formulated as follows. Consider an asynchronous communication network, consisting of $n+1$ processors, $p_0, \ldots, p_n$; each processor p_i has an m-bit *local input* w_i, and processor p_0 is distinguished as the *broadcaster*. In a correct solution to the problem all the processors write in their local output the value of the broadcaster's input, $w = w_0$.

Without loss of generality we assume that the network is arranged in a line. That is, each processor p_i (for $1 < i < n$) is connected only to p_{i-1} and p_{i+1}. The algorithm can be easily adapted to be executed on trees, rather than on a line (which is the "worst-case" tree) as described here. Thus, in an arbitrary topology network, performing the algorithm on a shortest-path tree yields optimal message and time complexities. (Our results are expressed in terms of n, as well as of other parameters; in the case of general networks n is replaced by the diameter of the network.)

This formulation of the problem can be interpreted as follows. The input w_i is stored at processor p_i and describes the local representation of the object at processor p_i. The correct description of the object is $w = w_0$, held by the broadcaster. The local descriptions w_i may differ from the correct one as a result of changes in the object. In particular, every two processors may have different descriptions due to different messages they got from the broadcaster in the past, as a result of message losses, topology changes, and the asynchronous nature of the network. Our goal is to inform all the processors throughout the network about the correct view of the object w, and to use the processor's local inputs given to each processor in order to minimize the time and communication complexities.

In this paper, we provide a randomized solution for the hardest version of this problem, in which each processor only knows its own input, and has no information regarding inputs of other processors. A weaker version of the problem is based on making the rather strong "neighbor knowledge" assumption, namely, assuming that each processor correctly knows the inputs of all its neighbors, in addition to its own. This assumption is justified in [ACK90], where it is shown that neighbor knowledge comes for free in the context of database and topology update protocols. Even for this weaker problem, none of the previously known solutions are efficient in *both* communication and time. (Following the original version of the current paper [ACK$^+$91], a deterministic algorithm was given for the weaker problem [AS91]. The complexity

of that algorithm is larger than that of the randomized algorithm presented here by a polylogarithmic factor.)

In order to quantify the possibility of exploiting local knowledge, we first introduce a new measure that captures the level of "information" of the knowledge held by each processor. Let the *discrepancy* δ_i of the input w_i held by processor p_i be the number of bits in which w_i, the local description at p_i, differs from the broadcaster's input w, which is the correct description of the object. Define also the *total discrepancy* $\Delta = \sum_i \delta_i$, the *average discrepancy* $\delta_{av} = \Delta/n$, and the *maximum discrepancy* $\delta_{max} = \max_i\{\delta_i\}$.

Our goal is to study the relationships between these discrepancies and the complexity of broadcast algorithms, following the intuition that the complexity of broadcast protocols should be proportional to discrepancy of processors' inputs; i.e., if the views of most processors are "almost correct," then the overhead of the protocol should be small. We therefore express the communication and time complexities of our solution as a function of m, n, and δ_{av}. The complexities are measured in the bit complexity model.

1.3. Basic solutions. The first obvious solution to the *broadcast with partial knowledge* problem is the aforementioned *full broadcast* protocol, which is wasteful in communication, i.e., requires $\Omega(nm)$ bits. On the other hand it is rather fast, since the broadcast can be done in a pipelined fashion and thus can terminate in $O(n + m)$ time. Thus, one would like to improve on this algorithm with respect to communication complexity, aiming towards reducing this complexity to be close to the total discrepancy Δ, while maintaining near-optimal time complexity.

The *incremental update* strategy proposed in [ACK90] poses an alternative approach. It can be applied only under the strong assumption of "neighbor knowledge," where each node is assumed to "know" the value of the database at its neighbor. The essence of this strategy is that a processor with "correct" view transmits to its neighbor a "correction" list, which contains all the positions where the neighbor's input is erroneous. When the neighbor receives all the corrections, it can assume that the rest of its input is correct and start using it for correcting its own neighbors who are further away from the source. In this algorithm, a "correction wave" propagates through the network from the source, until all nodes are corrected. Note that there is almost no pipelining possible in this algorithm as described above, since a node can start transmitting only after it is done with the receiving. Even in the simple case of a path network the protocol may require $\Omega(\Delta)$ time. As mentioned above, in the follow-up paper [AS91] the complexity of this strategy (for the "neighbor knowledge" variant of the problem) was improved significantly (although still not matching the lower bound), using a very sophisticated partitioning of the information and a recursive implementation.

1.4. Our results. In this paper, we provide an efficient randomized solution to the broadcast with partial knowledge problem. It has success probability at least $1 - \epsilon$, and uses $O(\Delta \log m + n \log \frac{n}{\epsilon})$ communication and $O(n + m)$ time, where ϵ is a parameter to the algorithm. Note that in all cases, we allow the inputs stored at the various processors to differ in arbitrary ways, subject to the discrepancy constraints.

Our upper bounds are derived using linear codes. Such codes were used before in constructing distributed algorithms for solving various problems. Metzner [Met84] uses Reed–Solomon and random codes to achieve efficient retransmission protocols in a complete network. Ben-Or, Goldwasser, and Wigderson [BOGW88] use Bose–Chaudhuri Hocquenghem (BCH) codes to guarantee privacy in a malicious environ-

TABLE 1
Comparison of protocols and lower bounds.

Algorithm	Communication	Time	Assumptions
Full broadcast (folklore)	nm	$n+m$	
Incr. Update [ACK90]	$n+\Delta \log m$	$n+\Delta \log m$	neighbor knowledge
Our algorithm	$\Delta \log m + n \log \frac{n}{\epsilon}$	$n+\log \frac{n}{\epsilon} + \min\{m, \Delta \log m\}$	
Our lower bound	$n+n\delta_{max} \log(m/\delta_{max})$	$n+\delta_{max} \log(m/\delta_{max})$	

TABLE 2
Applications of partial knowledge broadcast protocols to topology update.

Reference	Amortized Commun.	Quiescence time
Full broadcast [AAG87]	VE	$V+E$
Incr. update [ACK90]	$V \log E$	$V^2 \cdot \log E$
Our upper bound	$V \log E$	$E+V \log V$
Lower bound	$V \log E$	E

ment. Rabin [Rab89] uses codes to achieve a reliable fault-tolerant routing with a low overhead. Another closely related concept is that of source coding with side information [SW73].

Using simple arguments from information theory and communication complexity theory, we are able to show that our upper bounds are almost tight. We argue that when the average discrepancy is δ_{max}, the communication complexity is at least $\Omega(\Delta \log(\frac{m}{\delta_{max}}))$ and the time complexity is at least $\Omega(n+\delta_{max} \log(\frac{m}{\delta_{max}}))$. We also argue that in the case when no information is known about the discrepancies, any deterministic protocol would send $\Omega(nm)$ bits, even if there are *no* discrepancies at all.

The comparison of our protocols, previous results, and the lower bounds is given in Table 1.

1.5. Applications. One application of our work is to the classical network problem of topology update. This task is at the heart of many practical network protocols [MRR80, BGJ$^+$85, ACG$^+$90]. The problem can be formulated as follows. Initially, each processor is aware of the status of its adjacent links, i.e., whether each link is up or down, but is unaware of the status of other links. The purpose of the protocol is to supply each processor with this global link status information.

The topology update algorithm of [ACK90] is based on the *incremental update* strategy. The possibility of recurring network partitions and reconnections significantly complicates implementation of this strategy. Nevertheless, the resulting broadcast procedure is efficient in terms of communication (although not in time) and leads to essentially communication-optimal topology update protocols [ACK90].

A consequence of [ACK90] that is most significant for our purposes is the observation that it is possible to relate the complexities of the problem of broadcasting with partial knowledge to those of topology update, effectively reducing the former problem to the latter. Namely, given *any* solution for the broadcast with partial knowledge problem, one can construct a topology update protocol with lower or equal overheads in both communication and time.

Table 2 summarizes the complexities of protocols to the topology update task obtained by applying various broadcast with partial knowledge algorithms (with V, E denoting the number of vertices and edges in the network, respectively).

It is worth pointing out that our complexity results are presented in the bit com-

plexity model, whereas the results in [ACK90] are presented in the message complexity model which charges only one complexity unit for a message of size $O(\log n)$ bits.

Our algorithm may also be applicable for dealing with the issue of self-stabilization [Dij74, AKY90, KP90, APV91]. The self-stabilization approach is directed at dealing with intermittent faults that may change the memory contents of nodes and cause inconsistency between the local states of nodes. Dijkstra's example [Dij74] is that of a token passing system, where it is required that exactly one of the nodes holds a *token* at any given time. The faults may cause an illegal situation in which no node holds a token (each "assumes" that some other has it) or alternatively, that several nodes hold a token. Overcoming such faults requires the nodes to continuously check the states of their neighbors and possibly to correct them when necessary. It is conceivable that the faults cause only partial changes in memory, so our algorithm can be used to mend the situation. Note that in this context, it is essential that we do not make the "neighbor knowledge" assumption.

1.6. Organization of the paper. The rest of the paper is structured as follows. In section 2 we review for later use some necessary material concerning universal hash functions and coding theory. In section 3, we present our upper bound (algorithm BPart). Finally, lower bounds on the problem are established in section 4.

2. Preliminaries.

2.1. Universal hash functions. Universal hash functions have found many interesting applications since their introduction by Wegman and Carter [WC79]. A family of functions $\mathcal{F} = \{h : A \to B\}$ is called a *universal hash function* if for any $a_1 \neq a_2 \in A$ and $b_1, b_2 \in B$ the following holds:

$$\text{Prob}[h(a_1) = b_1 \text{ and } h(a_2) = b_2] = \frac{1}{|B|^2},$$

where the probability is taken over the possible choices of h, which is randomly and uniformly chosen from $\mathcal{F}$.

There are many families of simple universal hash functions. One example can be constructed as follows. Let p be a prime and let $B = Z_p$. (Note $|B| = p$.) Then

$$H = \{h_{\alpha,\beta}(x) = (\alpha x + \beta) \bmod p \mid \alpha, \beta \in Z_p\}$$

is a family of universal hash functions.

In the above example the encoding of a hash function requires only two elements from Z_p, and also p; therefore we can describe such a hash function using only $O(\log |B|)$ bits. (Note that the encoding of h does not depend on A.) Later, when using a universal hash function, it is assumed that it can be represented with $O(\log |B|)$ bits.

Another way to view the parameters is the following. We are interested in a family of universal hash functions $\mathcal{F}_\epsilon$ that has the following property: given any two distinct elements, the probability that a random hash function $h \in \mathcal{F}_\epsilon$ maps them to the same point is bounded by ϵ. From the properties of the universal hash function this occurs with probability $1/|B|$. Therefore, choosing $\epsilon = 1/|B|$, we conclude that there is a family of hash functions $\mathcal{F}_\epsilon$ whose encoding size is $O(\log \frac{1}{\epsilon})$.

2.2. Information theoretic background. The tools developed later on are based on some basic results from coding theory. A code $C_{m,d} : \{0,1\}^m \mapsto \{0,1\}^{m+r}$ is a mapping that transforms an input word $w \in \{0,1\}^m$ into a codeword $C_{m,d}(w) =$

$\hat{w} \in \{0,1\}^{m+r}$. The codes considered in this paper are standard "check bit" codes; namely, the resulting codeword $\hat{w}$ is assumed to be of the form $\hat{w} = w\|\rho$, where $\rho \in \{0,1\}^r$ is a "trail" of r check bits concatenated to the input word w, called the *syndrome*. Denote by the check bit syndrome that the code $C_{m,d}$ attaches to a word w by $C^*_{m,d}(w)$. The lengths of the entire codeword and the check bit syndrome are denoted in the sequel by $|C_{m,d}(w)|$ and $|C^*_{m,d}(w)|$, respectively.

A code $C_{m,d}$ is said to be *d-correcting* if the original word w can be correctly decoded from any word z that differs from the codeword $C_{m,d}(w)$ in no more than d places.

The following theorem states the properties possessed by the code necessary for our purposes.

THEOREM 2.1. *For any m and $d \leq m/3$, there exists a check-bit code $C_{m,d}$ with the following properties.*

1. *The check bit syndrome is of length $|C^*_{m,d}(w)| = O(d \log m)$.*
2. *The code $C_{m,d}$ is d-correcting.*
3. *The encoding and decoding operations ($C_{m,d}$ and $C^{-1}_{m,d}$, respectively) require time polynomial in m and d.*

In order to show the theorem, we can slightly modify BCH codes, so they will have all the above properties. It is well known that the decoding and encoding of BCH codes can be accomplished in polynomial time and that the length of the check bit syndrome is $O(d \log m)$. The only property that we need to comment about is the use of arbitrary m. The code words in BCH codes are of length $2^k - 1$. We simply have to extend our input (e.g., by padding zeros) to the appropriate size. When encoding, we first extend the input $w \in \{0,1\}^m$ to $2^k - 1 - |C^*_{m,d}(w)|$ bits and then perform the encoding. After the decoding, the padding bits will be removed.

All codes $C_{m,d}$ referred to later on in the paper are meant to be check bit codes that satisfy the properties in Theorem 2.1. The subscripts m, d are omitted whenever m and d are clear from the context.

3. Upper bounds. We develop our solution in a modular fashion via a number of steps. The first step is a simple deterministic algorithm named MAXIMUM, presented in section 3.1, which is based on the assumption that the maximum discrepancy δ_{max} is known to the broadcaster. Next, section 3.2 presents the algorithm AVERAGE, which assumes only knowledge of the average discrepancy. Then, in section 3.3 it is shown that the assumptions about knowledge of the discrepancy can be eliminated at the cost of increasing the time complexity by a factor of $\log \delta_{av}$. Finally, in section 3.4 it is shown that the algorithm can be condensed, so that its time complexity becomes optimal again.

3.1. Algorithm MAXIMUM. This section handles broadcast in the case where the maximum discrepancy δ_{max} is known and presents a straightforward broadcasting algorithm MAXIMUM, which assumes that the broadcaster "knows" δ_{max}. The algorithm requires $O(n\delta_{max} \log m)$ communication and $O(n + \delta_{max} \log m)$ time.

We should note that this algorithm is not efficient, since the maximum discrepancy can be very far from the average discrepancy. This algorithm is presented in order to be used in the next section as a subroutine.

For simplicity, it is assumed that the network is a simple path; namely, the $n+1$ processors $p_0, \ldots, p_n$ are arranged on a line, with a bidirectional link connecting processor p_i to processor p_{i+1} for every $0 \leq i < n$. Note that this does not restrict generality in any way, since the path is the worst topology for broadcast, and moreover, there exists an easy transformation from every other network to a path network by

using a depth-first tour [Eve79].

Algorithm MAXIMUM works as follows: the *broadcaster* encodes the broadcast message w using the code $C = C_{m,\delta_{max}}$. (Note that this code C is fixed and known to all other processors.) The broadcaster broadcasts only the check bit syndrome $C^*(w)$. The broadcasting proceeds in full pipelining. That is, each processor p_i for $i < n$ that receives the first bit of $C^*(w)$ immediately forwards it to processor p_{i+1}, without waiting for the entire value of $C^*(w)$.

Once a processor p_i has received the complete message $\rho = C^*(w)$, it concatenates it to its own input w_i, thus obtaining a complete (but possibly corrupted) codeword $\hat{w}_i = w_i \| \rho$ and decodes this codeword by computing $o_i = C^{-1}(\hat{w}_i)$, which is taken to be the output.

LEMMA 3.1. *If the input w_i of processor p_i is different from w in at most δ_{max} places, then $o_i = w$.*

Proof. Consider the word o_i output by processor p_i. As $\delta_i \leq \delta_{max}$, it follows that $\hat{w}_i = w_i \| \rho$ differs from $\hat{w} = w \| \rho$ in at most δ_{max} places. Since the code C is δ_{max}-correcting, it follows that $o_i = C^{-1}(\hat{w}_i) = C^{-1}(\hat{w}) = w$. ∎

LEMMA 3.2. *The time complexity of algorithm* MAXIMUM *is $n + O(\delta_{max} \cdot \log m)$.*

Proof. The algorithm broadcasts the message $\rho = C^*(w)$ in full pipelining. Hence the first bit of ρ reaches the last processor, p_n, by time n, and the entire message reaches p_n by time $n + |C^*(w)|$. The lemma follows since $|C^*(w)| = O(\delta_{max} \cdot \log m)$. ∎

LEMMA 3.3. *The communication complexity of algorithm* MAXIMUM *is $O(n \cdot \delta_{max} \log m)$.*

Proof. The message $C^*(w)$ traverses each edge exactly once. Therefore, the communication complexity is $n \cdot |C^*(w)| = n \cdot O(\delta_{max} \cdot \log m)$. ∎

We complete the description by noting that both the time and communication complexities can be improved for large δ_{max}. Specifically, if $\delta_{max} \log m > m$, then a full broadcast of the information is more efficient (namely, send w to all the processors). Therefore we have the following theorem.

THEOREM 3.4. *Given the value of δ_{max}, there is a deterministic algorithm for performing broadcast with partial information that requires $n + O(\min\{m, \delta_{max} \cdot \log m\})$ time and has communication complexity $O(n \cdot \min\{m, \delta_{max} \cdot \log m\})$.*

A similar result holds when the broadcaster knows only an upper bound d on the discrepancies, where the same complexities hold except with d replacing δ_{max}. When the upper bound is "accurate," namely, $d = O(\delta_{max})$, the complexities remain the same.

Note that the communication complexity of this algorithm is not good when there are differences between the discrepancies of nodes. For example, consider the case in which one has a high discrepancy. This forces the algorithm to send a long check bit syndrome also to the other nodes on the way, although they have low discrepancies. The algorithm of the next section manages to fix this problem, even though it relies on a weaker assumption.

3.2. Algorithm AVERAGE**.** In this section we replace the assumption of a known δ_{max} with the assumption that only the average discrepancy δ_{av} is known. Note that no assumptions are made about how the discrepancies are distributed. In particular, it may be that some processors have large discrepancies while others have the correct value. For simplicity of notation, we assume throughout the section that $\delta_{av} \geq 1$, or $\Delta \geq n$.

The broadcast algorithm AVERAGE presented in this section is randomized; i.e., it

guarantees the correctness of the output of each processor with high probability. The communication complexity of algorithm AVERAGE depends linearly on the average discrepancy δ_{av}, while its time complexity is still linear in m. Both complexities apply to the worst-case scenario.

We begin with a high level description of algorithm AVERAGE. The algorithm works in phases and invokes algorithm MAXIMUM of section 3.1 at each phase. At every phase of the execution, each processor can be in one of two states, denoted $\mathcal{K}$ and $\mathcal{R}$. Initially, only the broadcaster is in state $\mathcal{K}$, while the other processors are in state $\mathcal{R}$. Intuitively, a processor p_i switches from state $\mathcal{R}$ to state $\mathcal{K}$ when it concludes that its current guess for w is equal to the "real" broadcast word w.

The phases are designed to handle processors with increasingly large discrepancies. More specifically, let us classify the processors into classes $C_1, \ldots, C_\mu$, $\mu = \lceil \log(\frac{m}{\delta_{av} \log m}) \rceil$, where the class C_l contains all processors p_i whose discrepancy δ_i falls in the range $2^{l-1}\delta_{av} < \delta_i \leq \min\{m, 2^l \delta_{av}\}$ for $2 \leq l \leq \mu - 1$, $\delta_i \leq 2\delta_{av}$ for $l = 1$, and the rest in C_μ (i.e., $\delta_i \geq \frac{m}{\log m}$). Then each phase $l \geq 0$ is responsible for informing the processors in class C_l. This is done by letting the processors in state $\mathcal{K}$ broadcast to the other processors.

Note that the $\mathcal{K}$ and $\mathcal{R}$ states reflect, in a sense, only the processors' "state of mind," and not necessarily the true situation. It might happen that a processor switches prematurely to state $\mathcal{K}$, erroneously believing it holds the true value of the input w. Such an error might subsequently propagate to neighboring processors as well. Our analysis will show that this happens only with low probability.

By a simple counting argument, the fraction of processors whose discrepancy satisfies $\delta_i \geq k\delta_{av}$ is bounded from above by $\frac{1}{k}$ for every $k \geq 1$. The first phase attempts to correct the inputs of processors from C_1, while in general, the lth phase attempts to correct the processors in C_l. By the previous argument, at least half of the processors are in C_1, and furthermore, $\sum_{j=l}^{\mu} |C_j| \leq \frac{n}{2^l}$. Assuming that all the processors that shifted from $\mathcal{R}$ to $\mathcal{K}$ had the correct value, then after the lth phase, at most $\frac{n}{2^l}$ processors are in state $\mathcal{R}$.

Let us now describe the structure of a phase l in more detail. At the beginning of phase l, the current states of the processors induce a conceptual partition of the line network into consecutive intervals $I_1, \ldots, I_t$, with each interval $I = (p_i, p_{i+1}, \ldots)$ containing one or more processors, such that the first processor p_i is in state $\mathcal{K}$ and the rest of the processors (if any) are in state $\mathcal{R}$.

The algorithm maintains the property that each processor knows its state, as well as the state of its two neighbors, hence each processor knows its relative role in its interval, as either a "head" of the interval, an intermediate processor, or a "tail" (i.e., the last processor of the interval).

Suppose that processor p_i is in state $\mathcal{K}$ at the beginning of phase $l < \mu$ and is the "head" of some interval I. If the processor p_{i+1} is also in state $\mathcal{K}$, then the interval I contains only p_i, and thus p_i has finished its part in the algorithm. Otherwise, interval I contains at least one processor in state $\mathcal{R}$. In this case, processor p_i is assigned the role of the *broadcaster* with respect to its interval in phase l. More specifically, it needs to reveal its value to all processors of class C_l in its interval I. Hopefully, this results in the further partition of interval I into subintervals for the next phase.

Processor p_i performs this task by using algorithm MAXIMUM of section 3.1, with parameter $d_l = 2^l \delta_{av}$. To be more specific, if p_i's interval I contains other processors (i.e., processor p_{i+1} is in state $\mathcal{R}$) then p_i computes $C^*_{m,d_l}(o_i)$ and sends it to p_{i+1}. (In case $l = \mu$, processor p_i sends o_i.) As we shall see, with high probability, $o_i = w$

for any processor i that is in state $\mathcal{K}$. Therefore, later in this informal description we substitute $C^*_{m,d_l}(w)$ for $C^*_{m,d_l}(o_i)$. Consider that any intermediate processor p_j (in state $\mathcal{R}$) in interval I that receives a message simply forwards the message (using pipelining). The tail processor of the interval (i.e., the one whose successor is in state $\mathcal{K}$) does nothing.

It remains to explain when a processor decides to change its state from $\mathcal{R}$ to $\mathcal{K}$. This task requires an *initialization* phase, in which the broadcaster chooses a random universal hash function $h \in \mathcal{F}_{\epsilon/n\mu}$ and sends both the description of h and the hashed value of the broadcast message, i.e., the pair

$$\mathcal{H}(w) = \langle h, h(w) \rangle,$$

to all processors. Since the description of h requires $O(\log \frac{n\mu}{\epsilon})$ bits, the size of the message is $O(\log \frac{n\mu}{\epsilon}) = O(\log \log m + \log \frac{n}{\epsilon})$. The pair $\mathcal{H}(w)$ will later serve each processor to test whether its new computed value of w is correct.

Specifically, as said above, in phase $l < \mu$ each processor p_j in state $\mathcal{R}$ receives $\rho_l = C^*_{m,d_l}(w)$. It concatenates it to w_j and computes

$$g_j(l) = C^{-1}_{m,d_l}(w_j C^*_{m,d_l}(w)),$$

which is its "guess" for w. It then tests whether $h(g_j(l)) = h(w)$.

In case of equality, the processor deduces that its current guess is correct. It then changes its state from $\mathcal{R}$ to $\mathcal{K}$ and sets its output to be $o_j = g_j(l)$. At the last phase, $l = \mu$, when a processor p_j receives value o_i from some processor p_i, then p_j sets $o_j = o_i$. The algorithm ends after phase μ.

LEMMA 3.5. *The probability that some processor produces an incorrect output is bounded from above by* ϵ.

Proof. Let $\mathcal{E}_j(l)$ denote the event that processor p_j outputs an incorrect value at phase l, given that all the outputs at earlier phases were correct. This event implies that $g_j(l) \neq w$, but $h(g_j(l)) = h(w)$. However, the hash function h was chosen so as to guarantee that this probability is at most $\frac{\epsilon}{n\mu}$. Summing over all possible bad events $\mathcal{E}_j(l)$, it follows that the probability that some processor ends with an incorrect output is bounded by ϵ.

Another possible failure is that a processor stays in state $\mathcal{R}$. However, recall that in this case some other processor has to output an incorrect value before. ∎

Now, we analyze the communication and time complexities of the protocol. We first show that if no node mistakenly outputs an incorrect value, then both time and communication complexities are small.

LEMMA 3.6. *Assuming that no processor outputs an incorrect value, the time complexity of algorithm* AVERAGE *is* $O(n + m + \log \frac{1}{\epsilon})$.

Proof. The initialization phase involves sending a message $\mathcal{H}(w)$ containing the pair $\langle h, h(w) \rangle$, which is of size $O(\log \log m + \log \frac{n}{\epsilon})$ bits, to all n processors. This phase therefore requires time $O(n + \log \log m + \log \frac{n}{\epsilon})$. It remains to analyze the time required for the main phases.

Assuming that at the start of phase l all the processors in state $\mathcal{K}$ have the correct value, the number of processors in state $\mathcal{R}$ at the end of the phase is at most $n/2^l$. This follows from the fact that at phase l the algorithm corrects the input values of all processors whose discrepancy is at most $d_l = 2^l \delta_{av}$. Since the average discrepancy is δ_{av}, the number of processors with a larger discrepancy is at most $n/2^l$.

The time required for completing a phase l is clearly bounded from above by the number of processors in state $\mathcal{R}$ plus the size of the message sent in this phase,

i.e., $n/2^l + \min\{m, d_l \log m\}$. The first term is clearly bounded by n, and the second obtains its maximum at the last phase and is therefore bounded by $O(m)$.

Hence the time complexity is $O(n + m + \log \frac{1}{\epsilon})$. □

LEMMA 3.7. *Assuming that no processor outputs an incorrect value, the communication complexity of algorithm* AVERAGE *is* $O(\Delta \log m + n \log \frac{n}{\epsilon})$.

Proof. Again, the initialization phase requires sending a message of size $O(\log \log m + \log \frac{n}{\epsilon})$, which contributes $O(n(\log \log m + \log \frac{n}{\epsilon})) \leq O(\Delta \log m + n \log \frac{n}{\epsilon})$ to the communication complexity.

Let us now concentrate on the main phases. Consider a processor p_i with discrepancy δ_i. We count the number of bits that p_i *receives* during the entire execution. After phase $\mu_i = \lceil \log(\delta_i/\delta_{av}) \rceil$, assuming that all the processors in state $\mathcal{K}$ have the correct value, p_i should already be in state $\mathcal{K}$, and from then on it never receives messages. At each phase l prior to phase μ_i, processor p_i gets a message $C^*_{m,d_l}(w)$ of size $O(d_l \log m)$ bits. Therefore, the number of bits received by p_i throughout the execution is bounded by

$$\sum_{l=1}^{\mu_i} O(d_l \log m) = O(2^{\mu_i} \delta_{av} \log m) = O(\delta_i \log m).$$

Summing over all processors, the contribution of the main phases is bounded by

$$\sum_{i=1}^{n} O(\delta_i \log m) = O(\Delta \log m).$$

Consequently, the communication complexity of the entire algorithm is $O(\Delta \log m + n \log \frac{n}{\epsilon})$. □

Note that ϵ can always be chosen so as to make the failure probability polynomially small in m, without degrading the time or communication complexities of the algorithm. Consequently we have the following theorem.

THEOREM 3.8. *Given an average discrepancy* δ_{av} *and* $0 < \epsilon < 1$, *algorithm* AVERAGE *solves the broadcast with partial knowledge problem correctly with probability* $1 - \epsilon$. *In the case when the solution is correct, the time complexity is* $O(n + m + \log \frac{1}{\epsilon})$ *and the communication complexity is* $O(\delta_{av} \cdot n \cdot \log m + n \log \frac{n}{\epsilon})$ *bits.*

In case algorithm AVERAGE fails and the output is incorrect, we can guarantee only trivial bounds on the time and communication complexities of the algorithm. These bounds are derived from bounding the number of phases by $\log m$, and the number of bits in a message by m. This gives worst-case bounds of $O((n + m) \log m)$ time and $O(nm)$ communication. However, ϵ can be selected so as to equate the expected complexity (over all executions) with the high probability complexity (i.e., over the executions that have a correct output). Consequently we have the following corollary.

COROLLARY 3.9. *Algorithm* AVERAGE *has expected time complexity* $O(n + m)$ *and expected communication complexity of* $O(\Delta \log m + n \log nm)$ *bits.*

3.3. Unknown discrepancy. In the case when δ_{av} is not known in advance, we can solve the problem by initiating algorithm AVERAGE with a guessed average discrepancy $\hat{\delta}_{av} = 1$. Call this algorithm UNKNOWN. In such a case, in the $\log \delta_{av}$ first phases of algorithm AVERAGE, it may happen that no processor changes to $\mathcal{K}$. The communication complexity essentially remains the same, since the additional $O(\delta_{av} \cdot n) = O(\Delta)$ bits are absorbed in the previous bound. However, the time complexity does increase in this case by an additive factor of $O(n \log \delta_{av})$.

THEOREM 3.10. *Given* $0 < \epsilon < 1$, *algorithm* UNKNOWN *solves the broadcast with partial knowledge problem correctly with probability* $1-\epsilon$. *In the case when the solution is correct, the time complexity is* $O(n \log \delta_{av} + m + \log \frac{1}{\epsilon})$ *and the communication complexity is* $O(\Delta \log m + n \log \frac{n}{\epsilon})$ *bits.*

In a similar way to the previous section, we can bound the expected complexities by choosing ϵ appropriately.

COROLLARY 3.11. *Algorithm* UNKNOWN *has expected time complexity* $O(n \log \delta_{av} + m)$ *and expected communication complexity of* $O(\Delta \log m + n \log nm)$ *bits.*

3.4. Optimal time complexity. Intuitively, the main reason that the time in the previous subsection is not optimal is the fact that there can be "time spaces" between the phases. In this section we describe a simulation of algorithm UNKNOWN in which the information transmitted is fully pipelined. For the sake of clarity we describe the whole algorithm below.

Algorithm BPART**.** We assume that $\Delta \log m < mn$. If this assumption does not hold, then the algorithm might fail, at which time full broadcast can be engaged, since the maximum complexity must be paid in any case.

The algorithm requires an initialization phase similar to the one of algorithm AVERAGE, in which a random universal hash function $h \in \mathcal{F}_{\epsilon/n\mu}$ is chosen and the pair $\mathcal{H}(w)$ is sent to all processors.

The main part of the algorithm proceed as follows. Set $\mu = \log m - \log \log m$. The source transmits the syndromes of the encodings $w(i) = \mathcal{C}_{m,2^i}(w)$ of its vector w in all codes $\mathcal{C}^*_{m,2^i}$ of all levels $0 \le i \le \mu$, one after another. Hence the sequence transmitted by the source is of the form

$$\xi(w) = \langle w(0), w(1), \ldots, w(\mu), w \rangle.$$

Essentially, this information is to be forwarded along the line to all processors. Observe that a naive implementation of this would require communication complexity nm and time $m + n$. The communication complexity is minimized by stopping the flood of bits into a node p_j once this node is able to correctly decode the entire source's vector ξ (with high probability), i.e., to produce an output o_j such that $h(w) = h(o_j)$. This is done in the same way as in algorithm UNKNOWN, as described below.

During the main part of the algorithm, each processor p_j on the line does the following. It initially enters state $\mathcal{R}$, intuitively signifying the fact that its data may be outdated. It then receives the sequence $\xi(o_{j-1})$ from its predecessor p_{j-1}, constructs its own output o_j and sequence $\xi(o_j)$, and sends this sequence to its successor. The incoming sequence is received and processed entry by entry, as long as p_j is in state $\mathcal{R}$.

Each arriving entry $o_{j-1}(i) = \mathcal{C}^*_{m,2^i}(o_{j-1})$, $0 \le i \le \mu$, is processed as follows. First, the entry is stored as $o_j(i)$ and forwarded to p_{j+1}. In addition, p_j concatenates $o_{j-1}(i)$ to w_j and computes its "guess" for w,

$$g_j(i) = C^{-1}_{m,2^i}(w_j \cdot C^*_{m,2^i}(o_{j-1})).$$

It then tests whether $h(g_j(i)) = h(w)$.

In case of equality, the processor does the following. First, it changes its state from $\mathcal{R}$ to $\mathcal{K}$ and informs its predecessor p_{j-1} to stop sending the rest of the sequence $\xi(o_{j-1})$. Next, it sets its output to be $o_j = g_j(i)$ and computes the rest of the sequence $\xi(o_j)$ locally (in order to be able to continue sending it to its successor on the line).

Note that p_j may fail in its tests in all stages $i \leq \mu$. In this case, it will receive the entire sequence $\xi(o_{j-1})$ from its predecessor, including o_{j-1} in its last entry, and adopt this value for o_j.

Let us now bound the probability of failure.

LEMMA 3.12. *The probability that some processor produces an incorrect output is bounded from above by ϵ.*

Proof. The proof proceeds in the same way as that of Lemma 3.5, except that we define $\mathcal{E}_j(i)$ to be the event that processor p_j outputs an incorrect value after getting $o_{j-1}(i)$, while the output of every processor p_l to the left of p_j (i.e., such that $l < j$) that made its decision after receiving (all or some of) the syndromes $o_{l-1}(k)$ for $0 \leq k \leq i$ is correct. □

Finally, we analyze the communication and time complexities of the protocol.

THEOREM 3.13. *Assuming that no processor outputs an incorrect value, the communication complexity of algorithm* BPART *is $O(\Delta \log m + n \log \frac{n}{\epsilon})$.*

Proof. The proof is similar to that of Lemma 3.7. In particular, the initialization message $\mathcal{H}(w)$ requires sending $O(\log \frac{n\mu}{\epsilon}) = O(\log \log m + \log \frac{n}{\epsilon})$ bits. As for the ξ messages, nodes with more than $\frac{m}{\log m}$ errors receive at most $O(m)$ bits, and there can be at most $\delta_{av} \log m$ such nodes. Otherwise, a node p_j that has less than 2^i errors will decode the original message correctly (with high probability) after receiving $\mathcal{C}_{m,2^i}(o_{j-1})$, i.e., after receiving $O(2^i \cdot \log m)$ bits. □

THEOREM 3.14. *Assuming that no processor outputs an incorrect value, the time complexity of algorithm* BPART *is $O(n + \log \frac{n}{\epsilon} + \min\{m, \Delta \log m\})$.*

Proof. The upper bound follows from the fact that all the data from the source (including both the $\mathcal{H}(w)$ and the ξ messages) is sent along the line in full pipelining, and the combined length of the vector broadcast is $O(\log \frac{n}{\epsilon} + \min\{m, \Delta \log m\})$. □

As before, we bound the expected complexities by choosing ϵ appropriately.

COROLLARY 3.15. *Algorithm* BPART *has expected time complexity $O(n+m)$ and expected communication complexity $O(\Delta \log m + n \log nm)$.*

4. Lower bounds. In this section we establish some simple lower bounds that show that our construction is optimal. The first bound concerns the communication complexity of broadcast algorithms assuming maximum discrepancy $d = \delta_{max}$. Assume that all processors but the broadcaster have as input the all-zero vector, while the broadcast message is a vector containing exactly d ones, that is chosen arbitrarily from among all $\binom{m}{d}$ possible vectors, with all choices being equally likely. This implies that the entropy of the source (the broadcaster) is $\log \binom{m}{d}$. The entropy is clearly a lower bound on the number of bits each processor has to receive. Therefore, we have the following bound.

THEOREM 4.1. *The communication complexity of any broadcast algorithm is at least*

$$n \log \binom{m}{\delta_{max}} = \Omega\left(n\delta_{max} \log\left(\frac{m}{\delta_{max}}\right)\right).$$

The bound on the time required for broadcast is derived by considering the communication lower bound above and the last processor, and noticing that it cannot receive any information before time n. Therefore, we have the following theorem.

THEOREM 4.2. *The time complexity of any broadcast algorithm is at least*

$$n + \log \binom{m}{\delta_{max}} = \Omega\left(n + \delta_{max} \log\left(\frac{m}{\delta_{max}}\right)\right).$$

The next theorem establishes the limitations of the derandomization of the algorithm when the average discrepancy is δ_{av}, and it is not known in advance. Consider a simpler task, in which every processor has to decide whether its input equals the broadcast message or not. This is a well-studied problem in communication complexity theory, where lower bounds for the number of bit exchanges required of solving the equality problem are known. For the case of $n = 2$, Yao showed a lower bound of $\Omega(m)$ for deterministic algorithms [Yao79], and Tiwari extended it to a line of processors and showed an $\Omega(nm)$ lower bound [Tiw84]. This bound holds in particular when all the inputs are equal, in which case $\delta_{av} = 0$. This implies the following lower bound.

THEOREM 4.3. *For any deterministic or randomized Las-Vegas-type algorithm that has to work for an arbitrary δ_{av}, there is an input whose discrepancy is zero (i.e. $\delta_{av} = 0$), and the algorithm requires $\Omega(nm)$ communication complexity on this input.*

5. Conclusion. We have shown that one can take advantage of prior knowledge of the recipient in order to save in communication, while still keeping the time optimal in the worst case. It may be interesting to find out whether more efficient solutions can be derived for special cases. For example, it often happens that faults are related. Consequently, it may be likely that the set of bits in which w at some recipient disagrees with w_0 at the source, forms clusters, rather than being distributed arbitrarily in w. It may be interesting to devise an algorithm that performs better in such a case. A technique for finding regions of disagreements appears in [Met91].

There may be other interesting special cases. One such case is when later (and more updated) versions of a document are longer (or shorter) than previous versions. Our algorithm can solve this case by "padding" the shorter vector, but this seems to lead to an inefficient solution.

REFERENCES

[AAG87] Y. AFEK, B. AWERBUCH, AND E. GAFNI, *Applying static network protocols to dynamic networks*, in 28th Annual IEEE Symposium on Foundations of Computer Science, 1987.

[ACG+90] B. AWERBUCH, I. CIDON, I. GOPAL, M. KAPLAN, AND S. KUTTEN, *Distributed control for paris*, in Proc. 9th ACM Symposium on Principles of Distributed Computing, 1990.

[ACK90] B. AWERBUCH, I. CIDON, AND S. KUTTEN, *Optimal maintenance of replicated information*, in Proc. 31st IEEE Symposium on Foundations of Computer Science, 1990.

[ACK+91] B. AWERBUCH, I. CIDON, S. KUTTEN, Y. MANSOUR, AND D. PELEG, *Broadcast with partial knowledge*, in Proc. 10th ACM Symposium on Principles of Distributed Computing, 1991.

[APV91] B. AWERBUCH, B. PATT-SHAMIR, AND G. VARGHESE, *Self-stabilization by local checking and correction*, in Proc. 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 268–277.

[AS91] B. AWERBUCH AND L. J. SCHULMAN, *The maintenance of common data in a distributed system*, in Proc. 32nd IEEE Symposium on Foundations of Computer Science, 1991.

[AKY90] Y. AFEK, S. KUTTEN, AND M. YUNG, *Memory-efficient self-stabilization on general networks*, in Proc. 4th Workshop on Distributed Algorithms, Italy, September 1990. Lecture Notes in Comput. Sci. 486, Springer-Verlag, 1990. A later version appeared as *The local detection paradigm and its applications to self stabilization*, Theoret. Comput. Sci., 186 (1977), pp. 199–229.

[BGJ+85] A. E. BARATZ, J. P. GRAY, P. E. GREEN, JR., J. M. JAFFE, AND D. P. POZEFSKI, *Sna networks of small systems*, IEEE J. on Selected Areas in Communications, SAC-3(3) (1985), pp. 416–426.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson, *Completeness theorem for non-cryptographic fault tolerant distributed computing*, in Proc. 20th ACM Symposium on Theory of Computing, 1988.

[Dij74] E. W. Dijkstra, *Self stabilizing systems in spite of distributed control*, Comm. ACM, 17 (1974), pp. 643–644.

[Eve79] S. Even, *Graph Algorithms*, Computer Science Press, Rockville, MD, 1979.

[KP90] S. Katz and K. Perry, *Self-stabilizing extensions for message-passing systems*, in Proc. 10th ACM Symposium on Principles of Distributed Computing, Quebec, Canada, 1990.

[Met84] J. J. Metzner, *An improved broadcast retransmission protocol*, IEEE Trans. Commun., COM-32(6) (1984), pp. 679–683.

[Met91] J. J. Metzner, *Efficient replicated remote file comparison*, IEEE Trans. Comput., 40 (1991), pp. 651–660.

[MRR80] I. McQuillan, I. Richer, and E. C. Rosen, *The new routing algorithm for the arpanet*, IEEE Trans. Commun., COM-28 (1980), pp. 711–719.

[Rab89] M. Rabin, *Efficient dispersal of information for security, load balancing, and fault tolerance*, J. Assoc. Comput. Mach., 36 (1989), pp. 335–348.

[SG89] J. M. Spinelli and R. G. Gallager, Event driven topology broadcast without sequence numbers, IEEE Trans. Commun., 37 (1989), pp. 468–474.

[SW73] D. Slepian and J. K. Wolf, *Noiseless coding of correlated information sources*, IEEE Trans. Inform. Theory, IT-19 (1973), pp. 471–480.

[Tiw84] P. Tiwari, *Lower bounds on communication complexity in distributed computer networks*, in 25th Annual IEEE Symposium on Foundations of Computer Science, Singer Island, FL, 1984, pp. 109–117.

[WC79] M. N. Wegman and J. L. Carter, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[Yao79] A. Yao, *Some complexity questions related to distributed computing*, in Proc. 11th Annual ACM Symposium on Theory of Computing, Atlanta, GA, ACM SIGACT, 1979, pp. 209–213.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 525–540

PRIMAL-DUAL RNC APPROXIMATION ALGORITHMS FOR SET COVER AND COVERING INTEGER PROGRAMS*

SRIDHAR RAJAGOPALAN† AND VIJAY V. VAZIRANI‡

Abstract. We build on the classical greedy sequential set cover algorithm, in the spirit of the primal-dual schema, to obtain simple parallel approximation algorithms for the set cover problem and its generalizations. Our algorithms use randomization, and our randomized voting lemmas may be of independent interest. Fast parallel approximation algorithms were known before for set cover, though not for the generalizations considered in this paper.

Key words. algorithms, set cover, primal-dual, parallel, approximation, voting lemmas

PII. S0097539793260763

1. Introduction.

Given a universe $\mathcal{U}$, containing n elements, and a collection, $\mathcal{S} \doteq \{S_i : S_i \subseteq \mathcal{U}\}$, of subsets of the universe, the *set cover* problem asks for the smallest subcollection $\mathcal{C} \subseteq \mathcal{S}$ that covers all the n elements in $\mathcal{U}$ (i.e., $\bigcup_{S\in\mathcal{C}} S = \mathcal{U}$). In a more general setting, one can associate a cost, c_S, with each set $S \in \mathcal{S}$ and ask for the minimum cost subcollection which covers all of the elements.[1] We will use m to denote $|\mathcal{S}|$.

Set multicover and *multiset multicover* are successive natural generalizations of the set cover problem. In both problems, each element e has an integer coverage requirement r_e, which specifies how many times e has to be covered. In the case of multiset multicover, element e occurs in a set S with arbitrary multiplicity, denoted $m(S, e)$. Setting $r_e = 1$ and choosing $m(S, e)$ from $\{0, 1\}$ to denote whether S contains e gives back the set cover problem.

The most general problems we address here are *covering integer programs*. These are integer programs that have the following form:

$$\text{MIN } \mathbf{c}\cdot\mathbf{x}, \text{ s.t. } M\mathbf{x} \geq \mathbf{r}, \mathbf{x} \in \bar{\wedge}^+;$$

the vectors $\mathbf{c}$ and $\mathbf{r}$ and the matrix M are all nonnegative rational numbers.

Because of its generality, wide applicability, and clean combinatorial structure, the set cover problem occupies a central place in the theory of algorithms and approximations. Set cover was one of the problems shown to be NP-complete in Karp's seminal paper [Ka72]. Soon after this, the natural greedy algorithm, which repeatedly adds the set that contains the largest number of uncovered elements to the cover, was shown to be an H_n factor approximation algorithm for this problem ($H_n = 1 + 1/2 + \ldots + 1/n$) by Johnson [Jo74] and Lovasz [Lo75]. This result was extended to the minimum cost case by Chvatal [Ch79]. Lovasz establishes a slightly stronger statement, namely, that the ratio of the greedy solution to the optimum fractional solution is at most

*Received by the editors November 26, 1993; accepted for publication (in revised form) October 21, 1996; published electronically July 28, 1998.
http://www.siam.org/journals/sicomp/28-2/26076.html

†DIMACS, Princeton University, Princeton, NJ 08544. Current address: IBM Almaden Research Center, San Jose, CA 95120 (sridhar@almaden.ibm.com). This research was done while the author was a graduate student at the University of California, Berkeley, supported by NSF PYI Award CCR 88-96202 and NSF grant IRI 91-20074. Part of this work was done while the author was visiting IIT, Delhi.

‡College of Computing, Georgia Institute of Technology, Atlanta, GA (vazirani@cc.gatech.edu). This research was supported by NSF grant CCR-9627308.

[1] Here it is to be understood that the cost of a subcollection $\mathcal{C}$ is $\sum_{S\in\mathcal{C}} c_S$.

H_n. Consequently, the *integrality gap*, the ratio of the optimum integral solution to the optimum fractional one, is at most H_n. An approximation ratio of $O(\log n)$ for this problem has been shown to be essentially tight by Lund and Yannakakis [LY93].[2] More recently, Feige [Fe96] has shown that approximation ratios better than $\ln n$ are unlikely.

The first parallel algorithm for approximating set cover is due to Berger, Rompel, and Shor [BRS89], who found an RNC^5 algorithm with an approximation guarantee of $O(\log n)$. Further, this algorithm can be derandomized to obtain an NC^7 algorithm with the same approximation guarantee. Luby and Nisan [LN93], building on the work of Plotkin, Shmoys, and Tardos [PST91] have obtained a $(1+\epsilon)$ factor (for any constant $\epsilon > 0$), NC^3 approximation algorithm for covering and packing linear programs. Since the integrality gap for set cover is H_n, the Luby–Nisan algorithm approximates the cost of the optimal set cover to within a $(1+\epsilon)H_n$ factor. Furthermore (as noted by Luby and Nisan), in the case of set cover, by using a randomized rounding technique (see [Ra88]), fractional solutions can be rounded to integral solutions at most $O(\log n)$ times their value.

This paper describes a new RNC^3, $O(\log n)$ approximation algorithm for the set cover problem. This algorithm extends naturally to RNC^4 algorithms for the various extensions of set cover each achieving an approximation guarantee of $O(\log n)$. In addition, the approximation guarantee that we obtain in the case of covering integer programs is better than the best known sequential guarantee, due to Dobson [Do82].

2. A closer look at set cover. We begin by taking a closer look at the greedy algorithm for set cover. In the minimum cost case, the greedy algorithm chooses the set that covers new elements at the lowest average cost. More precisely, let $U(S)$ denote the set of yet uncovered elements in S. The greedy algorithm repeatedly adds the set $\operatorname{argmin}_S \left\{ \frac{c_S}{|U(S)|} \right\}$ to the set cover.

Choose $\text{cost}(e)$ to be the cost of covering e by the greedy algorithm. Thus, if S is the first set to cover e and $U(S) = k$ before S was added to the cover, then $\text{cost}(e) \doteq c_S/k$. Let e_i be the ith last element to be covered. Let OPT denote the optimum set cover as well as its cost.

We now observe that $\text{cost}(e_i) \leq \text{OPT}/i$ because there is a collection of sets, namely OPT, which covers all i elements $\{e_1, \ldots, e_i\}$. Thus, there is a set S such that $c_S \leq \text{OPT}/i|U(S)|$. Since e_i is the element of smallest cost among $\{e_1 \cdots e_i\}$, $\text{cost}(e_i) \leq \text{OPT}/i$. Therefore, the cost of the cover obtained by the greedy algorithm is at most $\sum_i \text{cost}(e_i) \leq \text{OPT} \sum_i \frac{1}{i} = \text{OPT}H_n$, which provides the approximation guarantee.

The simple proof given above can be viewed in the more general and powerful framework of linear programming and duality theory. One can state the set cover problem as an integer program:

$$\textbf{IP}: \quad \begin{array}{ll} \text{MIN} & \sum_S c_S x_S \\ \text{s.t.} & \sum_{S \ni e} x_S \geq 1, \\ & x_S \in \{0,1\}. \end{array}$$

By relaxing the integrality condition on $\mathbf{x}$, we obtain a linear program which has the

[2] More precisely, Lund and Yannakakis establish that there is a constant c such that unless $\tilde{\text{P}} = \tilde{\text{NP}}$, set cover cannot be approximated to a ratio smaller than $c \log n$.

following form and dual:

$$\begin{array}{lll} \textbf{LP}: & & \textbf{DP}: \\ \text{MIN} & \sum_S c_S x_S & \text{MAX} \quad \sum_e y_e \\ \text{s.t.} & \sum_{S \ni e} x_S \geq 1, & \text{s.t.} \quad \sum_{e \in S} y_e \leq c_S, \\ & x_S \geq 0; & \qquad y_e \geq 0. \end{array}$$

The primal linear program is a covering problem and the dual is a packing problem. We will now reanalyze the greedy algorithm in this context. Define value(e) of an uncovered element as

$$\text{value}(e) \doteq \min_{S \ni e} \frac{c_S}{|U(S)|}.$$

The "value" of an element is a nondecreasing function of time in that it gets bigger as more and more elements get covered and therefore each $U(S)$ gets smaller. The greedy algorithm guarantees that cost(e) = value(e) at the moment e is covered.

Now consider any set $S \in \mathcal{S}$. Let $e_i \in S$ be the ith last element of S to be covered by the greedy algorithm. Then, clearly, $\text{cost}(e_i) = \text{value}(e_i) \leq c_S/i$ at the moment of coverage. Thus, we establish the following inequality for each set S:

$$\sum_{e \in S} \text{cost}(e) \leq \left(1 + \frac{1}{2} + \cdots + \frac{1}{|S|}\right) c_S.$$

Alternately, if k is the size of the largest set $S \in \mathcal{S}$, then the assignment $y_e = \frac{\text{cost}(e)}{H_k}$ is dual feasible. The dual value for this assignment to $\mathbf{y}$ is $\mathbf{DP} = \sum_e y_e = \frac{1}{H_k} \sum_e \text{cost}(e) = \frac{1}{H_k}$greedy cost. Due to the duality theorem of linear programming, $\mathbf{DP}$ is a lower bound on the value of OPT. Thus, the greedy algorithm approximates the value of set cover to within a factor of H_k.

3. The key ideas behind our parallel algorithm. The greedy algorithm chooses a set to add to the set cover for which $\sum_{e \in S} \text{value}(e) = c_S$. One of the key ideas in this paper is to find a suitable relaxation of this set selection criterion which guarantees that rapid progress is made but does not degrade the approximation guarantee significantly. This is by no means a new notion. Indeed, it has been used in the context of set cover earlier [BRS89]. However, the way in which the relaxation is made and the resulting parallel algorithms are different from the choices made in [BRS89].

In our algorithm we identify cost-effective sets by choosing those that satisfy the inequality

$$\sum_{e \in U(S)} \text{value}(e) \geq \frac{c_S}{2}.$$

This criterion can be distinguished from the greedy criterion in that elements with different values contribute to the desirability of any set. This weighted mixing of diverse elements and the consequent better use of the dual variables, value(e), appears to lend power to our criterion.

Our relaxed criterion guarantees rapid progress. To see this, consider $\alpha \doteq \min_e \text{value}(e)$. Then, any set for which $\frac{c_S}{|U(S)|} \in [\alpha, 2\alpha]$ qualifies to be picked. Thus, after each iteration, the value of α doubles. This and a preprocessing step will enable us

to show rapid progress for our algorithm. However, an algorithm based solely on this relaxed criterion will not approximate set cover well, as exhibited by Example 3.0.1.

EXAMPLE 3.0.1. Let $\mathcal{U} = \{1, 2, \ldots, n\}$. Let $\mathcal{S}$ be the $\binom{n}{2}$ sets of size 2 derived from $\mathcal{U}$. The cost of each set is 1. The optimal cover consists of $\frac{n}{2}$ sets. However, the relaxed criterion chooses all the sets. □

We now address the approximation guarantee. We will be able to assign costs to each element, denoted cost(e), such that the cost of the set cover is at most $\sum_e \text{cost}(e)$. Further, we will establish that at the moment of coverage, cost(e) $\leq \mu$value(e) for each element e and a suitably chosen constant μ. If, for any algorithm, it is possible to choose element costs such that these conditions are satisfied, then we will say that the algorithm has the *parsimonious accounting* property with parameter μ. It is evident from the analysis of the greedy algorithm detailed earlier that the parsimonious accounting property with parameter μ suffices to establish an approximation guarantee of μH_k where k is the cardinality of the largest set in $\mathcal{S}$.

These observations motivate a straightforward method of relaxing the set selection criterion which, however, fails to achieve our stated goal, namely, fast parallel execution. The relaxation and an instance exhibiting its shortcomings are detailed by Example 3.0.2 below.

EXAMPLE 3.0.2. Number the sets arbitrarily. In each iteration, each uncovered element votes for the lowest numbered set that covers it at the least average cost. Any set with more than $\frac{U(S)}{2}$ votes adds itself to the set cover. It is easily verified that this algorithm has the parsimonious accounting property with parameter $\mu = 2$. However, the algorithm can be forced to execute $\Theta(n)$ iterations on the following input: let $\mathcal{U} = \{u_i, v_j : 1 \leq i, j \leq n\}$. Choose $\mathcal{S} = \{U_i, V_j : 1 \leq i, j \leq n-1\}$, where $U_i = \{u_i, u_{i+1}, v_{i+1}\}$ and $V_j = \{v_j, u_{j+1}, v_{j+1}\}$. Finally, choose the set costs to satisfy $c_{U_{i-1}} > c_{U_i} = c_{V_i} > c_{U_{i+1}}$ for ech i. □

The solution we propose finds a compromise between the two strategies detailed above to achieve both objectives, namely, rapid progress as well as good approximation. The critical extra ingredient used in making this possible is the introduction of randomization into the process.

The primal-dual scheme provides a general framework in which we can search for good and fast approximation algorithms for otherwise intractable problems. In the typical case, the hard problem is formulated as an integer program which is then relaxed to obtain a linear program and its dual. In this context, the algorithm starts with a primal, integral infeasible solution and a feasible, suboptimal dual solution. The algorithm proceeds by iteratively improving the feasibility of the primal and the optimality of the dual while maintaining primal integrality until the primal solution becomes feasible. On termination, the obtained primal integral feasible solution is compared directly with the feasible dual solution to give the approximation guarantee. The framework leaves sufficient room to use the combinatorial structure of the problem at hand: in designing the algorithm for the iterative improvement steps and in carrying out the proof of the approximation guarantee.

The greedy algorithm for set cover can be viewed as an instance in the above paradigm. At any intermediate stage of the algorithm let $y_e = \frac{\text{cost}(e)}{H_n}$ if e has been covered and $\frac{\text{value}(e)}{H_n}$ otherwise. Then, by the arguments presented in section 2, $\mathbf{y}$ is feasible for **DP**. The currently picked sets constitute the primal solution.

4. The parallel set cover algorithm. Our proposed algorithm for set cover is described in Figure 1. The preprocessing step is done once at the inception of

PARALLEL SETCOV

Preprocess.

Iteration:

For each uncovered element e, compute value(e).

For each set S: include S in $\mathcal{L}$ if

(•) $\sum_{e \in U(S)} \text{value}(e) \geq \frac{c_s}{2}$.

Phase:

(a) Permute $\mathcal{L}$ at random.

(b) Each uncovered element e votes for first set S (in the random order) such that $e \in S$.

(c) If $\sum_{e \text{ votes } S} \text{value}(e) \geq \frac{c_S}{16}$, S is added to the set cover.

(d) If any set fails to satisfy (•), it is deleted from $\mathcal{L}$.

Repeat until $\mathcal{L}$ is empty.

Iterate until all elements are covered.

FIG. 1. *The parallel set cover algorithm.*

the computation and is a technical step which we will explicitly establish in the next section. The purpose of this step is to reduce the range of values of c_S to one wherein the largest and smallest values are at most a polynomial factor from each other.

Notice that value(e) is computed only at the beginning of an iteration and is not updated at the inception of each phase.

4.1. Analysis of PARALLEL SETCOV. We now present an analysis of the proposed algorithm. We will first consider the approximation guarantee and then the running time. The content of the preprocessing step will be defined with the analysis of the running time.

4.1.1. Approximation guarantee. The algorithm PARALLEL SETCOV satisfies the parsimonious accounting property with $\mu = 16$. If we choose $\text{cost}(e) = 16\text{value}(e)$ if e votes for S and S is added to the set cover in the same phase, then it is easily verified that $\sum_e \text{cost}(e) \geq$ cost of cover.

4.1.2. Running time. We will now establish that PARALLEL SETCOV is in RNC^3. In order to do this, we will establish the following three assertions.

1. The algorithm executes $O(\log n)$ iterations.
2. With high probability, $(1 - o(1))$, every iteration terminates after $O(\log nm)$ phases.
3. Each of the steps in Figure 1 can be executed in time $\log nmR$, where R is the length of the largest set cost in bits.

These three assertions imply that PARALLEL SETCOV is in RNC^3. Indeed, it follows that PARALLEL SETCOV runs in $(\log n)(\log nm)(\log nmR)$ time on any standard PRAM or circuit model with access to coins.

The third assertion follows from the parallel complexity of integer arithmetic and parallel prefix computations. The details of these operations can be found in a number of standard texts on parallel computation (see [Le92], for instance).

In order to prove the first assertion, we have to detail the preprocessing step. Define $\beta = \max_e \min_{S \ni e} c_S$. Then, $\beta \leq$ cost of optimal cover $\leq n\beta$. The first inequality is because any cover has to pick some set that contains e^*, the maximizing element for β. The second inequality holds because for any arbitrary element e, there is a set of cost at most β containing e. Thus, there is a cover comprising of all these sets of cost at most $n\beta$.

Thus, any set S such that $c_S \geq n\beta$ could not possibly be in the optimum cover. The preprocessing step eliminates all such sets from consideration. Further, if for any element e there is a set S containing it with cost less than $\frac{\beta}{n}$, then we will add S to the set cover immediately. Since there are at most n elements, at most n sets can be added in this manner. These can all be added in parallel and the total cost incurred is at most an additional β. Since β is a lower bound on the cost of the set cover, the additional cost is subsumed in the approximation.

Thus, we can assume that for each set surviving the preprocessing stage, $c_S \in [\frac{\beta}{n}, n\beta]$. This consequence of the preprocessing stage is a key ingredient in establishing the following lemma.

LEMMA 4.1.1. PARALLEL SETCOV *requires at most* $O(\log n)$ *iterations.*

Proof. Define $\alpha \doteq \min_e \text{value}(e)$. At any point in the current iteration, consider a set S that has not yet been included in the set cover and such that $c_S \leq 2\alpha|U(S)|$. Then, by the definition of α,

$$\sum_{e \in U(S)} \text{value}(e) \geq |U(S)|\alpha \geq \frac{c_S}{2}.$$

Thus S satisfies ($\bullet$) and must have satisfied it at the inception of the iteration. Thus $S \in \mathcal{L}$. However, at the end of an iteration, $\mathcal{L}$ is empty. Thus for every remaining S, $c_S \geq 2\alpha|U(S)|$, or alternately, $\frac{c_S}{|U(S)|} \geq 2\alpha$. This ensures that in the next iteration, α increases by a factor of 2. Since α is at least $\frac{\beta}{n^2}$ and is at most $n\beta$, there can be no more than $\log n^3 = 3 \log n$ iterations. □

We will now show that the number of phases in every iteration is small with high probability. To this end, we will show the following lemma.

LEMMA 4.1.2. *Consider a fixed iteration, i. The probability that the number of phases in iteration i exceeds $O(\log nm)$ is smaller than $\frac{1}{n^2}$.*

Proof. We will focus our attention only on those sets that are in $\mathcal{L}$. Thus, the precondition $\sum_{e \in U(S)} \text{value}(e) \geq \frac{c_S}{2}$ is imposed on all sets that participate in this proof unless otherwise specified. The proof of this lemma is made via a potential function argument. The potential function is $\Phi \doteq \sum_{S \in \mathcal{L}} |U(S)|$. In other words, it is the number of uncovered element-set pairs in $\mathcal{L}$. In what follows, we will show that Φ decreases by a constant fraction (in expectation) after each phase. Since the initial value of Φ is at most nm, the lemma will follow from standard arguments.

Define the *degree* $\deg(e)$ of an element as

$$\deg(e) \doteq |\{S \in \mathcal{L} : S \ni e\}|.$$

Call a set-element pair $(S, e), e \in U(S)$ *good* if $\deg(e) \geq \deg(f)$ for at least three quarters of the elements in $U(S)$.

Let $e, f \in U(S)$ such that $\deg(e) \geq \deg(f)$. Let N_e, N_f, and N_b be the number of sets that contain e but not f, f but not e, and both e and f, respectively. Thus, $\text{prob}(e \text{ votes } S) = \frac{1}{N_e + N_b}$. The probability that both e and f vote for S is exactly the

probability that S is the first among $N_e + N_f + N_b$ sets which is exactly $\frac{1}{N_e+N_f+N_b}$. Since $\deg(e) \geq \deg(f)$ implies that $N_e \geq N_f$ we have

$$\text{prob}\,[f \text{ votes } S \mid e \text{ votes } S] = \frac{\text{prob}\,[e \text{ and } f \text{ vote } S]}{\text{prob}\,[e \text{ votes } S]} \geq \frac{N_e + N_b}{N_e + N_b + N_f} > \frac{1}{2}.$$

The statement above provides the heart of the proof, since it implies that if (S, e) is good, then S should get a lot of votes if e votes for S. Thus, under this condition, S should show a tendency to get added to the set cover. We shall now make this formal.

Noting that for any $f \in U(S)$, $\text{value}(f) \leq \frac{c_S}{|U(S)|}$, we see that for any good (S, e), $\sum_{f \in U(S), \deg(f) > \deg(e)} \leq \frac{c_S}{4}$. Since S satisfies $(\bullet)$, we obtain

$$\sum_{f \in U(S), \deg(f) \leq \deg(e)} \text{value}(f) \geq \frac{c_S}{2} - \frac{c_S}{4} = \frac{c_S}{4}.$$

The conditional probability statement above allows us to infer that if (S, e) is good and e votes for S, then the expected value of $\sum_{f \in U(S), f \text{ votes } S} \text{value}(f)$ is at least $\frac{c_S}{8}$. An application of Markov's inequality will show that the probability that S is picked is at least $\frac{1}{15}$.

We will ascribe the decrease in Φ when an element e votes for a set S and S is subsequently added to the set cover, to the set-element pair (S, e). The decrease in Φ that will then be ascribed to (S, e) is $\deg(e)$, since Φ decreases by 1 for each set containing e. Since e voted for only one set, any decrease in Φ is ascribed to only one (S, e) pair. Thus, the expected decrease in Φ, denoted $\Delta\Phi$, is at least

$$\begin{aligned}
E(\Delta\Phi) &\geq \sum_{(S,e): e \in U(S)} \text{prob}(e \text{ voted for } S,\ S \text{ was picked}) \cdot \deg(e) \\
&\geq \sum_{(S,\,e) \text{ good}} \Big(\text{prob}(e \text{ voted for } S) \times \text{prob}(S \text{ was picked} \mid e \text{ voted for } S) \times \deg(e)\Big) \\
&\geq \sum_{(S,\,e) \text{ good}} \frac{1}{\deg(e)} \frac{1}{15} \deg(e) \\
&= \frac{1}{15}(\text{number of good } (S, e) \text{ pairs}).
\end{aligned}$$

Finally since at least a quarter of all relevant (S, e) pairs are good, we observe that $E(\Delta\Phi) \geq \frac{1}{60}\Phi$. We recall the following fact from probability theory.

Fact 4.1.3. Let $\{X_t\}$ be a sequence of integer-valued and nonnegative random variables such that $E(X_t - X_{t+1} | X_t = x) \geq cx$ for some constant c. Let $Y \doteq \min_k\{X_k = 0\}$. Then, $\text{prob}(Y > O(\log(pX_0))) \leq p$. Here the asymptotic notation hides the dependence on $\frac{1}{c}$ which is linear. □

Notice that we have just established that the evolution of Φ satisfies the preconditions that allow us to apply Fact 4.1.3. Therefore, choosing $p = \frac{1}{n^2}$, we have our lemma. □

THEOREM 4.1. PARALLEL SETCOV *finds a cover which is at most* $16H_n$ *times the optimal set cover. Further, the total running time is* $O(\log n \cdot \log nm \cdot \log nmR)$ *with probability* $1 - \frac{1}{n}$.

Comment. The constant 16 can be improved to $2(1+\epsilon)$ for any $\epsilon > 0$. This is done by changing the number $\frac{c_S}{16}$ in step (c) to $\frac{c_S}{2(1+\epsilon)}$ and the quantity in the definition of $(\bullet)$ to $c_S(1-\epsilon^2)$. □

Comment. The conditional statement is a correlation inequality which we feel should be of independent interest. Generalizing this correlation inequality will be a central issue in our analysis of parallel algorithms for the generalizations of set cover. □

5. Set multicover and multiset multicover. The set multicover problem is a natural generalization of the set cover problem. In this generalization, each element e is associated with a coverage requirement, r_e, which indicates the depth to which e must be covered by any feasible cover. Thus, the set multicover problem can be formulated as an integer program as follows: MIN $\sum_S c_S x_S$ subject to $\sum_{S \ni e} x_S \geq r_e$ and $x_S \in \{0,1\}$.

Multiset multicover is the generalization where, in addition to the coverage requirement, each element e appears in any set S with a multiplicity $m(S,e)$. Thus, the integer program is MIN $\sum_S c_S x_S$ subject to $\sum m(S,e)x_S \geq r_e$ and $x_S \in \{0,1\}$.

On relaxing the integrality requirement on x_S we obtain the following linear program and dual in the case of multiset multicover.

LP :		**DP** :	
MIN	$\sum_S c_S x_S$	MAX	$\sum_e r_e y_e - \sum_S z_S$
s.t.	$\sum_S m(S,e)x_S \geq r_e$,	s.t.	$\sum_e m(S,e)y_e - z_S \leq c_S$,
	$-x_S \geq -1$,		$z_S \geq 0$,
	$x_S \geq 0$;		$y_e \geq 0$.

In the case of set multicover, $m(S,e)$ is simply the indicator function that takes a value of 1 if $e \in S$ and 0 otherwise. The interesting feature here is the need to explicitly limit the value of x_S to at most 1 and the associated appearance of the dual variables z_S. Notice that in the set cover problem the limit of 1 on the value of x_S was implicit.

In the following discussion, we shall largely restrict our attention to the more general case of multiset multicover. In some places, it is possible to obtain slightly stronger results in the case of set multicover. We will indicate these at appropriate points in the text.

5.1. Greedy algorithms. There is a natural greedy sequential algorithm for multiset multicover. Like the set cover algorithm, this algorithm works by repeatedly picking sets until all the coverage requirements are met. At an intermediate stage of this process, let $r(e)$ be the residual requirement of e. Thus, $r(e)$ is initially r_e and is decremented by $m(S,e)$ each time a set S is added to the cover. Define $a(S,e) \doteq \min\{m(S,e), r(e)\}$ and the set of alive elements in S, $A(S)$ to be the multiset containing exactly $a(S,e)$ copies of any element e if S is not already in the set cover, and the empty set if it is. The greedy algorithm repeatedly adds a set minimizing $\frac{c_S}{|A(S)|}$ to the set cover.

We will extend this notation to multisets in general. Thus, we will denote $|A(S)| \doteq \sum_e a(S,e)$. Let $k = \max_S \sum_e m(S,e)$. Let K denote $\sum_e r_e$. Dobson [Do82] shows that this natural extension of the greedy algorithm to set multicover achieves an approximation ratio of H_K. However, he does this by comparing the cost of the multicover obtained directly to the optimum integral solution of the set multicover problem. In what follows, we will establish an approximation ratio of H_k by comparing

the greedy solution to the best possible fractional multicover. Since we can always restrict $m(S,e)$ to at most r_e, $k \leq K$. Thus, this represents a slight improvement over Dobson's result.[3]

When a set S is picked, its cost c_S is ascribed equally to each tuple (e,i) where S covers e for the ith time. Here, i ranges from 1 to r_e. We will say S covers (e,i), in short, to describe this case. Obviously, the cost assigned to each tuple is exactly $\text{cost}(e,i) = \frac{c_S}{|A(S)|}$. Now, we choose $y_e = \frac{\max_i\{\text{cost}(e,i)\}}{H_k} = \frac{\text{cost}(e,r_e)}{H_k}$. If a set S is not picked, let $z_S = 0$, and otherwise let

$$z_S = \frac{\sum_{(e,i) \text{ covered by } S} (\text{cost}(e,r_e) - \text{cost}(e,i))}{H_k}.$$

The value of this dual assignment is easily verified to be the cost of the greedy multicover divided by H_k.

LEMMA 5.1.1. $\mathbf{y}, \mathbf{z}$ *is dual feasible.*

Proof. First, trivially, both $\mathbf{y}$ and $\mathbf{z}$ are nonnegative. Consider for any S,

$$\begin{aligned}
&\left(\sum_e m(S,e)y_e\right) - z_S \\
&\quad = \frac{1}{H_k}\left(\sum_e m(S,e)\text{cost}(e,r_e) - \sum_{(e,i) \text{ covered by } S} (\text{cost}(e,r_e) - \text{cost}(e,i))\right) \\
&\quad = \frac{1}{H_k}\left(\sum_{(e,i) \text{ covered by } S} \text{cost}(e,i) + \sum_{e \in S, \text{ not covered by } S} \text{cost}(e,r_e)\right).
\end{aligned}$$

We want to view a multiset S as a set which contains $m(S,e)$ copies of element e. Notice that there is a term in the right-hand side of the above expression corresponding to each element copy S. Thus, there are $m(S,e)$ terms corresponding to e. Let us arrange the element copies in S in the reverse order in which they were covered, for instance, if $m(S,e) = 10$ and $m(S,f) = 5$, and suppose $r(e)$ fell to 9 before $r(f)$ fell to 4 before $r(e)$ fell to 8. Then, the ninth copy of e precedes the fifth copy of f which precedes the tenth copy of e in this reverse ordering. Notice that the term corresponding to the jth element in this ordering is at most $\frac{c_S}{j}$. Thus, we have

$$\left(\sum_e m(S,e)y_e\right) - z_S \leq \frac{1}{H_k}\left(\sum_{i=1}^{k} \frac{1}{i}\right) c_S.$$

In other words, the dual constraint corresponding to S is satisfied. Thus, we establish the feasibility of $\mathbf{y}$, $\mathbf{z}$ for the dual problem. □

The consequence of the above arguments is the following theorem.

THEOREM 5.1. *The extended greedy algorithm finds a multiset multicover within an H_k factor of* $\mathbf{LP}^*$.

[3]Recall that in the case of set cover, the approximation factor is logarithmic in the size of the largest set and not just in n, the size of the universe. Similarly, in this case, the ratio is logarithmic in k, which is the "local size," as opposed to K, the "global size" of the problem.

PARALLEL MULTMULTCOV
Set $r(e) = r_e$ for each e.
Iteration:
For each element e, compute value$(e) = \min_{S \ni e} \frac{c_S}{|A(S)|}$.
For each set S: include S in $\mathcal{L}$ if
(•) $\sum_e a(S,e)\text{value}(e) \geq \frac{c_s}{2}$.
Phase:
Initialization:
(a) Permute $\mathcal{L}$ at random.
(b) Each element e votes for first $r(e)$ copies of itself in the random ordering of $\mathcal{L}$.
(c) If $\sum_{e \text{ votes } S} \text{value}(e) \geq \frac{c_S}{128}$, S is added to the set cover.
(d) Decrement $r(e)$ appropriately. Adjust $a(S,e)$ as required. Delete sets that are picked or fail to satisfy (•) from $\mathcal{L}$.
Repeat until $\mathcal{L}$ is empty.
Iterate until all elements are covered.

FIG. 2. *The parallel multiset multicover algorithm.*

5.2. Parsimonious accounting. More pertinently, the proof implies that the parsimonious accounting principle ensures approximation in the case of multiset multicover as well. By this we mean the following. Define the dynamic quantity value$(e) = \min\{\frac{c_S}{|A(S)|}\}$. Then, as long as we can assign costs cost(e,i) where i ranges from 1 to r_e such that

1. cost$(e,i) \leq \mu$value(e) at the moment that the set S covering (e,i) is picked,
2. $\sum_{(e,i)} \text{cost}(e,i) \geq$ cost of cover,

then the algorithm approximates the value of the multiset multicover to within μH_k. Here k is the largest set size, i.e., $\max_S \sum_e m(S,e)$. We note that in the case of set multicover, k is at most n, and thus we would have an H_n approximation. Moreover, in many instances, k could be substantially smaller than n.

5.3. Parallel algorithms and analysis. We now outline parallel algorithms for multiset multicover. The parallel multiset multicover algorithm is essentially the same as the set cover algorithm except that with each element we associate a dynamic variable, $r(e)$, initially r_e, which tracks the residual requirement of e. After a random permutation of the candidate sets $\mathcal{L}$ is chosen, each element votes for the first $r(e)$ copies of itself in the sequence.[4] The algorithm is detailed in Figure 2. Notice that we can assume without loss of generality that $r(e) \leq \deg(e)$.

5.4. Analysis. It is easy to see that the algorithm satisfies that parsimonious accounting property with $\mu = 128$. This establishes the approximation ratio.

The number of iterations is bounded by $O(\log mnk)$. As earlier, we will denote the cost of the optimum multiset multicover by $\mathbf{IP}^*$. The proof follows exactly along the lines of the proof for the set cover case. The only change required for proving this is in the definition of the crude estimator β: let $S_1, S_2 \cdots S_m$ be the sets arranged in

[4]For example, $r(e)$ is 4 and $m(S_1,e) = 2$, $a(S_2,e) = 3$, and $a(S_3,e) = 2$. Let the permutation be $S_1 < S_2 < S_3$. Then, e votes for S_1 twice and S_2 twice, casting a total of four votes. If the total number of copies of e in the candidate sets is less than $r(e)$, then e votes for them all.

increasing order of cost. Let β_e be the cost of the set containing the r_eth copy of e, and let $\beta = \max_e \beta_e$. Then, $\beta \leq \mathbf{IP}^* \leq mn\beta$. As before, we can restrict attention to the sets such that $c_S \in [\beta/m, mn\beta]$. Again, it can be easily established that $\min_e \text{value}(e)$ increases by a constant factor after each iteration. Since value(e) for any element is at least β/mnk and at most $mn\beta$, there can be only $O(\log mnk)$ iterations.

Notice that for the special case of set multicover, k is at most n. Thus, the bound is $\log mn$ iterations.

5.4.1. Phases in an iteration. The number of phases required in an iteration is at most $O(\log^2 nm)$. This is established by extending the corresponding lemma for set cover. However, the extension is non-trivial and we shall need some machinery to do this.

First we restrict our attention to the sets in $\mathcal{L}$, i.e., sets that satisfy (•). In the following discussion, we will denote the copies of element e in the set S by $e^{(i)}$. Here, i ranges from 1 to $a(S, e)$. We say that $e^{(i)}$ votes for S if $e^{(i)}$ is among the first $r(e)$ copies of e in the random ordering of $\mathcal{L}$.

We will now introduce some notation that will simplify our analysis. We denote by $r(e, i) \doteq r(e) - i + 1$. We denote by $\deg(e, i) \doteq \sum_{S \in \mathcal{L}} \min\{a(S, e), r(e, i)\}$. Notice that from the definitions, it follows that $r(e) = r(e, 1)$ and $\deg(e) = \deg(e, 1)$. The second since $a(S, e)$ is at most $r(e)$. In general, it is tricky to get a handle on the probability that $e^{(i)}$ obtains a vote for S. However, we shall now show the following lemma which says that the quantity $\frac{r(e,i)}{\deg(e,i)}$ approximates this quantity quite nicely.

LEMMA 5.4.1.

$$\frac{1}{2}\frac{r(e,i)}{\deg(e,i)} \leq \text{prob}(e^{(i)} \textit{ votes } S) \leq 4\frac{r(e,i)}{\deg(e,i)}.$$

Proof. The proof has two parts. The first establishes the upper bound, and the second, the lower bound. For the upper bound we notice that with each permutation of the sets such that $e^{(i)}$ votes S, we can associate at least $\frac{\deg(e,i)}{2r(e,i)} - 2$ permutations such that (e, i) does not vote for S. In order to make this association, consider the notion of a "rotation," exhibited by Figure 3. This figure is to be interpreted as

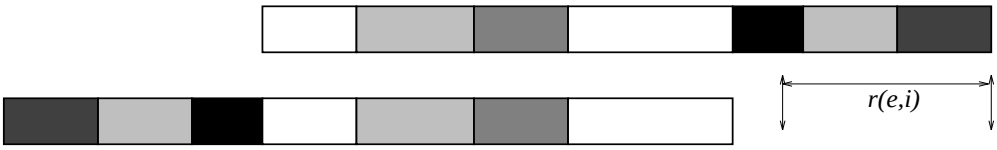


FIG. 3. *The top bar shows a permutation of $\mathcal{L}$, and the bottom, a rotated permutation.*

follows: the figure shows two permutations of $\mathcal{L}$. These two permutations differ by a "rotation." The shaded rectangles representing each permutation represent the various sets in $\mathcal{L}$. The length of these rectangles correspond to their contributions to $\deg(e, i)$. Thus, the total length of the set of rectangles representing any permutation of $\mathcal{L}$ is exactly $\deg(e, i)$.

A rotation is made by extracting from the end of the permutation a minimum number of sets such that they contribute at least $r(e, i)$ towards $\deg(e, i)$ and then placing them in reversed order in the front of the permutation (as shown).

It is easily verified that a rotation is reversible; i.e., it is possible to "unrotate" any rotated permutation to the original permutation. This is most easily seen by turning

Figure 3 upside down. More formally, the unrotation is performed by reversing the sequence, rotating, and then reversing again.

It is also easily verified that for any configuration such that $e^{(i)}$ votes S, it is possible to rotate at least $\frac{\deg(e,i)}{2r(e,i)} - 2$ times such that for each rotated configuration, $e^{(i)}$ does not vote S. This is because the maximum contribution of any set towards $\deg(e,i)$ is $r(e,i)$. Thus, we have associated with each voting permutation at least $\frac{\deg(e,i)}{2r(e,i)} - 2$ nonvoting permutations. Thus, the probability that $e^{(i)}$ votes for S is at most $(\frac{\deg(e,i)}{2r(e,i)} - 1)^{-1} = \frac{2r(e,i)}{\deg(e,i)-2r(e,i)}$. It can be seen via some simple algebraic manipulations that this implies that $\text{prob}(e^{(i)} \text{ votes } S) \leq \frac{4r(e,i)}{\deg(e,i)}$.[5]

For the second part, we need to do some simple analysis. Let us imagine that we permute $\mathcal{L}$ by choosing at random $X_T \in [0,1]$ for each set T and then sorting $\mathcal{L}$ in increasing order of X_T. Notice that we do not need to do this algorithmically; we introduce this just as a means to get a handle on the probability that interests us. Define $Y(x) \doteq \sum_{X_T<x} \min\{a(T,e), r(e,i)\}$. Then, it is easily verified that the events ($e^{(i)}$ votes S) and ($Y(X_S) \leq r(e,i)$) are equivalent. Since for any x, $E(Y(x)) = x\deg(e,i)$, we have by Markov's inequality,

$$\text{prob}(e^{(i)} \text{ votes } S \mid X_S = x) \geq 1 - \text{prob}(Y(x) \geq r(e,i) | X_S = x) \geq 1 - \frac{\deg(e,i)}{r(e,i)}x.$$

Let $\theta \in [0,1]$; then

$$\text{prob}(e^{(i)} \text{ votes } S) \geq \int_0^\theta \text{prob}(e^{(i)} \text{ votes } S \mid X_S = x)dx \geq \theta - \frac{\theta^2}{2}\frac{\deg(e,i)}{r(e,i)}.$$

Choosing $\theta = \frac{r(e,i)}{\deg(e,i)}$ completes the proof, since by our assumption (which, as we pointed out, can be made without loss of generality) that $r(e) \leq \deg(e)$, this fraction is smaller than 1. Otherwise, the lemma is trivial. □

LEMMA 5.4.2. *Let* $e^{(i)}, f^{(j)} \in S$. *Then*

$$\text{prob}\left(e^{(i)} \textit{ and } f^{(j)} \textit{ vote } S\right) \geq \frac{1}{2}\left(\frac{1}{\frac{\deg(e,i)}{r(e,i)} + \frac{\deg(f,j)}{r(f,j)}}\right).$$

Proof. The proof of this lemma is very similar to the proof of Lemma 5.4.1.

$$\begin{aligned}
&\text{prob}\,(e \text{ and } f \text{ vote } S)\\
&= \int_0^1 \text{prob}\,(e \text{ and } f \text{ vote } S | X_S = x)\,dx\\
&\geq \int_0^\theta \text{prob}\,(e \text{ and } f \text{ vote } S | X_S = x)\,dx, \quad \theta \in [0,1],\\
&\geq \int_0^\theta 1 - \text{prob}\,(e \text{ does not vote } S | X_S = x) - \text{prob}\,(f \text{ does not vote } S | X_S = x)\,dx.
\end{aligned} \tag{1}$$

[5] To see this, we consider two cases. If $\deg(e,i) \leq 4r(e,i)$, then the conclusion is trivial. Otherwise, it is easy to see that $\frac{2r(e,i)}{\deg(e,i)-2r(e,i)} \leq 4\frac{r(e,i)}{\deg(e,i)}$ by crossmultiplication

Define $Y_e(x)$ and $Y_f(x)$ as in the previous lemma,

$$\text{prob}(e^{(i)} \text{ does not vote for } S \mid X_S = x) \leq \frac{x \cdot \deg(e,i)}{r(e,i)},$$

and do similarly for $f^{(j)}$. Substituting these values back into (1), we obtain

$$\text{prob}(e^{(i)} \text{ and } f^{(j)} \text{ vote for } S) \geq \theta - \left(\frac{\deg(e,i)}{r(e,i)} + \frac{\deg(f,j)}{r(f,j)}\right) \cdot \frac{\theta^2}{2}.$$

Choosing the value of θ to maximize the right-hand side, we get[6]

$$\text{prob}\left(e^{(i)} \text{ and } f^{(j)} \text{ vote } S\right) \geq \frac{1}{2}\left(\frac{1}{\frac{\deg(e,i)}{r(e,i)} + \frac{\deg(f,j)}{r(f,j)}}\right). \qquad \square$$

LEMMA 5.4.3. *Let* $e^{(i)}, f^{(j)} \in S$ *such that* $\frac{r(e,i)}{\deg(e,i)} \leq \frac{r(f,j)}{\deg(f,j)}$.

$$\text{prob}(f^{(j)} \textit{ votes } S \mid e^{(i)} \textit{ votes } S) \geq \frac{1}{8}.$$

Proof. The proof is immediate from the previous two lemmas. $\square$

Remark. In the case of set multicover, the first of the two lemmas implying Lemma 5.4.3 is trivial. Indeed, since $a(S,e)$ is either 0 or 1, we can immediately infer that $\text{prob}(e \text{ votes } S) = \frac{r(e)}{\deg(e)}$. The second lemma (with i and j both set to 1) immediately implies the corresponding version of Lemma 5.4.3 with a bound of $\frac{1}{4}$. $\square$

Say that a set-element pair $(S, e^{(i)})$ is *good* if $\frac{\deg(e,i)}{r(e,i)} \geq \frac{\deg(f,j)}{r(f,j)}$ for at least three quarters of the elements $f^{(j)} \in S$. Then, as in the case of set cover, Lemma 5.4.3 implies that if $(S, e^{(i)})$ is good

$$\text{prob}\left(S \text{ is picked} \mid e^{(i)} \text{ votes for } S\right) \geq p,$$

where $p > 0$. The potential function we use is

$$\Phi \doteq \sum_e \deg(e) H_{r(e)} = \sum_e \deg(e) \sum_{i=1}^{r(e)} \frac{1}{r(e,i)}.$$

Initially, $\Phi \leq mn \log r$, where r is the largest requirement. The expected decrease in Φ ascribable to a good set-element pair $(S, e^{(i)})$, denoted $\Delta\Phi_{(S,e^{(i)})}$, is at least

$$\begin{aligned} E(\Delta\Phi_{(S,e^{(i)})}) &\geq \text{prob}(e^{(i)} \text{ voted for } S) \times \text{prob}(S \text{ was picked} \mid e \text{ voted for } S) \frac{\deg(e)}{r(e,i)} \\ &\geq \frac{r(e,i)}{\deg(e,i)} \frac{p}{2} \frac{\deg(e)}{r(e,i)} \\ &\geq \frac{p}{2}. \end{aligned}$$

[6] Again, the assumption that $r(e) \leq \deg(e)$ implies that our choice of θ is at most 1.

The first inequality is from the definition of Φ, the second is due to Lemmas 5.4.1 and 5.4.3, and the last is due to $\deg(e) \geq \deg(e, i)$. Since a constant fraction of all $(S, e^{(i)})$ are good, $\mathrm{E}\Delta\Phi \geq O(\frac{\Phi}{\log r})$, where r is the largest requirement value. From Fact 4.1.3, we know that the total number of phases is at most $O(\log r(\log mn + \log\log r))$.

THEOREM 5.2. PARALLEL MULTMULTCOV *approximates multiset multicover to within* $128H_n$*, using a linear number of processors and running in time* $O(\log^4 mnr)$.

COROLLARY 5.4.4. *If the number in step* (c) *of Figure* 2 *were changed to* $\frac{c_S}{32}$*, and the algorithm were run on an instance of set multicover, then it would be a* RNC^4 *algorithm approximating set multicover to within* $32H_n$.

Proof. The proof follows from the remark following Lemma 5.4.3. □

6. Covering integer programs. Covering integer programs are integer programs of the following form:

$$\textbf{CIP}: \quad \begin{array}{ll} \text{MIN} & \sum_S c_S x_S \\ \text{s.t. } \forall e & \sum_S M(S,e) x_S \geq r_e, \\ & x_S \in \bar{\wedge}^+. \end{array}$$

Here, $\bar{\wedge}^+$ are the nonnegative integers. The vectors $\mathbf{c}$ and $\mathbf{r}$ and the matrix M are all composed of nonnegative (rational) numbers. At this stage, the notion of a "set" does not have any meaning, however, we continue to use this notation simply to maintain consistency with the previous discussion. For the purpose of the subsequent discussion, the reader should keep in mind that S varies over one set of indices, and e, over the other. Without loss of generality, we may assume that $M(S, e) \leq R_e$.

What we present here is a scaling and rounding trick. The goal is to reduce to an instance of multiset multicover with polynomially bounded (and integral) element multiplicities and requirements. Moreover, in making this reduction, there should be no significant degradation in the approximation factor.

LEMMA 6.0.5. *There is an* NC^1 *reduction from covering integer programs to multiset multicover with element multiplicities and requirements at most* $O(n^2)$ *such that the cost of the fractional optimal (i.e., the cost of the LP relaxation) goes up by at most a constant factor.*

Proof. Essentially, we need to reduce the element requirements to a polynomial range and ensure that the requirements and multiplicities are integral. Then, replicating sets to the extent of the largest element requirement, we would get an instance of multiset multicover. We will achieve this as follows: we will obtain a (crude) approximation to the fractional optimal within a polynomial factor, and then we will ensure that the costs of sets are in a polynomial range around this estimate. Also, we will set all element requirements at a fixed polynomial and set the multiplicities accordingly. At this point, rounding down the multiplicities will change the fractional optimal by only a constant factor.

Let $\beta_e = R_e \cdot \min_S \frac{c_S}{M(S,e)}$ and $\beta = \max_e \beta_e$. Clearly, $\beta \leq \mathbf{CLP}^* \leq n\beta$, where $\mathbf{CLP}^*$ is the cost of the optimal solution to the LP relaxation of **CIP**. If a set S has large cost, i.e., $c_S > n\beta$, then S will not be used by the fractional optimal, and we will eliminate it from consideration. So, for the remaining sets, $c_S \leq n\beta$. Define $\alpha_S = \lceil \frac{\beta}{nc_S} \rceil$. We will clump together α_S copies of S (i.e., $M(S, e) \leftarrow M(S, e)\alpha_S$, $c_S \leftarrow c_S\alpha_S$). The cost of the set so created is at least $\frac{\beta}{n}$. Additionally, the fractional optimum is not affected by this scaling (though the same cannot be said of the integral optimal). Thus, we can assume that for each S, $c_S \in [\frac{\beta}{n}, n\beta]$.

If any element is covered to its requirement by a set of cost less than $\frac{\beta}{n}$, cover the element using that set and eliminate the element from consideration. The cost incurred in the process is at most β for all elements so covered, and this is subsumed in the constant factor. Notice that as a result of this step, the multiplicity of an element in a set will still be less than its requirements. The reason we require α_S to be integral is that we need to map back solutions from the reduced problem to the original problem. Henceforth, we can assume that the costs satisfy $c_S \in [\frac{\beta}{n}, n\beta]$. Next, we will fix the requirement of each element to be $2n^2$, and we will set the multiplicities appropriately, $m'(S,e) = \frac{M(S,e)}{r_e} \cdot 2n^2$. Since this is just multiplying each inequality by a constant, this will not change the (both fractional and integral) optimal solution or value.

Finally, we will round down the multiplicities, $m(S,e) = \lfloor m'(S,e) \rfloor$. We will show that this will increase the fractional optimal by a factor of at most 4. The same cannot be said for the integral optimum. This is the reason why we needed to compare the solution obtained by our approximation algorithms for multiset multicover to the fractional optimum.

Consider an optimal fractional solution to the problem with multiplicities $m'(S,e)$. Since the cost of this solution is at most $n\beta$, and the c_S is at least $\frac{\beta}{n}$, $\sum_S x_S \leq n^2$. Consider an element e, and let $\mathcal{S}$ be the collection of all sets S such that $m'(S,e) < 1$. Then, the total coverage of e due to sets in $\mathcal{S}$ is at most n^2. Therefore, the total coverage of e due to the remaining sets is at least n^2. Since for each of these sets, $m(S,e) \geq \frac{1}{2} m'(S,e)$, if we multiply each x_S by 4, element e will be covered to the extent of at least $2n^2$ in the rounded-down problem. The lemma follows. □

Notice that the reduction in Lemma 6.0.5 is such that a feasible solution to the instance of the multiset multicover problem can be mapped back to a feasible solution of the original problem without increasing the cost. Hence we get the following theorem.

THEOREM 6.1. *There is an $O(\log n)$ factor RNC^4 approximation algorithm for covering integer programs requiring $O(n^2 \#(A))$ processors, where $\#(A)$ is the number of nonzero entries in A.*

Since, in the reduction, all element requirements are set to $O(n^2)$, we obtain the processor bound stated above. Further, since we are comparing the performance of our algorithm with the fractional optimal, it follows that the integrality gap for covering integer programs, in which element multiplicities are bounded by requirements, is bounded by $O(\log n)$. It is easy to see that if multiplicities are not bounded by requirements, the integrality gap can be arbitrarily high.

The previous best (sequential) approximation guarantee for covering integer programs was $H_{nmax(A)}$, where $\max(A)$ is the largest entry in A, assuming that the smallest one is 1 [Do82]. Moreover, in that result, the performance of the algorithm given was compared to the integral optimal.

Notice that the multiset multicover problem with the restriction that each set can be picked at most once is not a covering integer program, since we will have negative coefficients. This raises the question of whether there is a larger class than covering integer programs for which we can achieve (even sequentially) an $O(\log n)$ approximation factor.

Acknowledgments. We thank the referees. Their comments greatly improved the presentation of the paper.

REFERENCES

[BRS89] B. Berger, J. Rompel, and P. Shor, *Efficient NC algorithms for set cover with applications to learning and geometry*, in Proc. 30th IEEE Symposium on the Foundations of Computer Science, 1989, pp. 54–59.

[Ch79] V. Chvatal, *A greedy heuristic for the set covering problem*, Math. Oper. Res., 4 (1979), pp. 233–235.

[Do82] G. Dobson, *Worst-case analysis of greedy heuristics for integer programming with nonnegative data*, Math. Oper. Res., 7 (1982), pp. 515–531.

[Fe96] U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, in Proc. 28th ACM Symposium on the Theory of Computing, 1996, pp. 312–318.

[Jo74] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.

[Ka72] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.

[LN93] M. Luby and N. Nisan, *A parallel approximation algorithm for positive linear programming*, in Proc. 25th ACM Symposium on Theory of Computing, 1993, pp. 448–457.

[Lo75] L. Lovasz, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13, pp. 383–390.

[Le92] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufman, San Francisco, 1992.

[LY93] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, in Proc. 25th ACM Symposium on Theory of Computing, 1993, pp. 286–293.

[PST91] S. A. Plotkin, D. B. Shmoys, and E. Tardos, *Fast approximation algorithms for fractional packing and covering problems*, in Proc. 32nd IEEE Symposium on the Foundations of Computer Science, 1991, pp. 495–504.

[Ra88] P. Raghavan, *Probabilistic construction of deterministic algorithms: Approximating packing integer programs*, J. Comput. System Sci., 37 (1988), pp. 130–143.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 541–573

OPTIMAL CONSTRUCTION OF EDGE-DISJOINT PATHS IN RANDOM GRAPHS*

ANDREI Z. BRODER†, ALAN M. FRIEZE‡, STEPHEN SUEN§, AND ELI UPFAL¶

Abstract. Given a graph $G = (V, E)$ with n vertices, m edges, and a family of κ pairs of vertices in V, we are interested in finding for each pair (a_i, b_i) a path connecting a_i to b_i such that the set of κ paths so found is edge disjoint. (For arbitrary graphs the problem is $\mathcal{NP}$-complete, although it is in $\mathcal{P}$ if κ is fixed.)

We present a polynomial time randomized algorithm for finding the optimal number of edge disjoint paths (up to constant factors) in the random graph $G_{n,m}$ for all edge densities above the connectivity threshold. (The graph is chosen first; then an adversary chooses the pairs of endpoints.) Our results give the first tight bounds for the edge-disjoint paths problem for any nontrivial class of graphs.

Key words. edge-disjoint paths, random graphs, eigenvalues of random graphs

AMS subject classifications. 05C38, 05C40, 05C80, 05C85, 60J15, 68Q20, 68Q25, 68R10, 90B10, 90B12

PII. S0097539795290805

1. Introduction.

Given a graph $G = (V, E)$ with n vertices, m edges, and a set of κ pairs of vertices in V, we are interested in finding for each pair (a_i, b_i) a path connecting a_i to b_i such that the set of κ paths so found is edge disjoint.

For arbitrary graphs the related decision problem is $\mathcal{NP}$-complete, although it is in $\mathcal{P}$ if κ is fixed (Robertson and Seymour [16]). Nevertheless, this negative result can be circumvented for certain classes of graphs. Peleg and Upfal [15] presented a polynomial time algorithm for the case where G is a (sufficiently strong) bounded degree expander graph, and $\kappa \leq n^{\epsilon}$ for a small constant ϵ that depends on the expansion property of the graph. (A precise upper bound for ϵ was not computed, but it is clearly less that 1/3.) This result has recently been improved by Broder, Frieze, and Upfal [8]: G still has to be a (sufficiently strong) bounded degree expander but κ can now grow as fast as $n/(\ln n)^{\theta}$, where θ depends only on the expansion properties of the input graph but is at least 7.

For the vertex-disjoint paths problem Kleinberg and Tardos [13] come within an $O(\log n)$ factor of the maximum possible κ for a class of planar graphs. In random graphs Shamir and Upfal have shown in [17] that any set of up to $O(\sqrt{n})$ pairs can be connected via vertex-disjoint paths; similar results using efficient flow techniques

*Received by the editors August 25, 1995; accepted for publication (in revised form) June 10, 1996; published electronically July 28, 1998. A preliminary version of this paper appeared in the *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994.

http://www.siam.org/journals/sicomp/28-2/29080.html

†Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301 (broder@src.dec.com).

‡Department of Mathematics, Carnegie-Mellon University, Pittsburgh, PA 15213 (af1p@euler.math.cmu.edu). A portion of this work was done while this author was visiting IBM Almaden. The work of this author was supported in part by NSF grants CCR9024935 and CCR9225008.

§Department of Mathematics, Carnegie-Mellon University, Pittsburgh, PA 15213 (suen@math.usf.edu).

¶IBM Almaden Research Center, San Jose, CA 95120, and Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel (eli@wisdom.weizmann.ac.il). Work at the Weizmann Institute was supported in part by the Norman D. Cohen Professorial Chair of Computer Science, by a MINERVA grant, and by the Israeli Academy of Science.

were also obtained by Hochbaum [12]. These results were proved for graphs with $m \geq Kn \log n$ random edges, where K is a sufficiently large constant.

Let D be the median distance between pairs of vertices in G. Clearly it is not possible to connect more than $O(m/D)$ pairs of vertices by edge-disjoint paths for all choices of pairs, since some choice would require more edges than all the edges available. In the case of bounded degree expanders, this absolute upper bound on κ is $O(n/\log n)$. The results mentioned above use only a vanishing fraction of the set of edges of the graph and thus are far from reaching this upper bound. In contrast, in this work we show that for the basic models of random graphs, $G_{n,m}$ and $G_{n,p}$, the absolute upper bound is achievable within a constant factor, and we present an algorithm that constructs the required paths in polynomial time.

As usual, let $G_{n,p}$ denote a random graph with vertex set $\{1, 2, \ldots, n\} = [n]$ in which each possible edge is included independently with probability p, and let $G_{n,m}$ denote a random graph also with vertex set $[n]$ and exactly m edges, all sets of m edges having equal probability. The degree of a vertex v is denoted by $d_G(v)$.

Our main result is formulated in the following theorem.

THEOREM 1. *Let $m = m(n)$ be such that $d = 2m/n \geq (1 + o(1)) \ln n$. Then as $n \longrightarrow \infty$ with probability $1 - o(1)$, the graph $G_{n,m}$ has the following property: there exist positive constants α and β such that for all sets of pairs of vertices $\{(a_i, b_i) \mid i = 1, \ldots, \kappa\}$ satisfying*

(i) $\kappa = \lceil \alpha m \ln d / \ln n \rceil$,

(ii) *for each vertex v, $|\{i : a_i = v\}| + |\{i : b_i = v\}| \leq \min\{d_G(v), \beta d\}$,*

there exist edge-disjoint paths in G, joining a_i to b_i, for each $i = 1, 2, \ldots, \kappa$. Furthermore, there is an $O(nm^2)$ time randomized algorithm for constructing these paths.

A similar result holds for $G_{n,p}$, with $d = np$ and $\kappa = \lceil \alpha n^2 p \ln d / (2 \ln n) \rceil$.

This result is the best possible up to constant factors. For (i) note that the distance between most pairs of vertices in G is $\Omega(\log n / \log d)$, and thus with m edges we can connect at most $O(m \log d / \log n)$ pairs. For (ii) note that a vertex v can be the endpoint of at most $d_G(v)$ different paths. Furthermore suppose that $d \geq n^\gamma$ for some constant $\gamma > 0$ so that $\kappa \geq \lceil \alpha \gamma n d / 2 \rceil$. Let $\epsilon = \alpha\gamma/3$, $A = [\epsilon n]$, and $B = [n] \setminus A$. Now with probability $1 - o(1)$ there are less than $(1 + o(1))\epsilon(1 - \epsilon)nd$ edges between A and B in $G_{n,m}$. However, almost all vertices of A have degree $(1 + o(1))d$ and if for these vertices we ask for $(1 - \epsilon/2)d$ edge-disjoint paths to vertices in B then the number of paths required is at most $(1 + o(1))\epsilon(1 - \epsilon/2)nd < \kappa$, but, without further restrictions, this many paths would require at least $(1 - o(1))\epsilon(1 - \epsilon/2)nd > (1 + o(1))\epsilon(1 - \epsilon)nd$ edges between A and B, which is more than what is available. This justifies an upper bound of $1 - \epsilon/2$ for β of Theorem 1.

We note that we have proved similar optimal results for the vertex-disjoint paths problem in random graphs [7].

The construction of $n/(\ln n)^\theta$ edge-disjoint paths on expander graphs that was described in [8] was achieved through the use of the Lovász local lemma [9]. Sets of possible paths were constructed for each pair, and the local lemma was applied to prove that there is a global choice of one path per set such that all the choices are edge disjoint. However, this approach can only be used when the total number of edges in the final set of disjoint paths is a vanishing fraction of the number of edges in the graph; inherently, it does not lead to optimal bounds.

Here we address the problem in a different way. After a randomization phase, similar to the one in [8], the disjoint paths are constructed one after the other, and all the edges seen during the construction are deleted from the graph. The paths

connecting each pair are chosen through a "random walk" type process. The crux of the analysis is to show that after a number of pairs have already been connected, the remaining graph is sufficiently connected to continue with this process. To prove that, we use a good estimate on the eigenvalues of the intermediate graphs generated by the algorithm. (Since we cannot throw logarithmic factors at our trouble spots, the proofs are rather intricate, although the algorithm itself is quite simple.) Eventually the number of pairs not yet connected becomes small enough that we can use [8] directly.

The disjoint paths problem has numerous algorithmic applications. One that has received increased attention in recent years is in the context of communication networks. The only efficient way to transmit high volume communication, such as in multimedia applications, is through disjoint paths that are dedicated to one pair of processors for the duration of the communication. To efficiently utilize the network one needs a very simple algorithm that with minimum overhead constructs a large number of edge-disjoint paths between a given set of requests. The algorithm we study is simple and easy to implement (after eliminating some steps that are needed only for the proof) and thus suggests some possibly good practical heuristics.

In section 3 we present a very brief overview of the algorithm. The details of the algorithm are exposed in section 4. The remainder of the paper gives the analysis.

2. Preliminaries. The paper contains a number of unspecified constants of which α and β above are the first. Exact values could be given, but it is easier for us *and the reader* if we simply give the relations between them. New constants will be introduced as $C_0, C_1, \ldots$ without further comment. Furthermore, specific constants have been chosen for convenience, we made no attempt to optimize them, and, in general, we only claim that inequalities dependent on n hold for n sufficiently large.

For a graph $G = (V, E)$ we use $\delta(G)$ and $\Delta(G)$ to denote the smallest and largest degrees, respectively. For a set $S \subseteq V$ we define its neighbor set, $N(S, G)$, as

$$N(S, G) = \{v \in V \setminus S : \exists w \in S \text{ such that } \{v, w\} \in E\}.$$

For $S \subseteq V$, we use $G[S]$ to denote the subgraph of G induced by S.

The *Chernoff bounds* on the tails of the binomial $\text{bin}(n, \theta)$ that we use are

$$\mathbf{Pr}(\text{Bin}(n, \theta) \leq (1 - \epsilon)n\theta) \leq e^{-\epsilon^2 n\theta/2}, \tag{1}$$

$$\mathbf{Pr}(\text{Bin}(n, \theta) \geq (1 + \epsilon)n\theta) \leq e^{-\epsilon^2 n\theta/3}, \tag{2}$$

valid for $0 \leq \epsilon \leq 1$.

3. Overview of the algorithm. Our algorithm divides naturally into the five phases sketched below.

Phase 1. Partition G into five edge-disjoint graphs $G_i = (V_i, E_i)$, $1 \leq i \leq 5$. Phase 2 will use only the graph G_1; Phase 3 will use only the graph G_2; Phase 4 will use only the graph G_3; and Phase 5 will use only the graphs G_4 and G_5. The partition is such that $V_1 = V$ but $V_2 = V_3 = V_4 = V_5 \subseteq V$ with $|V_2| = n - o(n)$.

Phase 2. Choose a random multiset $Z = \{z_1, \ldots, z_{2\kappa}\}$ of 2κ points in V_2. Connect the endpoints $\{(a_i, b_i) \mid i = 1, \ldots, \kappa\}$ to the newly chosen points in an arbitrary manner via edge-disjoint paths in G_1 using a flow algorithm. Let $\tilde{a}_i$ (resp., $\tilde{b}_i$) be the vertex connected to a_i (resp., b_i). The original problem is now reduced to finding edge-disjoint paths from $\tilde{a}_i$ to $\tilde{b}_i$ for each i. (This randomization was used in [8] and has its roots in Valiant's routing algorithm [19].)

Phase 3. For each $z \in Z$ in turn, we do a random walk of length $\tau = \lceil C_0 \ln n / \ln d \rceil$ in G_2, starting at z. We remove the edges of the jth walk before embarking on the $j+1$st. This keeps the paths constructed edge disjoint. The terminating endpoint of the walk starting at $\tilde{a}_i$ (resp., $\tilde{b}_i$) will be denoted by $\hat{a}_i$ (resp., $\hat{b}_i$) for $1 \le i \le \kappa$. The analysis below shows that in almost every $G_{n,p}$, the (multi)set of vertices $\hat{a}_1, \ldots, \hat{a}_\kappa, \hat{b}_1, \ldots, \hat{b}_\kappa$ is *not too far* from being independently, uniformly distributed.

Phase 4. For each i in turn, we repeatedly do a certain type of random walk in G_3 starting from $\hat{a}_i$ until one of these walks ends at $\hat{b}_i$. We keep the last walk as our path from $\hat{a}_i$ to $\hat{b}_i$ and remove from G_3 all edges seen in these walks. (The analysis below promises that this process will succeed whp[1] for most i.) Not every pair $(\hat{a}_i, \hat{b}_i)$ will be successfully connected in this phase, but the final path for each pair that succeeds is the concatenation of the paths from a_i to $\tilde{a}_i$, and from b_i to $\tilde{b}_i$ found in Phase 2, the paths from $\tilde{a}_i$ to $\hat{a}_i$ and $\tilde{b}_i$ to $\hat{b}_i$ found in Phase 3, and the path from $\hat{a}_i$ to $\hat{b}_i$ found here.

Phase 5. At the end of Phase 4, whp, there will be at most $n^{1-\epsilon}$ pairs $(\hat{a}_i, \hat{b}_i)$, for a constant $\epsilon > 0$, which have not been joined by paths. We use the algorithm of [8] to join them by edge-disjoint paths, using only the edges of G_4 and G_5, and then construct the final paths as above.

To prove Theorem 1 it suffices to show that for almost every $G_{n,p}$,

(i) Phases 1 and 2 will succeed for *all* choices of $a_1, \ldots, b_\kappa$ and *almost every* choice of $z_1, \ldots, z_{2\kappa}$;

(ii) Phases 3, 4, and 5 are successful for *almost every* choice of $z_1, \ldots, z_{2\kappa}$ and *any* mapping $\{\tilde{a}_1, \ldots, \tilde{a}_\kappa, \tilde{b}_1, \ldots, \tilde{b}_k\} \leftrightarrow \{z_1, \ldots, z_{2\kappa}\}$.

Note that to prove these facts we have to consider only one experiment; namely, choose $G_{n,p}$ or $G_{n,m}$ at random and then $z_1, \ldots, z_{2\kappa}$ at random. From this we can deduce that almost every $G_{n,p}$ or $G_{n,m}$ is such that for *all* choices of $a_1, \ldots, b_\kappa$ and almost every choice of $z_1, \ldots, z_{2\kappa}$, we can find edge-disjoint paths $a_i - \tilde{a}_i - \hat{a}_i - \hat{b}_i - \tilde{b}_i - b_i$ for $1 \le i \le \kappa$.

4. Description of the algorithm. The input to our algorithm is a random graph $G_{n,p}$ and a set of pairs of vertices $\{(a_i, b_i) \mid i = 1, \ldots, \kappa\}$ satisfying the premises of Theorem 1. The output is a set of κ edge-disjoint paths, $P_1, \ldots, P_\kappa$ such that P_i connects a_i to b_i.

4.1. Phase 1. We start by partitioning G into five edge-disjoint graphs $G_i = (V_i, E_i)$ for $1 \le i \le 5$. Phase 2 will use only G_1; Phase 3 will use only G_2; Phase 4 will use only G_3; Phase 5 will use only G_4 and G_5. The partition is such that $V_1 = V$ but $V_2 = V_3 = V_4 = V_5 \subseteq V$ with $|V_1| = n - o(n)$.

In this construction, we use the notion of a k-core. The k-core of a graph H is the largest $S \subseteq V(H)$ which induces a subgraph of minimum degree at least k. It is unique and can be found by repeatedly removing vertices of degree less than k until what remains is empty or has minimum degree k.

The algorithm Split depicted in Figure 1 starts by constructing preliminary versions of these graphs, denoted G_i' for $1 \le i \le 5$. Then edges and vertices are deleted from $G_2', \ldots, G_5'$ in order to achieve certain minimum degree properties.

We will show later (Lemma 2) that whp this algorithm terminates with $|K| = n - o(n)$. Note that Split ensures that the following hold.

(i) The final graphs G_i, $2 \le i \le 5$ have the same vertex set K.

[1] In this paper, an event $\mathcal{E}_n$ is said to occur whp (with high probability) if $\mathbf{Pr}(\mathcal{E}_n) = 1 - o(n^{-9/10})$ as $n \longrightarrow \infty$. For reasons explained in section 7, the usual $1 - o(1)$ does not suffice here.

```
1.  algorithm SPLIT
2.  begin
3.      Divide E into E'_i, 1 ≤ i ≤ 5 by placing each edge of E
            independently with probability 5/6 in E'_1, and with
            probability 1/24 into each of E'_i for 2 ≤ i ≤ 5.
4.      For 1 ≤ i ≤ 5 set G'_i ← (V, E'_i)
5.      K ← ⌊d/2⌋-core of G'_1
6.      For 2 ≤ i ≤ 5 set G_i ← (K, E'_i ∩ (K × K))
7.      while ∃v ∈ K such that min{d_{G_i}(v) : 2 ≤ i ≤ 5} < d/30 do
8.          For 2 ≤ i ≤ 5 remove v and its adjacent edges from
                G_i.
9.          K ← K \ {v}
10.     od
11.     For 2 ≤ i ≤ 5 set V_i ← V(G_i) and set E_i ← E(G_i)
12.     G_1 ← (V, E \ (E_2 ∪ E_3 ∪ E_4 ∪ E_5))
13. end SPLIT
```

FIG. 1. *Algorithm* SPLIT.

(ii) Every $v \in K$ has degree at least $\lfloor d/2 \rfloor$ in G_1 and at least $\lfloor d/30 \rfloor$ in each of G_i, $2 \le i \le 5$.

(iii) If $v \in V \setminus K$ then $d_{G_1}(v) = d_G(v)$.

4.2. Phase 2. Choose $z_1, z_2, \ldots, z_{2\kappa} \in V_2$ uniformly and randomly with replacement. Let Z denote the multiset $\{z_1, z_2, \ldots, z_{2\kappa}\}$. We are going to replace the problem of finding paths from a_i to b_i by that of finding paths from $\tilde{a}_i$ to $\tilde{b}_i$, where $\{\tilde{a}_1, \tilde{b}_1, \tilde{a}_2, \tilde{b}_2, \ldots, \tilde{a}_\kappa, \tilde{b}_\kappa\} = Z$ as multisets. Let A denote the multiset $\{a_1, b_1, a_2, b_2, \ldots, a_\kappa, b_\kappa\}$.

We connect A to Z via edge-disjoint paths in the graph G_1 using network flow techniques. We construct a network as follows.

(i) Each undirected edge of G_1 gets capacity 1.

(ii) Each member of A becomes a source and each member of Z becomes a sink.

(iii) If a vertex occurs r times in A then it becomes a source with supply r, and if a vertex occurs s times in Z, then it becomes a sink with demand s.

Then we find a flow from A to Z that satisfies all demands. Since the maximum flow has integer values, it decomposes naturally into $|A|$ edge-disjoint paths (together perhaps with some cycles). If a path joins a_i to $z \in Z$, then we let $\tilde{a}_i = z$. Similarly, if a path joins b_i to $z \in Z$, then we let $\tilde{b}_i = z$.

Thus Phase 2 finds edge-disjoint paths $P_i^{(1)}$ from a_i to $\tilde{a}_i$ and $P_i^{(5)}$ from $\tilde{b}_i$ to b_i, $1 \le i \le \kappa$, where the vertices $\tilde{a}_1, \tilde{b}_1, \tilde{a}_2, \tilde{b}_2, \ldots, \tilde{a}_\kappa, \tilde{b}_\kappa \in V_2$ are chosen uniformly at random with replacement. (Some of these paths may of course be single vertices.) On the other hand there may be some difficult conditioning involved in the pairing of $\tilde{a}_i$ with $\tilde{b}_i$, $1 \le i \le \kappa$. We deal with this in Phase 3.

4.3. Phase 3. We construct paths $P_i^{(2)}$, $P_i^{(4)}$ in G_2 with start vertices $\tilde{a}_i, \tilde{b}_i$, respectively, for $1 \le i \le \kappa$. Each path is constructed by simulating a random walk

of length $\tau = \lceil C_0 \ln n / \ln d \rceil$ from each start point. The endpoints of $P_i^{(2)}$, $P_i^{(4)}$ are $\hat{a}_i, \hat{b}_i$, respectively. The edges of a walk are deleted from G_2 before the next one starts. This keeps the paths edge disjoint. We construct these walks with start points Z in the *random* order $z_1, z_2, \ldots, z_{2\kappa}$. (This random order is helpful in the proof of (21) below.) W_i denotes the walk started at z_i; it ends at $\hat{z}_i$. Γ_i denotes the state of G_2 after the edges of $W_1, W_2, \ldots, W_{i-1}$ have been deleted.

A *random walk* on an undirected graph (or multigraph) $G = (V, E)$ is a Markov chain $\{X_t\}$ on V associated with a particle that moves from vertex to vertex according to the following rule: the probability of a transition from vertex v of degree d_v to a vertex w is $1/d_v$ if $\{v, w\} \in E$ and 0 otherwise. (For multigraphs, each edge out of a vertex is an equally likely exit; loops are counted as two exits.) Its stationary distribution, denoted by π or $\pi(G)$, is given by $\pi_v = d_v/(2|E|)$. A trajectory W of length τ is a sequence of vertices $[w_0, w_1, \ldots, w_\tau]$ such that $\{w_t, w_{t+1}\} \in E$ for $1 \le t < \tau$. The Markov chain induces a probability distribution on trajectories in the usual way. We use $P_G^{(\tau)}(a, b)$ to denote the probability that a random walk in G of length τ starting at a terminates at b.

4.4. Phase 4. The problem now is to find edge-disjoint paths $P_i^{(3)}$ joining $\hat{a}_i$ to $\hat{b}_i$ for $1 \le i \le \kappa$. We use only the edges of G_3 to avoid conflict with paths already chosen in $G_1 \cup G_2$. Thus eventually we can take P_i to be the path (after removing cycles if necessary) that joins a_i to $\tilde{a}_i$ via $P_i^{(1)}$, $\tilde{a}_i$ to $\hat{a}_i$ via $P_i^{(2)}$, $\hat{a}_i$ to $\hat{b}_i$ via $P_i^{(3)}$, $\hat{b}_i$ to $\tilde{b}_i$ via $P_i^{(4)}$, and $\tilde{b}_i$ to b_i via $P_i^{(5)}$. (Actually, this will only be true for *most* i. If $d = O(\ln n)$ then a fifth phase may be necessary to find paths for some indices i.)

The paths $P_i^{(3)}$ are again found by simulating a random walk. The reader might expect us to choose a random walk from those with endpoints $\hat{a}_i, \hat{b}_i$. The main problem with this is that the distribution of $\hat{b}_i$ may be significantly different from the steady state distribution of a walk from $\hat{a}_i$ in G_3. If we choose a walk in this manner then deleting it will condition the graph in a way that is complex to analyze, especially as we have to repeat the procedure κ times.

We overcome this by choosing a set of random walks and use rejection sampling to make the final walk have the correct distribution. There is still the complication that the $\hat{b}_i$ are chosen before we do the walks. This leads to the subroutine WALK described next. See Figure 2. WALK$(\hat{a}_i, \hat{b}_i, \hat{\Gamma}_i, \Gamma_j, z_j)$ generates a series of random walks of length τ in $\hat{\Gamma}_i$ starting from $\hat{a}_i$. The graph $\hat{\Gamma}_i$ is such that $\hat{\Gamma}_i \subseteq G_3$ with $V(\hat{\Gamma}_i) = V(G_3) = V(\Gamma_j)$, and j is defined by $z_j \equiv \tilde{b}_i$. The last walk generated ends at $\hat{b}_i$ which, by the construction used in the previous phase, has the distribution

$$\hat{p}_v = P_{\Gamma_j}^{(\tau)}(z_j, v).$$

The somewhat strange method used to generate these walks will be further explained in section 7.

We maintain an array of counters, $S[v]$, for $v \in V_3$, initially all 0. The counter $S[v]$ shows how many times v was used as a start point of a walk. No vertex is allowed to be the start of more than $d/120$ walks; thus there is a chance (in fact, only when $d = O(\ln n)$) that for some pairs of vertices Phase 4 will not connect them. The indices of these pairs are kept in a set L and considered in the last phase.

The distributions p_v and $\hat{p}_v$ can be computed in $O(nm\tau)$ time by computing powers of the transition matrix, after which a random walk can be found in $O(n\tau)$ time. (For details see [8].) The analysis will show that in the range of interest whp s

subroutine WALK($\hat{a}_i, \hat{b}_i, \hat{\Gamma}_i, \Gamma_j, z_j$)
begin
/* By construction, $z_j = \tilde{b}_i$. */
$p_v \leftarrow P^{(\tau)}_{\hat{\Gamma}_i}(\hat{a}_i, v)$ for $v \in V_3$
$\hat{p}_v \leftarrow P^{(\tau)}_{\Gamma_j}(z_j, v)$ for $v \in V_2$ (the distribution of $\hat{b}_i$)
$p_{\min} \leftarrow \min\{p_v : v \in V_3\}$
$\hat{p}_{\max} \leftarrow \max\{\hat{p}_v : v \in V_2\}$
Choose r from the geometric distribution with probability of success $s = p_{\min}/\hat{p}_{\max}$
if $S[\hat{a}_i] + r \geq d/120$ **then**
    $L \leftarrow L \cup \{i\}$
    exit WALK
else
    $S[\hat{a}_i] \leftarrow S[\hat{a}_i] + r$
fi
for k **from** 1 **to** $r - 1$ **do**
    Choose x_k according to $\mathbf{Pr}(x_k = v) = (p_v - \hat{p}_v p_{\min}/\hat{p}_{\max})/(1 - s)$
od
$x_r \leftarrow \hat{b}_i$
for k **from** 1 **to** r **do**
    Pick a walk $\hat{W}_k$ of length τ in $\hat{\Gamma}_i$ according to the distribution on trajectories, conditioned on start point $= \hat{a}_i$ and endpoint $= x_k$
od
output $\hat{W}_1, \hat{W}_2, \ldots, \hat{W}_r$
end WALK

FIG. 2. *Algorithm* WALK.

is bounded away from zero by a constant; hence the expected total running time of WALK is $O(nm\tau)$.

The complete algorithm for Phase 4, GENPATHS, is depicted in Figure 3. The expected running time of GENPATHS is $O(\kappa nm\tau) = O(nm^2)$.

4.5. Phase 5. Use (a slight modification of) the algorithm of [8] to find edge-disjoint paths in $G_4 \cup G_5$ from $\hat{a}_i$ to $\hat{b}_i$ for $i \in L$.

5. Analysis of Phase 1. In Lemmas 2, 3, and 4 we calculate with $G_{n,p}$ and deduce the result for $G_{n,m}$ via

$$\mathbf{Pr}(G_{n,m} \in \mathcal{P}) \leq O(n^{1/2})\, \mathbf{Pr}(G_{n,p} \in \mathcal{P})$$

for any graph property $\mathcal{P}$, assuming $m = \binom{n}{2} p \geq n$.

```
1.  algorithm GenPaths
2.  begin
3.      Let E(W) denote the edge set of a walk W.
4.      Γ̂_1 ← G_3
5.      for i = 1 to κ do
6.          Define j such that z_j ≡ b̃_i.
7.          Execute Walk(â_i, b̂_i, Γ̂_i, Γ_j, z_j)
8.          if i ∉ L then
9.              P_i^(3) ← Ŵ_r
10.             Γ̂_{i+1} ← Γ̂_i \ ( ∪_{j=1}^{r} E(Ŵ_j) )
11.         fi
12.     od
13. end GenPaths
```

FIG. 3. *Algorithm* GenPaths.

Our immediate task regarding this phase is to prove Lemma 1.

LEMMA 2. *With high probability, the vertex set* $K = V_i$, $2 \le i \le 5$ *satisfies*

$$|K| = n - o(n).$$

Proof. Let K_0 denote the value of K immediately prior to the execution of the while loop of Split; that is, K_0 is the $\lfloor d/2 \rfloor$ core of G_1'. The final K is the largest subset S of K_0 for which $\delta(G_i'[S]) \ge d/30$, $2 \le i \le 5$.

Let $A_1 = \{v \in V : d_{G_1'}(v) \le 2d/3\}$ and $A_i = \{v \in V_i : d_{G_i'}(v) \le d/27,\ 2 \le i \le 5\}$. Let $A = \bigcup_{i=1}^{5} A_i$. We show that whp

(i) $|A| = o(n)$,

(ii) $v \in V \setminus A$ implies that v has at most 50,000 G-neighbors in A.

It follows from (ii) and the definition of A that for d sufficiently large $K \supseteq V \setminus A$ and then (i) implies the lemma.

We start from the fact that for any $k \ge 1$,

$$\mathbf{Pr}(|A_1| \ge k) \le \binom{n}{k} \mathbf{Pr}(\mathrm{Bin}(n-k, 5p/6) \le 2d/3)^k.$$

Putting $k = k_1 = \lceil n^{61/62} \rceil$ and using (1) with $\epsilon = 1/5$, we obtain

$$\mathbf{Pr}(|A_1| \ge k_1) \le \left(\frac{ne}{k_1}\right)^{k_1} e^{-k_1 d/61} = \left(\frac{ne^{1-d/61}}{k_1}\right)^{k_1} = o(n^{-2}). \tag{3}$$

Also, for any fixed $k \ge 300$,

$$\begin{aligned} &\mathbf{Pr}(\exists v \in V : |N(v, G_{n,p}) \cap A_1| \ge k) \\ &\qquad \le n\binom{n}{k} p^k \mathbf{Pr}(\mathrm{Bin}(n-k-1, 5p/6) \le 2d/3)^k \\ &\qquad \le n d^k e^{-kd/61} = o(n^{-2}). \end{aligned} \tag{4}$$

Similarly, for any $k \geq 1$ and $i \geq 2$,

$$\mathbf{Pr}(|A_i| \geq k) \leq \binom{n}{k} \mathbf{Pr}(\mathrm{Bin}(n-k, p/24) \leq d/27)^k.$$

Now putting $k = k_2 = \lceil n^{3999/4000} \rceil$ and using (1) with $\epsilon = 1/9$, we obtain

$$\mathbf{Pr}(|A_i| \geq k_2) \leq \left(\frac{ne}{k_2}\right)^{k_2} e^{-k_2 d/3900} = \left(\frac{ne^{1-d/3900}}{k_2}\right)^{k_2} = o(n^{-2}), \tag{5}$$

and for any fixed $k \geq 12000$,

$$\begin{aligned} &\mathbf{Pr}(\exists v \in V : |N(v, G_{n,p}) \cap A_i| \geq k) \\ &\qquad \leq n\binom{n}{k} p^k \, \mathbf{Pr}(\mathrm{Bin}(n-k-1, p/24) \leq d/27)^k \\ &\qquad \leq n d^k e^{-kd/3900} = o(n^{-2}). \end{aligned} \tag{6}$$

From (3) and (5) we conclude that whp $A = o(n)$, and from (4) and (6) we conclude that whp no vertex in G has more than 50,000 neighbors in A. □

6. Analysis of Phase 2. In this section we show that if our input graph $G = (V, E)$ is $G_{n,p}$, then whp, after we run SPLIT, we can find in G_1 edge-disjoint paths from a_i to $\tilde{a}_i$ and b_i to $\tilde{b}_i$ for $1 \leq i \leq \kappa$ for *any* choice for $a_1, \ldots, b_\kappa$ consistent with the premises of Theorem 1 and for *almost every* choice for $\tilde{a}_1, \ldots, \tilde{b}_\kappa$.

Let A and Z be as defined in section 4.2. For $v \in V$, let $\alpha(v)$ be the multiplicity of $v \in A$ and $\xi(v)$ be the multiplicity of $v \in Z$. For $S \subseteq V$, let $\alpha(S) = \sum_{v \in S} \alpha(v)$ and $\xi(S) = \sum_{v \in S} \xi(v)$. For sets $S, T \subseteq V$, let $e_{G_1}(S, T)$ denote the number of edges of G_1 with an endpoint in S and the other endpoint in T. It suffices to prove that

$$e_{G_1}(S, \bar{S}) \geq \alpha(S) - \xi(S) \qquad \forall S \subseteq V. \tag{7}$$

We can then apply a theorem of Gale [11] (or see Bondy and Murty [6, Theorem 11.8]) to deduce the existence of the required flow in G_1 for the successful run of Phase 2. (We must of course demonstrate (7) for all A satisfying the premises of Theorem 1 and almost all Z.)

We next prove three lemmas instrumental in proving Lemma 6 below.

LEMMA 3. *With high probability, for any $v \in V_2$,*

$$\xi(v) \leq \beta d_{G_1}(v). \tag{8}$$

Proof. Observe that $\xi(v)$ has the distribution $\mathrm{Bin}(2\kappa, |V_2|^{-1})$. Thus

$$\begin{aligned} \mathbf{Pr}(\xi(v) > \beta d_{G_1}(v)) &\leq \binom{2\kappa}{\beta d/2} |V_2|^{-\beta d/2} \leq \left(\frac{4e\kappa}{\beta d} \cdot \frac{1+o(1)}{n}\right)^{\beta d/2} \\ &\leq \left(\frac{12\alpha \ln d}{\beta \ln n}\right)^{\beta d/2} = o(n^{-3}), \end{aligned}$$

provided that

$$\alpha \leq \beta e^{-6/\beta}/12. \tag{9}$$ □

LEMMA 4. (a) *G_1 has the following property whp: if $S \subseteq V$ and $n_0 = ne^{-d/10} \le |S| \le n/2$ then $e_{G_1}(S, \bar{S}) \ge d|S|/5$.*

(b) *$G_{n,p}$ has the following property whp: if $S \subseteq V$ and $|S| \le n_0$ then $e_G(S,S) \le 2|S|$.*

Proof. (a) Note that G_1' is distributed as $G_{n,5p/6}$ and $G_1' \subseteq G_1$. But

$$\begin{aligned}
&\mathbf{Pr}(G_{n,5p/6} \text{ does not satisfy property (a)}) \\
&\quad \le \sum_{k=n_0}^{n/2} \binom{n}{k} \mathbf{Pr}(\mathrm{Bin}(k(n-k), 5p/6) \le kd/5) \\
&\quad \le \sum_{k=n_0}^{n/2} \binom{n}{k} \mathbf{Pr}\left(\mathrm{Bin}(k(n-k), 5p/6) \le \frac{1}{2}k(n-k)\frac{5}{6}p\right) \\
&\quad \le \sum_{k=n_0}^{n/2} \left(\frac{ne}{k}\right)^k \exp\left(-\frac{5}{48}k(n-k)p\right) = o(n^{-2}).
\end{aligned}$$

(b) Note that property (b) holds trivially for $|S| \le 5$ or $d \ge 10 \ln n$, which implies $n_0 \le 1$. Assume $d \le 10 \ln n$ and $|S| \ge 6$. Thus

$$\begin{aligned}
&\mathbf{Pr}(G_{n,p} \text{ does not satisfy property (b)}) \\
&\quad \le \sum_{k=6}^{n_0} \binom{n}{k} \binom{k(k-1)/2}{2k} p^{2k} \le \sum_{k=6}^{n_0} \left(\frac{ne}{k}\right)^k \left(\frac{k^2 ep}{2k}\right)^{2k} = o(n^{-2}). \qquad \square
\end{aligned}$$

LEMMA 5. *Let $I = \{v \in V : d_{G_1}(v) \le 2\beta d\}$. Then whp no two (distinct) vertices in I are within distance of two or less in G_1.*

Proof. Observe first that $I = \emptyset$ if $d \ge C \ln n$ for C sufficiently large. We can thus assume that $d = O(\log n)$ for the rest of the proof of this lemma. If $v \in I$ then either $d_G(v) \le 2\beta d$ or $d_{G_1}(v) \ne d_G(v)$. The latter cannot be true, since it implies that $v \in V_2$, and then $d_{G_1}(v) \ge \lfloor d/2 \rfloor$. Thus

$$\mathbf{Pr}(I \text{ contains an edge}) \le n^2 p \left(\sum_{k=0}^{2\beta d} \binom{n-2}{k} p^k (1-p)^{n-k}\right)^2.$$

But

$$\begin{aligned}
&\sum_{k=0}^{2\beta d} \binom{n-2}{k} p^k (1-p)^{n-k} \\
&\qquad = O\left(\binom{n}{2\beta d} p^{2\beta d}(1-p)^n\right) = O\left(\left(\frac{e}{2\beta}\right)^{2\beta d} e^{-d}\right) = O(n^{-.99}),
\end{aligned}$$

provided that

$$2\beta(1 - \ln 2\beta) \le 1/100. \tag{10}$$

Thus

$$\mathbf{Pr}(I \text{ contains an edge}) = o(n^{-9/10}).$$

A similar calculation deals with the case of a path of length two joining two vertices of I.

The rather tedious calculation for $G_{n,m}$ is left to the interested reader—see [5] for details of a similar calculation. □

Now inequality (7) will follow easily from Lemma 5.

LEMMA 6. *Define for every* $v \in V$

$$\theta(v) = \min\{d_{G_1}(v), \beta d\}.$$

Then whp for every $S \subseteq V$ *satisfying* $1 \le |S| \le n/2$,

$$e_{G_1}(S, \bar{S}) \ge \theta(S), \tag{11}$$

where $\bar{S} = V \setminus S$ *and* $\theta(S) = \sum_{v \in S} \theta(v)$.

Proof. Since $\beta < 1/5$, Lemma 4(a) implies that for $S \subseteq V$ satisfying $n_0 \le |S| \le n/2$,

$$e_{G_1}(S, \bar{S}) \ge d|S|/5 \ge \theta(S).$$

Suppose next that $|S| \le n_0$. Let

$$I_1 = \{v \in V : d_{G_1}(v) \le 2\beta d\},$$
$$I_2 = \{v \in V : 2\beta d < d_{G_1}(v)\}.$$

Let $S_i = S \cap I_i$ for $i = 1, 2$. Then

$$e_{G_1}(S, \bar{S}) = e_{G_1}(S_1, \bar{S}_1) + e_{G_1}(S_2, \bar{S}_2) - 2e_{G_1}(S_2, S_1).$$

But by Lemma 5 $G_1[S_1]$ has no edges, so that

$$e_{G_1}(S_1, \bar{S}_1) \ge \theta(S_1) \qquad \text{whp},$$

and using Lemma 4(b),

$$e_{G_1}(S_2, \bar{S}_2) \ge (2\beta d - 4)|S_2| \qquad \text{whp},$$

and since Lemma 5 implies that whp no vertex in S_2 is adjacent to two or more vertices in S_1, we have also

$$e_{G_1}(S_2, S_1) \le |S_2| \qquad \text{whp}.$$

It thus follows that whp

$$e_{G_1}(S, \bar{S}) \ge \theta(S_1) + (2\beta d - 6)\,|S_2| \ge \theta(S),$$

where the last inequality holds for sufficiently large n so that $\beta d > 6$. This shows (11). □

We now show that Lemma 5 implies equation (7). First note that condition (ii) in Theorem 1 implies $\alpha(v) \le \theta(v)$ for all $v \in V$.

Second observe that SPLIT guarantees that for $v \in Z$, $\theta(v) = \beta d$, assuming that $\beta < 1/2$, since $Z \subseteq V_2$ and every $v \in V_2$ has degree at least $\lfloor d/2 \rfloor$ in G_1. Thus

$$\mathbf{Pr}(\exists v \in V \text{ such that } \xi(v) > \theta(v) \mid |V_2|)$$
$$\le |V_2| 2 \binom{\kappa}{\beta d} |V_2|^{-\beta d} \le 2n \left(\frac{(1 + o(1))e\alpha m \ln d}{n\beta d \ln n} \right)^{\beta d} = o(n^{-2}),$$

provided that α is sufficiently small. We can thus assume that whp $\xi(v) \le \theta(v)$ for all $v \in V$.

To complete the proof of equation (7), note first that for $|S| \le n/2$, by Lemma 6,

$$e_{G_1}(S, \bar{S}) \ge \theta(S) \ge \alpha(S) \ge \alpha(S) - \xi(S);$$

and for $|S| \ge n/2$,

$$e_{G_1}(S, \bar{S}) = e_{G_1}(\bar{S}, S) \ge \theta(\bar{S}) \ge \xi(\bar{S}) \ge \xi(\bar{S}) - \alpha(\bar{S}) = \alpha(S) - \xi(S).$$

7. Analysis of Phase 3. If a vertex $v \in V_2$ has degree $d_v^{(i)}$ in Γ_i, then the steady state probability of a random walk in Γ_i being at v is

$$\pi_v^{(i)} = \frac{d_v^{(i)}}{\sum_{w \in V_2} d_w^{(i)}}.$$

The main thrust of our analysis is to show that the joint distribution of the $\hat{z}_i$ is close to that of independent samples from $\pi^{(i)}$ for $1 \le i \le 2\kappa$; that is, whp for $v \in V_2$ and $1 \le i \le 2\kappa$,

$$\mathbf{Pr}(\hat{z}_i = v \mid \Gamma_i,\ \hat{z}_j,\ j \ne i) = (1 + o(1))\pi_v^{(i)}. \tag{12}$$

In this case, when we come to join $\hat{a}_i$ to $\hat{b}_i$ then we can argue that $\hat{b}_i$ is (essentially) independent of $\hat{a}_i$. It is difficult to argue this for $\tilde{a}_i, \tilde{b}_i$ since they have been "chosen" as pairs by a flow algorithm. This is why we need Phase 3.

Let $\mathcal{E}_0$ denote the intersection of the events previously shown to hold whp. Let $P^{(i)}$ denote the transition probability matrix of a random walk on Γ_i. Let $\lambda^{(i)}$ be the second largest eigenvalue of $P^{(i)}$. We will prove Theorem 2 later.

THEOREM 7. *For $1 \le i \le \kappa$ let $\mathcal{E}_i$ be the event that*

(a) *the maximum degree $\Delta^{(i)}$ in Γ_i satisfies*

$$\Delta^{(i)} \le C_1 d; \tag{13}$$

and

(b) *the minimum degree $\delta^{(i)}$ in Γ_i satisfies*

$$\delta^{(i)} \ge d/C_2. \tag{14}$$

If $d \le n^{1/10}$ then there exists a constant $\gamma = \gamma(C_1, C_2) > 0$ such that if $\mathcal{F}_i$ denotes the event that

$$\lambda^{(i)} \le \gamma/\sqrt{d} \tag{15}$$

and $\mathcal{U}_i = \mathcal{F}_i \cap \mathcal{E}_i \cap \cdots \cap \mathcal{F}_1 \cap \mathcal{E}_1 \cap \mathcal{E}_0$, then

$$\mathbf{Pr}(\mathcal{F}_i \mid \mathcal{E}_i, \mathcal{U}_{i-1}) = \mathbf{Pr}(\mathcal{F}_i \mid \mathcal{E}_i) = 1 - O(n^{-3}). \tag{16}$$

Proof. See section 10. □

The reader will notice the bound $d \le n^{1/10}$ in the theorem above. If $d > n^{1/10}$ we can randomly split the edge set of G into $r = \lceil 2d/n^{1/10} \rceil$ subsets $E_1, E_2, \ldots, E_r$, each of size roughly $m' = m/r$. We can similarly split the set of κ pairs into r roughly

equal sets K_i. We can then use the graph $G_i = (V, E_i)$ to find paths for the pairs in K_i. Every vertex of every G_i will have degree roughly d/r whp. Hence since

$$\frac{\kappa}{r} \le \alpha m' \frac{\ln n}{\ln d},$$

we can apply Theorem 1 to each G_i, which implies that we succeed whp on each K_i, and thus we will succeed overall[2] with probability $1 - o(1)$. Therefore, without loss of generality, we can assume from now on that $d \le n^{1/10}$.

We now return to the analysis of Phase 3. We start by assuming that

$$\mathcal{E}_i \text{ and } \mathcal{F}_i \text{ hold for every } i. \tag{17}$$

It is well known that the second eigenvalue determines the rate of convergence of a Markov chain to its steady state. An explicit form of this result was obtained by Sinclair and Jerrum [18]: if $P^{(t)}_{\Gamma_i}(u, v)$ denotes the probability that a random walk of length t in Γ_i that starts at u will end at v, then assuming $\mathcal{E}_i$ we have

$$|P^{(t)}_{\Gamma_i}(u,v) - \pi^{(i)}_v| \le \left(\lambda^{(i)}\right)^t \sqrt{\frac{\pi^{(i)}_v}{\pi^{(i)}_u}} \le \left(\lambda^{(i)}\right)^t \sqrt{C_1 C_2} \le \frac{\gamma(C_1, C_2)^t}{d^{t/2}} \sqrt{C_1 C_2}. \tag{18}$$

Since in the algorithm we take $t = \tau = \lceil C_0 \ln n / \ln d \rceil$, this implies (12).

We now proceed to show that the assumption (17) is indeed correct whp. We take $C_1 = 5$ and $C_2 = 60$. Since Γ_i is a subgraph of $G_{n,p}$, inequality (13) holds for all i whp, and since $\Gamma_1 = G_2$ and by construction $\delta(G_2) \ge d/30$, inequality (14) holds for Γ_1; thus $\mathcal{E}_1$ holds. Applying Theorem 7, we see that $\mathcal{F}_1$ holds whp. We continue by showing inductively that for $i \ge 0$,

$$\mathbf{Pr}(\mathcal{U}_i \mid \mathcal{E}_0) = 1 - O(in^{-3}).$$

Since

$$\mathbf{Pr}(\mathcal{U}_{i+1} \mid \mathcal{U}_i) = \frac{\mathbf{Pr}(\mathcal{U}_{i+1})}{\mathbf{Pr}(\mathcal{U}_i)} = \mathbf{Pr}(\mathcal{F}_{i+1} \mid \mathcal{E}_{i+1}, \mathcal{U}_i)\, \mathbf{Pr}(\mathcal{E}_{i+1} \mid \mathcal{U}_i),$$

and

$$\mathbf{Pr}(\mathcal{U}_{i+1} \mid \mathcal{E}_0) = \mathbf{Pr}(\mathcal{U}_{i+1} \mid \mathcal{U}_i)\, \mathbf{Pr}(\mathcal{U}_i \mid \mathcal{E}_0),$$

and given Theorem 7, we only need to prove that

$$\mathbf{Pr}(\mathcal{E}_{i+1} \mid \mathcal{U}_i) = 1 - O(n^{-3}), \tag{19}$$

which reduces to proving that given $\mathcal{U}_i$, the removal of the walks $W_1, \ldots, W_i$ from G_2 does not reduce the degree of any vertex to less than $d/60$.

Now assume $\mathcal{U}_i$. Consider the walk W_i on Γ_i. For $v \in V_2$, let $Z_{i,v}$ denote the number of edges incident with v that are covered by W_i, and let $N_{i,v}$ be the number of visits to v during W_i. Let $q_k = \mathbf{Pr}(N_{i,v} = k \mid \mathcal{U}_i)$ for $k \ge 1$. We claim that independently of $W_1, W_2, \ldots, W_{i-1}$, there exist constants C_3 and C_4 so that

$$q_k \le \frac{C_4 C_3^{k-1} \ln n}{d^{k-1} n \ln d}. \tag{20}$$

[2]This is the reason for our definition of whp. The number r of subgraphs G_i is $O(n^{9/10})$ and we succeed with probability $1 - o(n^{-9/10})$ on each.

To prove (20) for $k = 1$, fix Γ_i, and let $h_v(t)$ be the probability that the walk is at v at time t. Then

$$h_v(0) = 1/|V_2| \le C_1 C_2 \pi_v^{(i)} \tag{21}$$

since the walk starts from z_i which is a vertex chosen uniformly at random in $|V_2|$. (The last inequality follows assuming that $\mathcal{U}_i$ occurs, and thus $\mathcal{E}_i$ occurs.)

We next show inductively that for all $v \in V_2$, we have $h_v(t) \le C_1 C_2 \pi_v^{(i)}$. This follows from stationarity equations and

$$h_v(t+1) = \sum_{w \in N(v:\Gamma_i)} \frac{h_w(t)}{d_w^{(i)}} \le C_1 C_2 \pi_v^{(i)}. \tag{22}$$

Hence since $\tau = \lceil C_0 \ln n / \ln d \rceil$ and $\pi_v^{(i)} \le C_1 C_2 / n$, there is a constant C_4 so that

$$q_1 \le \sum_{t=0}^{\tau} h_v(t) \le \frac{C_4 \ln n}{n \ln d}.$$

We next prove (20) for $k \ge 2$. Fix Γ_i and for vertex v let ρ_v be the probability that a random walk of length τ from v ever returns to v. Since a return to v requires at least two steps, we obtain from equation (18) that there exists a constant C_3 such that

$$\rho_v \le \tau \pi_v^{(i)} + \sqrt{C_1 C_2} \sum_{t \ge 2} \frac{\gamma(C_1, C_2)^t}{d^{t/2}} \le \frac{C_3}{d}. \tag{23}$$

This gives (20) since

$$q_k \le (\rho_v)^{k-1} \sum_{t=1}^{\tau} h_v(t).$$

We now show that (20) implies (19). First (20) implies that for any constant c,

$$\mathbf{E}(e^{2cN_{i,v}} \mid \mathcal{U}_i, W_1, \ldots, W_{i-1}) \le 1 + \sum_{k \ge 1} e^{2ck} \frac{C_4 C_3^{k-1} \ln n}{d^{k-1} n \ln d} \le 1 + \frac{2C_4 e^{2c} \ln n}{n \ln d}.$$

Clearly $Z_{i,v} \le 2N_{i,v}$. Thus for any constant $c > 0$ and any $t > 0$,

$$\begin{aligned}
&\mathbf{Pr}\left(\sum_{j=1}^{i} Z_{j,v} \ge t \;\middle|\; \mathcal{U}_i \right) \\
&\quad \le e^{-ct}\, \mathbf{E}\left(\exp\left(2c \sum_{j=1}^{i} N_{j,v} \right) \;\middle|\; \mathcal{U}_i \right) \\
&\quad \le e^{-ct} \left(1 + \frac{2C_4 e^{2c} \ln n}{n \ln d} \right) \mathbf{E}\left(\exp\left(2c \sum_{j=1}^{i-1} N_{j,v} \right) \;\middle|\; \mathcal{U}_i \right) \\
&\quad \le e^{-ct} \exp\left(\frac{2C_4 e^{2c} \ln n}{n \ln d} \right) \mathbf{E}\left(\exp\left(2c \sum_{j=1}^{i-1} N_{j,v} \right) \;\middle|\; \mathcal{U}_{i-1} \right) \frac{\mathbf{Pr}(\mathcal{U}_{i-1})}{\mathbf{Pr}(\mathcal{U}_i)}
\end{aligned}$$

$$\le \exp\left(-ct + i\frac{2C_4e^{2c}\ln n}{n\ln d}\right)\frac{1}{\mathbf{Pr}(\mathcal{U}_i)}$$
$$\le \exp\left(-ct + 2\kappa\frac{2C_4e^{2c}\ln n}{n\ln d}\right)(1+O(in^{-3}))$$
$$\le 2\exp(-ct + 4\alpha C_4e^{2c}d).$$

Taking $t = d/60$, $c = 240$, and $\alpha \le (4C_4e^{480})^{-1}$, we obtain that

$$\mathbf{Pr}\left(\sum_{j=1}^{i} Z_{j,v} \ge \frac{d}{60} \,\middle|\, \mathcal{U}_i\right) \le 2n^{-3},$$

and since the minimum degree in G_2 is at least $d/30$, this proves (19). (Recall that $C_2 = 60$.)

8. Analysis of Phase 4. We start by discussing the subroutine WALK. Consider a modification of WALK depicted in Figure 4.

```
1.   subroutine WALK1(â_i, Γ̂_i, Γ_j, z_j)
2.   begin
3.   /* By construction, z_j = b̃_i. */
4.   p_v ← P^(τ)_{Γ̂_i}(â_i, v) for v ∈ V(Γ̂_i)
5.   p̂_v ← P^(τ)_{Γ_j}(z_j, v) for v ∈ V(Γ̂_i) (the distribution of b̂_i)
6.   p_min ← min{p_v : v ∈ V(Γ̂_i)}
7.   p̂_max ← max{p̂_v : v ∈ V(Γ̂_i)}
8.   r̄ ← 0
9.   forever do
10.      r̄ ← r̄ + 1
11.      S[â_i] ← S[â_i] + 1
12.      if S[â_i] ≥ d/120 then
13.          L ← L ∪ {i}
14.          exit WALK1
15.      fi
16.      Pick a walk W̄_r̄ of length τ according to the distribution on
             trajectories, conditioned on start point = â_i
17.      Let x̄_r̄ be the terminal vertex of W̄_r̄
18.      With probability p̂_{x̄_r̄} p_min/(p_{x̄_r̄} p̂_max) accept W̄_r̄ and
             exitloop
19.  od
20.  output W̄_1, W̄_2, ..., W̄_r̄
21.  end WALK1
```

FIG. 4. *Algorithm* WALK1.

LEMMA 8. *In* WALK1, $\bar{x}_{\bar{r}}$ *is chosen according to the distribution* $\hat{p}$.

Proof. The probability s that a walk is accepted at the last step in the loop is given by

$$s = \sum_{v \in V(G)} p_v \frac{\hat{p}_v p_{\min}}{p_v \hat{p}_{\max}} = \frac{p_{\min}}{\hat{p}_{\max}}. \tag{24}$$

(Observe that $\hat{p}_{\max} \geq 1/|V(G)| \geq p_{\min}$.) Thus if S_0 is the value of $S[\hat{a}_i]$ at the start of WALK1 and $k_0 = d/120 - S_0$, then

$$\mathbf{Pr}(\bar{x}_{\bar{r}} = v \mid \text{step 14 is not executed}) = \frac{1}{1-(1-s)^{k_0}} \sum_{k=0}^{k_0-1} (1-s)^k p_v \frac{\hat{p}_v p_{\min}}{p_v \hat{p}_{\max}} = \hat{p}_v. \tag{25}$$

Also, $\mathbf{Pr}$(step 14 is executed) is equal to $(1-s)^{k_0}$ in both procedures. □

Hence $\bar{W}_{\bar{r}}$ is a random walk to a vertex chosen with distribution $\hat{p}$. Furthermore, since the minimum degree of any graph in which a walk is constructed is at least $d/60$ and the maximum degree is at most $5d$, we find

$$s \geq \frac{1}{300} \stackrel{\text{def}}{=} \sigma \tag{26}$$

and therefore the expected number of generated walks is constant.

There is a minor problem in that we want to choose the endpoints before we do the walks. This leads to the algorithm WALK described before. We now turn to its analysis.

LEMMA 9. *Suppose that* $\hat{b}_i$ *is chosen from* $V(G)$ *with distribution* $\hat{p}$. *Then the set of walks* $\bar{W}_1, \ldots, \bar{W}_{\bar{r}}$ *in* WALK1$(\hat{a}_i, G, \Gamma_j, z_j)$ *and the set of walks* $\hat{W}_1, \ldots, \hat{W}_r$ *in* WALK$(\hat{a}_i, \hat{b}_i, G, \Gamma_j, z_j)$ *have the same distribution.*

Proof. Note first from the proof of Lemma 8 that $\bar{r}$ and r have the same truncated geometric distribution. Also we have from Lemma 8 that $\bar{x}_{\bar{r}}$ and $x_r = \hat{b}_i$ have the same distribution. Consider next that for $v_1, v_2, \ldots, v_i \in V(G)$,

$$\begin{aligned}
&\mathbf{Pr}(\bar{x}_1 = v_1, \ldots, \bar{x}_i = v_i \text{ and } \bar{r} > i) \\
&\qquad = \prod_{j=1}^{i} \left(\left(1 - \frac{\hat{p}_{v_j} p_{\min}}{p_{v_j} \hat{p}_{\max}} \right) p_{v_j} \right) = \prod_{j=1}^{i} \left(p_{v_j} - \frac{\hat{p}_{v_j} p_{\min}}{\hat{p}_{\max}} \right) \\
&\qquad = (1-s)^i \prod_{j=1}^{i} \left(\frac{p_{v_j} - \hat{p}_{v_j} p_{\min}/\hat{p}_{\max}}{1-s} \right) \\
&\qquad = \mathbf{Pr}(x_1 = v_1, \ldots, x_i = v_i \text{ and } r > i).
\end{aligned}$$

Thus $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{\bar{r}}$ and $x_1, x_2, \ldots, x_r$ have the same distribution. Finally, the lemma follows from the fact that the distribution of $\bar{W}_j$ conditional on $\bar{x}_j = v$ is clearly equal to that of $\hat{W}_j$ conditional on $x_j = v$. □

The net effect of GENPATHS is to run WALK κ times. In light of Theorem 7 we only need show that the minimum degree in G_3 is not made too small by the deletion of paths generated by WALK. This requires a slightly more complicated analysis than for Phase 3. The main problem in extending the analysis of Phase 3 is that we cannot argue now that (21) holds independently of previous walks. Each execution of WALK

(or, equivalently, WALK1) involves a set of walks with the same starting point. The initial vertex of each set is chosen nearly randomly, but it is the same for each walk.

For the purpose of the analysis we relate to WALK1. Fix $v \in V_3$ and $1 \le i \le \kappa$ and let $\bar{W}_1, \bar{W}_2, \ldots, \bar{W}_r$ denote the walks made while trying to connect $\hat{a}_i$ to $\hat{b}_i$. We shall refer to these walks as the ith *bundle*, B_i. We shall follow closely the line of proof used in the analysis of Phase 3, with all the events now referring to $\hat{\Gamma}_i$ rather than Γ_i. As before, the proof reduces to showing that given $\mathcal{U}_i$, the removal of the bundles $B_1, \ldots, B_i$ from G_3 does not reduce the degree of any vertex in G_3 to less than $d/60$.

The stationary distribution on $\hat{\Gamma}_i$ is denoted $\hat{\pi}^i$.

LEMMA 10. *Assuming $\mathcal{U}_i$, the probability that a fixed vertex v is visited by bundle i is less than*

$$\frac{C_5 \ln n}{n \ln d}.$$

Proof. Assume for a moment that the probability of a walk being accepted is decreased to exactly σ (see (26)). This can only increase the number of visits to v, but the number of walks is now independent of the start point. For every walk in the bundle we can show via (20) and (21) applied to $\hat{\Gamma}_i$ that the expected number of visits to v is less than $C_1 C_2 \tau \hat{\pi}_v^{(i)}$; thus the expected total number of visits is less than

$$\frac{C_1 C_2}{\sigma} \tau \hat{\pi}_v^{(i)} = \frac{C_5 \ln n}{n \ln d}. \qquad \square$$

LEMMA 11. *Assume $\mathcal{U}_i$ and consider a random walk of length τ in $\hat{\Gamma}_i$ starting from vertex v. Then*

(a) *the probability that the walk returns k times to v is less than $(C_3/d)^k$;*

(b) *for any vertex $u \ne v$, the probability that u is visited k times is less than $(C_3/d)^k$.*

Proof. As before let ρ_v be the probability that a random walk of length τ from v ever returns to v. From (23) applied to $\hat{\Gamma}_i$,

$$\rho_v \le \frac{C_3}{d}.$$

For part (a) notice that the probability of k returns to v is bounded by

$$\sum_{t=2}^{\tau} \left(\hat{\pi}_v^{(i)} + \frac{\gamma (C_1, C_2)^t}{d^{t/2}} \right) \rho_v^{k-1} \le \left(\frac{C_3}{d} \right)^k.$$

For part (b) the probability of k visits to u is bounded by

$$\frac{C_2}{d} \rho_u^{k-1} + \sum_{t=2}^{\tau} \left(\hat{\pi}_u^{(i)} + \frac{\gamma (C_1, C_2)^t}{d^{t/2}} \right) \rho_u^{k-1} \le \left(\frac{C_3}{d} \right)^k.$$

(The first term deals with the case when u is a neighbor of v.) $\square$

We are ready now to evaluate the number of visits to a fixed vertex v. For this goal we will distinguish between *free* visits and *start* visits. If $v = \hat{a}_i$, then v undergoes $|B_i|$ visits as the start point of all the walks in the bundle. All other visits to v are free visits. In particular a return visit to $\hat{a}_i$ is a free visit.

Analogously to the analysis of Phase 3, let $N_{i,v}$ be the number of *free* visits to v during B_i and let $q_k = \mathbf{Pr}(N_{i,v} = k \mid \mathcal{U}_i)$ for $k \geq 1$. We claim that independently of $B_1, B_2, \ldots, B_{i-1}$, there exists a constant C_6 so that

$$q_k \leq \frac{C_5 C_6^{k-1} \ln n}{d^{k-1} n \ln d}. \tag{27}$$

To simplify notation view the r walks in bundle B_i as a single walk X_t that restarts from $\hat{a}_i$ every τ steps. Let $h_v(t)$ be the probability that this walk is at v at time t. Then

$$q_k \leq \sum_{1 \leq t \leq \tau} h_v(t)\, \mathbf{Pr}(k-1 \text{ free visits to } v \text{ after } t \mid \hat{a}_i, X_t = v).$$

Now given r, the number of walks in bundle B_i, the $k-1$ free visits to v can be distributed among the r walks in at most $\binom{k+r-2}{r-1}$ ways. So in view of Lemma 11 we have

$$\mathbf{Pr}(k-1 \text{ free visits to } v \text{ after } t \mid \hat{a}_i, X_t = v, r) \leq \binom{k+r-2}{r-1} \left(\frac{C_3}{d}\right)^{k-1}.$$

From Lemma 10

$$\sum_{1 \leq t \leq \tau} h_v(t) \leq \frac{C_5 \ln n}{n \ln d},$$

and using equation (26) we finally obtain that

$$q_k \leq \frac{C_5 \ln n}{n \ln d} \sum_{1 \leq r \leq \infty} \binom{k+r-2}{r-1} \left(\frac{C_3}{d}\right)^{k-1} \sigma (1-\sigma)^{r-1} = \frac{C_5 \ln n}{n \ln d} \left(\frac{C_3}{\sigma d}\right)^{k-1},$$

which proves (27). From here we can proceed exactly as in the analysis of Phase 3 to show that the decrease in degree due to free visits is no more than $d/120$ whp, provided that α is small enough. By construction the reduction in degree due to start visits is at most $d/120$, so that the total reduction in degree during Phase 4 is at most $d/60$ as required. It remains to show that not too many pairs are deferred to Phase 5.

9. Analysis of Phase 5. We start by bounding the number of pairs not connected in Phase 4. Recall that a pair $\hat{a}_i, \hat{b}_i$ is not connected iff the total number of walks started from $\hat{a}_i$ would have exceeded $d/120$.

Fix $v \in V_3$. From equation (22) and the discussion that follows it, we have that for every i

$$\mathbf{Pr}(\hat{a}_i = v) \leq \frac{C_1^2 C_2^2}{n} \stackrel{\text{def}}{=} p.$$

Thus in view of (26) the number of starts from v is dominated by a random variable with the following probability generating function:

$$\sum_i \binom{\kappa}{i} p^i (1-p)^i \frac{\sigma x}{1 - x(1-\sigma)} = \left(\frac{\sigma p x}{1 - x(1-\sigma)} + 1 - p\right)^{\kappa}.$$

In general, given a probability generating function $f(x)$ for the random variable $X \geq 0$, and an integer $a \geq 1$ we have

$$\mathbf{Pr}(X \geq a) \leq \frac{f(x)}{x^a}, \qquad \rho > x \geq 1,$$

where ρ is the radius of convergence of f. So let X be the random variable that counts starts from v. Choosing

$$x = \frac{2+\sigma}{2+\sigma-\sigma^2} = 1 + \frac{\sigma^2}{2+\sigma-\sigma^2},$$

we obtain that

$$\begin{aligned}\mathbf{Pr}\left(X \geq \frac{d}{120}\right) &\leq \left(1+\frac{\sigma p}{2}\right)^{\kappa}\left(1+\frac{\sigma^2}{2+\sigma-\sigma^2}\right)^{-d/120} \\ &\leq \exp\left(\frac{\sigma p}{2}\kappa - \frac{\sigma^2}{3}\frac{d}{120}\right) \\ &= \exp\left(\frac{C_1^2 C_2^2 \sigma\alpha n d \ln d}{2n\ln n} - \frac{\sigma^2 d}{360}\right) \leq \exp\left(-\frac{\sigma^2 d}{400}\right)\end{aligned}$$

for α small enough.

At the end of Phase 4 we will be left with a set L of indices of pairs $(\hat{a}_i, \hat{b}_i)$ for which Phase 4 failed to find a path. The discussion above shows that $|L|$ is dominated in distribution by $\mathrm{Bin}(n, \exp(-\sigma^2 d/500))$, so whp $L = \emptyset$ if $d \geq 1000\sigma^{-2}\log n$ and otherwise $|L| \leq n^{1-\epsilon}$ for a *constant* $\epsilon > 0$. So assume that $d = O(\log n)$.

We join the pairs in L using a modification of the algorithm of [8]. That algorithm starts by splitting the edges of an expander graph to form two disjoint expanding subgraphs. This is unnecessary here as G_4 and G_5 will suffice for the two expander graphs; namely, G_4 can be used for the flow phase of [8] and then G_5 can be used for the random walks phase of [8]. The algorithm is capable of joining $\Omega(n/(\ln n)^c)$ pairs for some constant $c > 0$, provided the graph in the flow phase has edge expansion at least 1 and the second eigenvalue of the graph used in the random walks phase has a second eigenvalue bounded away from 1. Here whp we have fewer than $n^{1-\epsilon}$ pairs, the graph G_4 has an edge expansion $\Omega(\ln n)$, and the graph G_5 has a second eigenvalue of size $O(1/\sqrt{\ln n})$. So from this point of view there is room to spare.

On the other hand [8] only deals with the case where the required path endpoints are distinct. We will replace the flow phase of [8] with the following procedure. Suppose $v \in V_3$ is required to be an endpoint $\lambda(v)$ times. We have

$$\sum_{v \in V_3} \lambda(v) = 2|L| \leq 2n^{1-\epsilon}.$$

Furthermore, $\lambda(v)$ is dominated in distribution by $\mathrm{Bin}(\kappa, (1+o(1))n)$; hence $\mathbf{E}(\lambda(v)) = O(d\log d/\log n) = O(\log\log n)$ and whp

$$\lambda(v) \leq C_7 \log n/\log\log n \qquad \text{for all } v \in V_3.$$

We start Phase 5 by constructing for each $v \in V$ and $1 \leq i \leq \lambda(v)$ a set of $2/\epsilon$ random walks of length τ with start point v. We delete the edges of previous walks before beginning the next walk. The analysis of Phase 3 shows that we will succeed

in constructing these walks whp, since in Phase 3 the average number of walks per start point was $O(d \log d/logn)$ and the maximum was βd while the corresponding numbers are now $o(1)$ and $O(d/\log d)$.

The probability that k such walks all end at the endpoints already visited is bounded by

$$\binom{4\epsilon^{-1}|L|}{k} O(n^{-k}) = O(n^{-\epsilon k}),$$

so whp for each v and i at least one of the $2/\epsilon$ random walks ends up at a previously unvisited point. Thus we can associate with each $v \in V_3$ a set of $\lambda(v)$ endpoints of walks started from v, and all these sets are disjoint. From here we can continue with the second phase of [8] on G_5.

It only remains to prove Theorem 7.

10. Proof of Theorem 7. Now it is not too difficult to verify that the second eigenvalue of the walk on Γ_1 is not too large. There is however a technical problem in the fact that we are deleting the edges of a random graph by a process that conditions the distribution. We overcome this by considering graphs with a fixed degree sequence and consequently the configuration model of multigraphs. (We need now only consider $G_{n,m}$. To handle $G_{n,p}$ we simply condition on the number of edges being close to the expected number.)

10.1. Configuration model. The graphs G_i, $2 \le i \le 5$ will be random given their degree sequences. This is because the executions of lines 5 and 8 of SPLIT (Figure 1) do not condition the remaining graphs, once we are given their degree sequences. This idea has been used several times previously; see, for example, Bollobàs, Fenner, and Frieze [5].

The simplest model for graphs with a fixed degree sequence is the *configuration model* of Bollobás [3], which is a probabilistic interpretation of the counting formula of Bender and Canfield [2]. Let $\mathbf{d} = \{d_1, d_2, \ldots, d_\nu\}$ denote a degree sequence, $D_i = \{1, \ldots, d_i\} \times \{i\}$ for $1 \le i \le \nu$ and $D = \cup_{i=1}^{\nu} D_i$. Let $\Omega = \Omega(D)$ be the set of partitions of D into pairs. If $F \in \Omega$ then the multigraph $M = M(F)$ is defined as follows: $V(M) = [\nu]$ and there is an edge $\{i, j\}$ for every pair in F of the form $\{(x, i), (y, j)\}$ (for some x and y). It is unfortunate that we have to introduce multigraphs, but the salient properties of M are as follows in Lemma 10.

LEMMA 12. (a) *If M is simple, then it is equally likely to be any simple graph with degree sequence* $\mathbf{d}$.

(b) $\mathbf{Pr}(M \text{ is simple}) = \exp\{-O(\mu^2/\nu^2)\}$ *where $\mu = |D|/2$ is the number of edges in M; hence $2\mu/\nu$ is the average degree of M.*

We consider the probability space of multigraphs $M(F, \phi)$ where F is chosen randomly from Ω. We are interested in the case where

$$\delta = \min \mathbf{d} \ge d/C_2,$$
$$\Delta = \max \mathbf{d} \le C_1 d.$$

It will be useful to think of F as being constructed sequentially by the algorithm CONSTRUCT depicted in Figure 5.

It is important to observe that for any $t > 0$, $F \setminus F_t$ is a random member of $\Omega(R_t)$.

An important consequence of the above observation is that if we start with $M = M(F)$, then the multigraph obtained by removing from M the edges of a random

```
1.   algorithm CONSTRUCT
2.   begin
3.       F_0 ← ∅; R_0 ← W
4.       for t = 1 to m* do
5.           Choose u_t ∈ R_{t−1} arbitrarily
6.           Choose v_t randomly from R_{t−1} \ {u_t}
7.           F_t ← F_{t−1} ∪ {{u_t, v_t}}; R_t ← R_{t−1} \ {u_t, v_t}
8.       od
9.   output F
10.  end CONSTRUCT
```

FIG. 5. *Algorithm* CONSTRUCT.

walk W remains random. Indeed, we may imagine CONSTRUCT as performed in parallel with our walk W. Suppose our walk makes a transition from a vertex x, and the current value of R_t in CONSTRUCT is R. The transition from x is equivalent to choosing a random member $u = u_t$ of D_x. If $u \in R$, then we perform one step of CONSTRUCT and pair u with a point $v = v_t \in R \setminus \{u\}$. If $v \in D_y$ for some y, then the walk makes a transition from x to y. If $u \notin R$ then v is the point already paired with u. Thus since $F \setminus F_t$ is random, we see that removing from M the edges of a random walk results in a multigraph from a *random* configuration.

10.2. Random walks on configurations. We only discuss G_3 since the situation for G_2 is identical. Suppose G_3 has degree sequence $\mathbf{d}' = (d'_1, d'_2, \dots, d'_\nu)$, where $\nu = n(1-o(1))$. As observed, G_3 is random given its degree sequence. In our analysis, we want to consider G_3 as of the form $M(F)$ conditional on it being simple.

Each of the κ iterations deletes some pairs from F. Suppose $F^{(i)}$ denotes the remaining pairs at the start of iteration i and $D^{(i)} = \bigcup F^{(i)}$. If we ignore the condition that $M(F)$ is simple, then $F^{(i)}$ is a random member of $\Omega(D^{(i)})$. This requires a little justification. Our algorithm produces paths by choosing $\hat{a}_1, \hat{a}_2, \dots, \hat{a}_\kappa$ and $\hat{b}_1, \hat{b}_2, \dots, \hat{b}_\kappa$ at random and by applying WALK. As observed in Lemma 9, this is equivalent to just applying WALK1 a number of times. By our arguments of the previous section, deleting edges in the walks produced by WALK1 leaves a random configuration.

Thus we may imagine that initially we have a multigraph M_1. Then for $i \geq 2$ we apply GENPATHS to M_{i-1} and eventually produce M_i in which case M_i is a multigraph from a random configuration (when its degree sequence is given).

All that remains now is to show that (15) holds with suitably high probability for M_i, $i \geq 1$, conditioned on it being simple.

10.3. Eigenvalues. We will prove (15) by imitating the proof of Kahn and Szemerédi [10].

Let $\mathbf{d} = d_1, d_2, \dots, d_n$ be a degree sequence with maximum $\Delta = o(n^{1/2})$ and minimum $\delta > 0$ such that $\Delta/\delta < \theta$ for some constant θ. (Strictly speaking we should be concerned with $\mathbf{d} = d_1, d_2, \dots, d_\nu$, but $\nu = n - o(n)$ whp and n is "friendlier.") Let $M = M(F)$ be the multigraph on $[n]$ formed from a random configuration $F \in \Omega(\mathbf{d})$. Use e_{uv} to denote the number of edges joining vertices u and v. Consider the Markov

chain of a random walk on M. The transition matrix of the chain is

$$P_{uv} = \frac{e_{uv}}{d_u}.$$

Note that since the Markov chain is reversible, all eigenvalues of P are real and the largest eigenvalue of P equals 1. The eigenvalues are denoted by

$$1 = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n.$$

We need to show that conditional on M being simple with probability $1 - O(n^{-3})$,

$$\rho^* = \max\{|\lambda_2|, |\lambda_n|\} \leq \gamma/\sqrt{d}, \tag{28}$$

where $d = \sum_{i=1}^{n} d_i/n$ and $\gamma = \gamma(\theta)$.

LEMMA 13. *Let A be the matrix*

$$A_{uv} = \frac{e_{uv}}{d_u d_v},$$

and let

$$\rho_1 = \max\left\{|y^t A y| : \sum_u y_u = 0, \sum_u y_u^2 = 1\right\}.$$

Then $\rho^* \leq \Delta\rho_1$.

Proof. Let Q be the matrix

$$Q_{uv} = \frac{e_{uv}}{d_u^{1/2} d_v^{1/2}}.$$

Note that Q and P are similar; that is, $Q = DPD^{-1}$ where D is a diagonal matrix with diagonal elements $d_1^{1/2}, \ldots, d_n^{1/2}$ and so (λ, v) is an eigenvalue–eigenvector pair of P iff (λ, Dv) is an eigenvalue–eigenvector pair of Q. Since the largest eigenvalue of Q is 1 with eigenvector $(d_1^{1/2}, d_2^{1/2}, \ldots, d_n^{1/2})$, the Rayleigh quotient principle gives that

$$\rho^* = \max\left\{\frac{|\sum_{u,v} x_u Q_{uv} x_v|}{\sum_u x_u^2} \;\middle|\; \sum_u x_u d_u^{1/2} = 0\right\}.$$

Since

$$\sum_{u,v} x_u Q_{uv} x_v = \sum_{u,v} x_u d_u^{1/2} A_{uv} x_v d_v^{1/2}$$

and

$$\sum_u x_u^2 \geq \frac{1}{\Delta} \sum_u x_u^2 d_u,$$

we have, on putting $y_u = x_u d_u^{1/2}$,

$$\rho^* \leq \Delta \max\left\{|y^t A y| \;\middle|\; \sum_u y_u = 0, \sum_u y_u^2 = 1\right\} = \Delta\rho_1. \qquad \square$$

Following Kahn and Szemerédi, choose a real $\epsilon \in (0,1)$ (eventually ϵ will be fixed in equation (38)), and let

$$T = \left\{ x \in \left(\frac{\epsilon}{n^{1/2}}\boldsymbol{Z}\right)^n \;\middle|\; \sum_u x_u = 0, \sum_u x_u^2 \le 1 \right\},$$

where $\boldsymbol{Z}$ denotes the set of integers. Then by considering the total volume of cubes of side $\epsilon/\sqrt{n}$ which have their centers in T, we see that

$$\begin{aligned} |T| &\le \left(\frac{n^{1/2}}{\epsilon}\right)^n \boldsymbol{V}\mathrm{ol}\left(\left\{ x \in \boldsymbol{R}^n \;\middle|\; \sum_u x_u^2 \le \left(1+\frac{\epsilon}{2}\right) \right\} \right) \\ &= \left(\frac{(2+\epsilon)n^{1/2}}{2\epsilon}\right)^n \frac{\pi^{n/2}}{\Gamma(n/2+1)} \le \left(\frac{(2+\epsilon)n^{1/2}}{2\epsilon}\right)^n \frac{\pi^{n/2}e^{n/2}}{(n/2)^{n/2}\sqrt{\pi n}} \\ &\le \left(\frac{(2+\epsilon)\sqrt{2\pi e}}{2\epsilon}\right)^n. \end{aligned} \tag{29}$$

LEMMA 14. *Let ρ_1 be defined as in Lemma 13. We claim that*

$$\rho_1 \le (1-\epsilon)^{-2}\rho,$$

where

$$\rho = \max\{ |x^t A y| \mid x, y \in T \}.$$

Proof. Let $S = \{ x \in \boldsymbol{R}^n \mid \sum_u x_u = 0, \sum_u x_u^2 \le 1 \}$. We first show that for every $x \in S$ there is a $y \in T$ such that $x - y \in \epsilon S$ and $\|x - y\| \le \epsilon$. Suppose that for $i = 1, 2, \ldots, n$,

$$x_i = \epsilon m_i n^{-1/2} + f_i, \qquad m_i \in \boldsymbol{Z}, \qquad f_i \in [0, \epsilon n^{-1/2}).$$

Note that since $\sum_u x_u = 0$, we have $\sum_i f_i = \epsilon f n^{-1/2}$, where f is a nonnegative integer less than n. Rearrange subscripts so that $m_i \le m_j$ whenever $i \le j$. Define a vector $y \in \boldsymbol{R}^n$ so that

$$y_u = \begin{cases} \epsilon(m_u+1)n^{-1/2} & \text{if } u \le f, \\ \epsilon m_u n^{-1/2} & \text{if } u > f. \end{cases}$$

Then we have

(a) $\displaystyle\sum_u y_u = \sum_u x_u = 0,$

(b) $\displaystyle\sum_u y_u^2 \le \sum_u x_u^2 \le 1,$

(c) $\|x - y\| \le \epsilon$ (since $|x_u - y_u| \le \epsilon n^{-1/2}$).

Thus y is in T and has the required property. It follows that one can apply the above construction to obtain that for any $x \in S$ there are $x^{(0)}, x^{(1)}, \ldots$ in T such that

$$x = \sum_i \epsilon^i x^{(i)},$$

and therefore for any $x \in S$ there are $x^{(i)} \in T$ such that

$$x^t Ax = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty}\left(x^{(i)}\right)^t Ax^{(j)}\epsilon^{i+j} \le (1-\epsilon)^{-2}\max\{|y^tAz| : y,z \in T\}.$$

The lemma now follows. □

Now write

$$\rho = \max\{|x^tMy| : x, y \in T\}.$$

Our aim is to find a probabilistic upper bound for ρ of order $O(\Delta^{-3/2})$ that will verify (28). This is done by considering the random variables $X = X(x,y) = \sum_{u,v} x_u A_{uv} y_v$, where $x, y \in T$. Note that for any two distinct points in the configuration the probability that the two points are joined by an edge is $1/(2m-1)$, where $2m = \sum_{i=1}^{n} d_i$. Thus for $u \neq v$,

$$\mathbf{E}[e_{uv}] = \frac{d_u d_v}{2m-1}$$

and

$$\mathbf{E}[e_{uu}] = \frac{d_u(d_u-1)}{2(2m-1)}.$$

Fix $x, y \in T$ and define

$$B = \left\{(u,v) \;\middle|\; 0 < |x_u y_v| < \Delta^{1/2}/n\right\}. \tag{30}$$

Let

$$X' = \sum_{(u,v)\in B} x_u A_{uv} y_v \quad \text{and} \quad X'' = \sum_{(u,v)\notin B} x_u A_{uv} y_v,$$

so that $X = X' + X''$.

10.4. Estimating X'. Note that

$$\mathbf{E}[X'] = \sum_{(u,v)\in B}\frac{x_u y_v}{2m-1} + \sum_{(u,u)\in B}\frac{x_u y_u (d_u-1)}{2(2m-1)d_u}.$$

Write S_1 and S_2 for the first and second sums in the above equation. Then

$$|S_2| \le \sum_{(u,u)\in B}\frac{|x_u y_u|(d_u-1)}{2(2m-1)d_u} \le \frac{\Delta^{1/2}}{4m}. \tag{31}$$

For S_1 we follow Lemma 2.4 in [10]. Since $\sum_u x_u = \sum_v y_v = 0$ we have $\sum_{u,v} x_u y_v = 0$ and so

$$\left|\sum_{(u,v)\in B} x_u y_v\right| = \left|\sum_{(u,v)\notin B} x_u y_v\right|.$$

Now

$$\left|\sum_{(u,v)\notin B} x_u y_v\right| \le \sum_{|x_u y_v| \ge \Delta^{1/2}/n}\frac{x_u^2 y_v^2}{|x_u y_v|} \le \frac{n}{\Delta^{1/2}}\sum_{u,v} x_u^2 y_v^2 \le \frac{n}{\Delta^{1/2}}.$$

Hence

$$|\,\mathbf{E}[X']| \le \frac{n}{(2m-1)\Delta^{1/2}} + \frac{\Delta^{1/2}}{4m} = (1+o(1))\frac{n}{2m\Delta^{1/2}}. \tag{32}$$

We next show that X' is concentrated around its mean. For this we need some more notation. Recall that a configuration is a perfect matching F of the set $W = \cup_{i=1}^{n}\{i\}\times[d_i]$. We call the elements in W points and assume that the points in W are ordered lexicographically. For $\alpha \in W$, let $v(\alpha)$ denote the first component of α, and for a pair $e = \{\alpha, \beta\}$ in a configuration with $\alpha < \beta$, we write $t(e)$ for $v(\alpha)$ and $h(e)$ for $v(\beta)$. For real x, define

$$\chi(x) = \begin{cases} x & \text{if } |x| < \Delta^{1/2}/n, \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$X' = \sum_{e\in F}\frac{\chi(x_{t(e)}y_{h(e)})}{d_{t(e)}d_{h(e)}} + \sum_{e\in F}\frac{\chi(x_{h(e)}y_{t(e)})}{d_{t(e)}d_{h(e)}} = X'_a + X'_b, \quad \text{say.} \tag{33}$$

We next write $F = F_1 \cup F_2 \cup F_3$ where

$$\begin{aligned} F_1 &= \{e \in F : |x_{t(e)}| > n^{-1/2}/\epsilon\}, \\ F_2 &= \{e \in F : |y_{h(e)}| > n^{-1/2}/\epsilon, |x_{t(e)}| \le n^{-1/2}/\epsilon\}, \\ F_3 &= \{e \in F : |y_{h(e)}| \le n^{-1/2}/\epsilon, |x_{t(e)}| \le n^{-1/2}/\epsilon\}. \end{aligned}$$

Then let

$$X_i = \sum_{e\in F_i}\frac{\chi(x_{t(e)}y_{h(e)})}{d_{t(e)}d_{h(e)}} \qquad \text{for } i = 1,2,3,$$

so that

$$X'_a = X_1 + X_2 + X_3.$$

Recall that $\Delta/\delta < \theta$, a constant. We claim Lemma 15.

LEMMA 15. *There are constants $B_i = B_i(\theta) > 0$ for $i = 1,2,3$ such that for any $t > 0$,*

$$\mathbf{Pr}(|X_1 - \mathbf{E}[X_1]| \ge t\Delta^{-3/2}) \le 2\exp(-tn + B_1 n), \tag{34}$$

$$\mathbf{Pr}(|X_2 - \mathbf{E}[X_2]| \ge t\Delta^{-3/2}) \le 2\exp(-tn + B_2 n), \tag{35}$$

$$\mathbf{Pr}(|X_3 - \mathbf{E}[X_3]| \ge t\Delta^{-3/2}) \le 2\exp(-tn + B_3 n). \tag{36}$$

Proof. We first prove (34). Assume without loss of generality that $|x_i| \ge |x_{i+1}|$ for all i and let the pairs $\{\alpha_i, \beta_i\}$ for $1 \le i \le m$ that compose F be ordered such that $\alpha_i < \beta_i$ and $\alpha_i < \alpha_{i+1}$. Recall that the order among points is lexicographic; thus $v(\alpha_i) \le v(\alpha_{i+1})$ and $|x_{v(\alpha_i)}| \ge |x_{v(\alpha_{i+1})}|$.

Let $\equiv_k$ be the equivalence relation on Ω such that $F \equiv_k F'$ if and only if the sequences of the first k pairs in F and F' are identical. Write Ω_k for the set of equivalence classes, and $\mathcal{F}_k$ for the corresponding σ-algebra. Define $Y_k = \mathbf{E}[X_1|\mathcal{F}_k]$; that is, Y_k is a function from Ω to $\boldsymbol{R}$ so that $Y_k(F)$ equals the expected value of

X_1 conditional on the first k pairs being exactly equal to the first k pairs in F. Now $Y_0, Y_1, \ldots, Y_m$ is a Doob martingale with $Y_0 = \mathbf{E}[X_1]$ and $Y_m = X_1$. Define $Z_k = Y_k - Y_{k-1}$. Note that as in Lemma 2.7 in [10], if there is $f_k(\zeta)$ such that $Z'_k = \mathbf{E}[\exp(\zeta^2 Z_k^2)|\mathcal{F}_{k-1}] \le f_k(\zeta)$, then for all t and $\zeta > 0$,

$$\mathbf{Pr}(|X_1 - \mathbf{E}[X_1]| \ge t) \le 2e^{-\zeta t} \prod_{k=1}^{m} f_k(\zeta). \tag{37}$$

We next write down the distribution of Z_k. Define

$$\hat{\chi}(x,y) = \begin{cases} xy & \text{if } |xy| < \Delta^{1/2}/n \text{ and } |x| > n^{-1/2}/\epsilon, \\ 0 & \text{otherwise.} \end{cases}$$

For a pair $e = \{\alpha, \beta\}$ in F with $\alpha < \beta$, write

$$q(e) = \frac{\hat{\chi}(x_{v(\alpha)}, y_{v(\beta)})}{d_{v(\alpha)} d_{v(\beta)}}.$$

Then

$$X_1(F) = \sum_{e \in F} q(e).$$

Note that we can express

$$Z_k(F) = \frac{2^{m-k}(m-k)!}{(2m-2k)!} \left\{ \sum_{F' \equiv_k F} X_1(F') - \frac{1}{2m-2k+1} \sum_{F'' \equiv_{k-1} F} X_1(F'') \right\}.$$

Let $\{\alpha, \beta\}$ be the kth pair in F with $\alpha < \beta$ and let J be the set of points contained in the first k pairs in F. For $\eta \notin J - \{\beta\}$ and for $F' \equiv_k F$, we define F'_η as follows. Suppose that η is matched with γ in F'. Write $e = \{\alpha, \beta\}$, $f = \{\eta, \gamma\}$, $e' = \{\alpha, \eta\}$, $f' = \{\gamma, \beta\}$. Then F'_η is defined to be $(F' - \{e, f\}) \cup \{e', f'\}$, giving $F'_\eta \equiv_{k-1} F$ and $F'_\beta = F'$. Note also that $\{\{F'_\eta \mid \eta \notin J - \{\beta\}\} \mid F' \equiv_k F\}$ is a partition of $\{F'' \mid F'' \equiv_{k-1} F\}$. Thus

$$Z_k(F) = \frac{2^{m-k}(m-k)!}{(2m-2k)!} \frac{1}{2m-2k+1} \sum_{F' \equiv_k F} \sum_{\eta \notin J} \left(X_1(F') - X_1(F'_\eta) \right).$$

Also, since

$$X_1(F') - X_1(F'_\eta) = q(e) + q(f) - q(e') - q(f'),$$

we have

$$Z_k(F) = \sum_{\eta \notin J} \sum_{\gamma \notin J, \gamma \ne \eta} \frac{q(\{\alpha,\beta\}) + q(\{\gamma,\eta\}) - q(\{\alpha,\eta\}) - q(\{\gamma,\beta\})}{(2m-2k+1)(2m-2k-1)}.$$

Note that since $\sum x_u^2 \le 1$, there at most $n\epsilon^2$ indices u such that $|x_u| > n^{-1/2}/\epsilon$. Thus $Z_k = 0$ if $k \ge \epsilon^2 \Delta n$; otherwise

$$2m - 2k - 1 \ge 2m - 2\epsilon^2 \Delta n - 1 \ge \delta n$$

if we choose ϵ so that

$$\frac{\epsilon^2\Delta}{\delta} = \frac{1}{3}. \tag{38}$$

Therefore,

$$|Z_k(F)| \le \frac{1}{(\delta n)^2} \sum_{\eta\notin J} \sum_{\gamma\notin J, \gamma\neq\eta} \{|q(\{\alpha,\beta\})| + |q(\{\gamma,\eta\})| + |q(\{\alpha,\eta\})| + |q(\{\gamma,\beta\})|\}.$$

Let

$$y^\alpha = \frac{1}{|x_{v(\alpha)}|} \min\{|yx_{v(\alpha)}|, \Delta^{1/2}/n\},$$

and note that $x_{v(\alpha)} \ge \max\{x_{v(\beta)}, x_{v(\gamma)}, x_{v(\eta)}\}$ and that $|x_{v(\eta)}| \le |x_{v(\alpha)}|$ implies $|x_{v(\eta)}|y^\eta \le |x_{v(\alpha)}|y^\alpha$. Therefore,

$$\begin{aligned}
&|q(\{\alpha,\beta\})| \le \delta^{-2}|x_{v(\alpha)}|y^\alpha_{v(\beta)},\\
&|q(\{\gamma,\eta\})| \le \delta^{-2}(|x_{v(\gamma)}|y^\gamma_{v(\eta)} + |x_{v(\eta)}|y^\eta_{v(\gamma)}) \le \delta^{-2}(|x_{v(\alpha)}|y^\alpha_{v(\eta)} + |x_{v(\alpha)}|y^\alpha_{v(\gamma)}),\\
&|q(\{\alpha,\eta\})| \le \delta^{-2}|x_{v(\alpha)}|y^\alpha_{v(\eta)},\\
&|q(\{\gamma,\beta\})| \le \delta^{-2}(|x_{v(\gamma)}|y^\gamma_{v(\beta)} + |x_{v(\beta)}|y^\beta_{v(\gamma)}) \le \delta^{-2}(|x_{v(\alpha)}|y^\alpha_{v(\beta)} + |x_{v(\alpha)}|y^\alpha_{v(\gamma)}).
\end{aligned}$$

Next observe that since $\sum y_u^2 \le 1$ implies $\sum |y_u| \le n^{1/2}$, we have for example

$$\sum_{\eta\notin J} y^\alpha_{v(\eta)} \le \sum_{\eta\notin J} |y_{v(\eta)}| \le \Delta \sum_{w=1}^{n} |y_w| \le \Delta n^{1/2}.$$

Thus we have

$$|Z_k(F)| \le 4\Delta^2\delta^{-4}|x_{v(\alpha)}|(y^\alpha_{v(\beta)} + n^{-1/2}).$$

Writing $Z'_k = \mathbf{E}[\exp(\zeta^2 Z_k^2)|\mathcal{F}_{k-1}]$, we have

$$Z'_k(F) \le \frac{1}{2m-2k-1} \sum_{\nu\notin J-\{\beta\}} \exp(16\zeta^2\Delta^4\delta^{-8}(x_{v(\alpha)})^2(y^\alpha_{v(\nu)} + n^{-1/2})^2).$$

Take

$$\zeta = \Delta^{3/2}n, \tag{39}$$

which means that the expression

$$\begin{aligned}
&\zeta^2\Delta^4\delta^{-8}(x_{v(\alpha)})^2(y^\alpha_{v(\nu)} + n^{-1/2})^2\\
&\qquad = \zeta^2\Delta^4\delta^{-8}\Big\{(x_{v(\alpha)}y^\alpha_{v(\nu)})^2 + 2(x_{v(\alpha)}y^\alpha_{v(\nu)})x^\alpha_{v(\alpha)}n^{-1/2} + (x_{v(\alpha)})^2 n^{-1}\Big\}
\end{aligned}$$

is bounded by $4\theta^8\epsilon^{-2}$.

(Here we use $x_{v(\alpha)}y^\alpha_{v(\nu)} \le \sqrt{\Delta}/n$ and $y^\alpha_{v(\nu)} \ge \epsilon/\sqrt{n}$ (since $y_{v(\nu)} \neq 0$) to get $x_{v(\alpha)} \le \sqrt{\Delta}/(\epsilon\sqrt{n})$.)

Hence putting $B = \exp\{64\theta^8\epsilon^{-2}\}$ and using $e^x \le 1 + xe^x$ for $x \ge 0$,

$$\begin{aligned} Z'_k(F) &\le 1 + \frac{B}{2m-2k-1} \sum_{\nu \notin J - \{\beta\}} \zeta^2 \Delta^4 \delta^{-8} (x_{v(\alpha)})^2 (y^\alpha_{v(\nu)} + n^{-1/2})^2 \\ &\le 1 + B\zeta^2\Delta^4\delta^{-9} n^{-1} (x_{v(\alpha)})^2 \sum_\nu (y^2_{v(\nu)} + 2|y_{v(\nu)}| n^{-1/2} + n^{-1}) \\ &\le \exp(4B\zeta^2\Delta^5\delta^{-9} n^{-1} (x_{v(\alpha)})^2). \end{aligned}$$

Writing $r(k) = \lceil k/\Delta \rceil$, we have for any $F \in \Omega$,

$$Z'_k(F) \le \exp(4B\zeta^2\Delta^5\delta^{-9} n^{-1} (x_{r(k)})^2).$$

Thus, using (37), we have

$$\begin{aligned} \mathbf{Pr}(|X_1 - \mathbf{E}[X_1]| \ge t\Delta^{-3/2}) &\le 2e^{-\zeta t\Delta^{-3/2}} \exp\left(\sum_{k=1}^m 4B\zeta^2\Delta^5\delta^{-9} n^{-1} (x_{r(k)})^2 \right) \\ &\le 2\exp\left(-tn + 4B\Delta^8\delta^{-9} n \sum_{k=1}^m (x_{r(k)})^2 \right) \\ &\le 2\exp\left(-tn + 4B\Delta^9\delta^{-9} n \right). \end{aligned} \tag{40}$$

This proves (34).

The proof of (35) is almost identical even though F_2 has a slightly different definition of F_1. We simply reorder F according to $y_{v(\beta)}$ and go through the proof above without using the condition $|x_{v(\alpha)}| \le n^{-1/2}/\epsilon$.

The proof of (36) is much simpler. We use the more usual martingale argument (Alon and Spencer [1], Bollobás [4], McDiarmid [14]); for now if $Y_k = \mathbf{E}(X_3 \mid \mathcal{F}_k)$ then $|Y_k - Y_{k-1}| \le 4/(\epsilon^2 n\delta^2)$. Since we took (in (38)), $\epsilon = 1/\sqrt{3\theta}$, we have

$$\begin{aligned} \mathbf{Pr}(|X_3 - \mathbf{E}[X_3]| \ge t\Delta^{-3/2}) &\le 2\exp\left(\frac{-t^2\epsilon^4 n^2\delta^4}{32\Delta^3 m} \right) \\ &\le 2\exp\left(\frac{-t^2\epsilon^4 n}{32\theta^4} \right) \le 2\exp\left(\frac{-t^2 n}{288\theta^6} \right). \end{aligned}$$

□

Note that the lemma above shows that there is a constant $B > 0$ such that

$$\mathbf{Pr}(|X'_a - \mathbf{E}[X'_a]| \ge t\Delta^{-3/2}) \le 6\exp(-tn + Bn).$$

Clearly the same result holds for the second sum X'_b in (33). Thus we have that for any $\hat{\xi} < \xi \in (0,1)$ there is a $K = K(\theta, \xi) > 0$ such that

$$\mathbf{Pr}(|X' - \mathbf{E}[X']| \ge K\Delta^{-3/2} \mid M \text{ is simple}) \le 2^{O(d^2)} \hat{\xi}^n \le \xi^n. \tag{41}$$

Note that we should multiply the right-hand side of (41) by $\kappa \le n^2$ to account for the probability there exists M_i for which X' is large.

10.5. Estimating X''. In view of (41), it only remains to show that $X'' = O(\Delta^{-3/2})$ with suitably high probability. We shall first prove a preliminary result showing that the random graph G with degree sequence $\mathbf{d}$ is unlikely to have a *dense* subgraph. It will be enough to consider the case $G = G_3$ and argue an immediate implication for its subgraphs.

LEMMA 16. *Let G be chosen randomly from the set $\mathcal{G}(\mathbf{d})$ of simple graphs with vertex set $[n]$ and degree sequence $\mathbf{d}$. For $A, B \subseteq [n]$, let $e(A,B)$ be the number of edges joining a vertex in A to a vertex in B and $\mu(A,B) = \theta|A||B|\Delta/n$, where $\theta > \Delta/\delta$ is sufficiently large. For every constant $K > 0$ there is a constant $C = C(\theta, K)$ such that with probability $1 - o(n^{-K})$ every pair of $A, B \subseteq [n]$, with $|A| \le |B|$, satisfies at least one of the following:*

(i) $e(A,B) \le C\mu(A,B)$,

(ii) $e(A,B)\ln\frac{e(A,B)}{\mu(A,B)} \le C|B|\ln\frac{n}{|B|}$.

Proof. Write $a = |A|$, $b = |B|$, and let d_A and d_B be the sums of degrees in A and in B, respectively. Condition (i) clearly holds *deterministically* if b is at least a constant fraction of n since $e(A,B) \le a\Delta$. Assume then that $a, b \le n/(4\theta)$.

We prove later that for any set of possible edges S, $|S| \le n\Delta/(4\theta) \le n\delta/2$, we have

$$\mathbf{Pr}(G \text{ contains } S) \le \left(\frac{\Delta^2}{m}\right)^{|S|}. \tag{42}$$

Thus the probability that there exists a pair (A,B) with $e(A,B) = t$ is at most

$$\binom{n}{b}\binom{n}{a}\binom{ab}{t}\left(\frac{\Delta^2}{m}\right)^t \le \left(\frac{ne}{b}\right)^{2b}\left(\frac{abe\Delta^2}{mt}\right)^t \le \left(\frac{ne}{b}\right)^{2b}\left(\frac{\mu(A,B)}{t}\right)^t e^t.$$

Now consider a value x that satisfies

$$\begin{aligned} x\ln\left(\frac{x}{\mu(A,B)}\right) &> Cb\ln\left(\frac{n}{b}\right) \ge \frac{1}{2}Cb\ln\left(\frac{en}{b}\right) \\ x &> C\mu(A,B) \\ x &\ge (\ln n)^2. \end{aligned}$$

Then clearly

$$\mathbf{Pr}(\exists A, B:\ e(A,B) = x) \le n^{-\ln n},$$

and therefore

$$\begin{aligned} &\mathbf{Pr}\left(\exists A, B:\ (e(A,B) \ge (\ln n)^2)\ \&\ \neg(\mathrm{i})\ \&\ \neg(\mathrm{ii})\right) \\ &\qquad = \sum_{(\ln n)^2 \le x \le n^2} \mathbf{Pr}\Big(\exists A, B:\ (e(A,B) = x)\ \&\ \neg(\mathrm{i})\ \&\ \neg(\mathrm{ii})\Big) \le n^2 n^{-\ln n}. \end{aligned}$$

It remains to deal with $\mathbf{Pr}\left(\exists A, B:\ (e(A,B) < (\ln n)^2)\ \&\ \neg(\mathrm{i})\ \&\ \neg(\mathrm{ii})\right)$. If $e(A,B) < (\ln n)^2$ and (ii) does not hold then

$$2e(A,B)\ln n > Cb\ln\left(\frac{n}{b}\right) \ge Cb\ln(4\theta) \tag{43}$$

and so $b \le e(A,B)\ln n \le (\ln n)^3$, which in turn, from the first inequality in (43), implies that $e(A,B) > Cb/3$. But the probability that $e(A,B) \ge Cb/3$, for C sufficiently large, and $b \le (\ln n)^3$ can be bounded by

$$\begin{aligned} \binom{n}{b}\binom{n}{a}\sum_{t=Cb/3}^{\Delta(\ln n)^3}\binom{ab}{t}\left(\frac{\Delta^2}{m}\right)^t &\le 2\binom{n}{b}\binom{n}{a}\binom{ab}{Cb/3}\left(\frac{\Delta^2}{m}\right)^{Cb/3} \\ &\le 2\left(\frac{ne}{b}\right)^{2b}\left(\frac{3ea\Delta^2}{Cm}\right)^{Cb/3} \\ &\le 2(n^{2-3C/10}b^{C/3-2})^b. \end{aligned}$$

This yields the conclusion of the lemma.

Proof of (42). Let $S = \{e_1, e_2, \ldots, e_s\}$, $\mathcal{G}_0 = \mathcal{G}(\mathbf{d})$ and $\mathcal{G}_i = \{G \in \mathcal{G}(\mathbf{d}) : G$ contains $\{e_1, e_2, \ldots, e_i\}$ for $1 \le i \le s$. It is sufficient to prove that for $0 \le i < s$,

$$\frac{|\mathcal{G}_{i+1}|}{|\mathcal{G}_i|} \le \frac{\Delta^2}{2m - 2\Delta^2 - 2s} \le \frac{\Delta^2}{m}, \tag{44}$$

where the second inequality follows from our bound on s. To prove the first inequality we consider

$$X = \{(H_1, H_2) : H_1 \in \mathcal{G}_i \setminus \mathcal{G}_{i+1}, H_2 \in \mathcal{G}_{i+1}, H_1 \sim H_2\},$$

where $H_1 \sim H_2$ means that there is some four-cycle with edges $f_1 = e_{i+1}, f_2, f_3, f_4$ such that H_2 is obtained from H_1 by adding f_1, f_3 and deleting f_2, f_4. The first inequality in (44) follows immediately from the following:

(i) a particular $H_1 \in \mathcal{G}_i \setminus \mathcal{G}_{i+1}$ appears in at most Δ^2 pairs of X;

(ii) a particular $H_2 \in \mathcal{G}_{i+1}$ appears in at least $2m - 2\Delta^2 - s$ pairs of X.

Let $e_{i+1} = (x, y)$. For (i) observe that there are at most Δ^2 choices for f_2, f_4—one is incident with x and one is incident with y. For (ii), given $H_2 \in \mathcal{G}_{i+1}$ we chose an *oriented* edge $f_3 = (u, v) \in H_2$ not incident with e_{i+1}. Let $f_2 = (x, u)$ and $f_4 = (y, v)$. At most $2(\Delta - 1)^2$ choices of f_3 are forbidden because at least one of f_2, f_4 are already in H_1 and at most $s - 1$ choices are disallowed because $f_3 \in S$. □

We now explain why it suffices just to consider G_3 (and G_2) for the large pairs and not their subgraphs $\hat{\Gamma}_j$ (and Γ_j). Indeed, if one of the conditions (i) or (ii) hold for G_3 then at least one holds for any of its subgraphs Γ. If condition (i) was true for G_3 then it is true a fortiori for Γ. Similarly, if condition (i) fails, $C \ge 1$, and condition (ii) holds for G_3 then it holds a fortiori for Γ.

LEMMA 17. *Given the assertions in Lemma* 16, $X'' = \sum_{(u,v) \notin B} x_u A_{uv} y_v$, *where* $B = \{(u, v) \mid 0 < |x_u y_v| < \Delta^{1/2}/n\}$, *satisfies*

$$X'' = O(\Delta^{-3/2})$$

for every *pair* $x, y \in T$.

Proof. Given $x \in T$, we write

$$S_i(x) = \{u : \epsilon^{2-i} n^{-1/2} \le |x_u| < \epsilon^{1-i} n^{-1/2}\}, \qquad i \in I,$$

where $I = \{i : S_i(x) \ne \emptyset\}$. Define J and $S_j(y)$ analogously. Also, for $S \subseteq [n]$ and $x \in T$, write

$$(x_S)_u = \begin{cases} x_u & \text{if } u \in S, \\ 0 & \text{otherwise.} \end{cases}$$

Given $x, y \in T$, we write $A_i = S_i(x)$, $B_j = S_j(y)$, $a_i = |A_i|$, $b_j = |B_j|$. Let

$$\mathcal{C} = \{(i, j) \mid i, j > 0,\ \epsilon^{2-i-j} > \sqrt{\Delta},\ a_i \le b_j\},$$
$$\mathcal{C}' = \{(i, j) \mid i, j > 0,\ \epsilon^{2-i-j} > \sqrt{\Delta},\ a_i > b_j\}.$$

Since

$$X'' = \sum_{|x_u y_v| \ge \Delta^{1/2}/n} x_u A_{uv} y_v,$$

it is sufficient to show that

$$\sum_{(i,j)\in\mathcal{C}} (x_{A_i})^t A y_{B_j} = O(\Delta^{-3/2})$$

or, equivalently, if $e_{i,j} = e(A_i, B_j)$,

$$\frac{1}{n}\sum_{(i,j)\in\mathcal{C}} \frac{e_{i,j}}{\epsilon^{i+j}} = O(\sqrt{\Delta}).$$

The sum on $\mathcal{C}'$ follows by symmetry.

Note that since $\sum x_u^2 \le 1$ for $x \in T$, we have

$$\sum_{i\in I} a_i/\epsilon^{2(i-2)} \le n, \qquad \sum_{j\in J} b_j/\epsilon^{2(i-2)} \le n.$$

Next partition $\mathcal{C}$ into $\mathcal{C}_I \cup \mathcal{C}_{II}$, where $\mathcal{C}_x$ is the set of $(i,j) \in \mathcal{C}$ such that $e(A_i, B_i)$ satisfies assertion x in Lemma 16. First, using the definition of $\mathcal{C}$, we have

$$\frac{1}{n}\sum_{(i,j)\in\mathcal{C}_I} \frac{e_{i,j}}{\epsilon^{i+j}} = O\left(\frac{1}{n^2}\sum_{(i,j)\in\mathcal{C}_I} \frac{a_i b_j \Delta}{\epsilon^{i+j}}\right) = O\left(\frac{\Delta}{n^2}\sum_{(i,j)\in\mathcal{C}_I} \frac{a_i b_j}{\epsilon^{2(i+j)}\sqrt{\Delta}}\right) = O(\sqrt{\Delta}).$$

It therefore remains to show

$$\frac{1}{n}\sum_{(i,j)\in\mathcal{C}_{II}} \frac{e_{i,j}}{\epsilon^{i+j}} = O(\sqrt{\Delta}). \tag{45}$$

For $k = 1, \dots, 5$, let $\mathcal{D}_k$ be the set of $(i,j) \in \mathcal{C}_{II}$ satisfying (k) below but not (k') if $k' < k$.

(1) $\epsilon^j \ge \epsilon^i\sqrt{\Delta}$,
(2) $e_{i,j} \le \mu_{i,j}/(\epsilon^{i+j}\sqrt{\Delta})$, where $\mu_{i,j} = \mu(A_i, B_j)$,
(3) $\ln(e_{i,j}/\mu_{i,j}) \ge \frac{1}{4}\ln(n/b_j)$,
(4) $n/b_j \le \epsilon^{-4j}$,
(5) $n/b_j > \epsilon^{-4j}$.

Then equation (45) follows if for $k = 1, \dots, 5$,

$$H_k = \frac{1}{n}\sum_{(i,j)\in\mathcal{D}_k} \frac{e_{i,j}}{\epsilon^{i+j}} = O(\sqrt{\Delta}).$$

Start by noting that since $(i,j) \in \mathcal{C}_{II}$, we have

$$e_{i,j}\ln(e_{i,j}/\mu_{i,j}) \le C b_j \ln(n/b_j). \tag{46}$$

For $k = 1$, from the trivial inequality $e_{i,j} \le a_i\Delta$ we have

$$H_1 \le \frac{1}{n}\sum_i \sum_{j:\epsilon^j \ge \epsilon^i\sqrt{\Delta}} \frac{a_i\Delta}{\epsilon^{i+j}} = O\left(\frac{1}{n}\sum_i \frac{a_i\sqrt{\Delta}}{\epsilon^{2i}}\right) = O(\sqrt{\Delta}).$$

For $k = 2$, we have

$$H_2 \le \frac{1}{n}\sum_{i,j} \frac{\mu_{i,j}}{\sqrt{\Delta}\epsilon^{2(i+j)}} = O\left(\frac{\sqrt{\Delta}}{n^2}\sum_{i,j} \frac{a_i b_j}{\epsilon^{2(i+j)}}\right) = O(\sqrt{\Delta}).$$

For $k = 3$, equation (46) implies that

$$e_{i,j} = O(b_j),$$

and so using $(i,j) \notin \mathcal{D}_1$, that is, $\epsilon^j < \epsilon^i\sqrt{\Delta}$,

$$H_3 = O\Big(\frac{1}{n}\sum_j \sum_{i:\epsilon^i > \epsilon^j/\sqrt{\Delta}} \frac{b_j}{\epsilon^{i+j}}\Big) = O\Big(\frac{1}{n}\sum_j \frac{\sqrt{\Delta}b_j}{\epsilon^{2j}}\Big) = O(\sqrt{\Delta}).$$

For $k = 4$, using $(i,j) \notin \mathcal{D}_3$, we have

$$\frac{e_{i,j}}{\mu_{i,j}} \le \frac{1}{\epsilon^j}.$$

Also, using $(i,j) \notin \mathcal{D}_2$, we have

$$\frac{e_{i,j}}{\mu_{i,j}} \ge \frac{1}{\epsilon^{i+j}\sqrt{\Delta}},$$

thus giving

$$\epsilon^{-i} \le \sqrt{\Delta}.$$

From (46), we also have $e_{i,j} = O(jb_j)$ (using also $e_{i,j} \ge C\mu_{i,j}$). Thus

$$H_4 = O\Big(\frac{1}{n}\sum_j \sum_{i:\epsilon^{-i} \le \sqrt{\Delta}} \frac{jb_j}{\epsilon^{i+j}}\Big) = O\Big(\frac{\sqrt{\Delta}}{n}\sum_j \frac{jb_j}{\epsilon^j}\Big).$$

Since $\sum_{j\in J} b_j/(n\epsilon^{2j}) = O(1)$ we have

$$H_4 = O(\sqrt{\Delta}).$$

For $k = 5$, since $b_j < n\epsilon^{4j}$, we have from (46) that

$$e_{i,j} \le Cn\epsilon^{4j}\ln\epsilon^{-4j} = O(nj\epsilon^{4j}).$$

Also, since $(i,j) \notin \mathcal{D}_1$, we have $\epsilon^j < \epsilon^i\sqrt{\Delta}$; thus

$$H_5 = O\Big(\sum_j \sum_{i:\epsilon^i > \epsilon^j/\sqrt{\Delta}} j\epsilon^{3j-i}\Big) = O\Big(\sqrt{\Delta}\sum_j j\epsilon^{2j}\Big) = O(\sqrt{\Delta}). \qquad \square$$

Observe finally that for future reference we have in fact proven the following lemma.

LEMMA 18. *Let* $\mathbf{d} = d_1, d_2, \ldots, d_n$ *be a degree sequence with maximum degree* $\Delta = o(n^{1/2})$ *and minimum degree* δ *such that* $\Delta/\delta < \theta$ *for some constant* $\theta > 0$. *Let* G *be chosen randomly from the set of* simple *graphs with degree sequence* $\mathbf{d}$. *Let* $0 < c < 1$ *be an arbitrary constant and* $\mathcal{G}$ *be the set of vertex induced subgraphs* H *of* G *which have degree at least* $c\delta$. *Let* $K > 0$ *be an arbitrary constant. Then with probability* $1 - O(n^{-K})$ every *graph* H *in* $\mathcal{G}$ *has a second eigenvalue at most* $\gamma/\sqrt{\Delta}$ *where* $\gamma = \gamma(\theta, c, K)$.

Proof. We can handle "small pairs" by using multigraphs and pass to simple graphs as above. We observe that only the failure probability (41) now needs to be

inflated by $2^{n+O(d^2)}$ and this is handled by making ξ small enough or γ large enough. The case of "large" pairs is handled as before by deducing it from what happens in G. □

There are no lower bounds explicitly stated for δ, but our results are not useful for small minimum degree. It follows from (40) that γ is at least $4\theta^9 \exp\{192\theta^9\}$. Thus say for $\delta \le 10^6$ we will have $\gamma \ge \Delta$ and so the estimate for the second eigenvalue will exceed one, the largest eigenvalue.

Acknowledgment. It is a pleasure to acknowledge the work of the anonymous reviewers. In particular, we would like to thank the referee who pointed out a small gap in the originally submitted version of the paper.

REFERENCES

[1] N. Alon and J. H. Spencer, *The Probabilistic Method*, John Wiley and Sons, New York, 1992.
[2] E. A. Bender and E. R. Canfield, *The asymptotic number of labelled graphs with given degree sequences*, J. Combin. Theory Ser. A, 24 (1978), pp. 296–307.
[3] B. Bollobás, *A probabilistic proof of an asymptotic formula for the number of labelled regular graphs*, European J. Combin., 1 (1980), pp. 311–316.
[4] B. Bollobás, *Martingales, isoperimetric inequalities and random graphs*, Colloq. Math. Soc. János Bolyai, 52 (1987), pp. 113–139.
[5] B. Bollobás, T. I. Fenner, and A. Frieze, *Hamilton cycles in random graphs with minimal degree at least k*, in "A Tribute to Paul Erdös," A. Baker, B. Bollobás, and A. Hajnal, eds., Cambridge University Press, Cambridge, England, 1990, pp. 59–96.
[6] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North–Holland, Amsterdam, 1976.
[7] A. Z. Broder, A. M. Frieze, S. Suen, and E. Upfal, *An efficient algorithm for the vertex-disjoint paths problem in random graphs*, in Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms, Atlanta, GA, Jan. 1996, pp. 261–268.
[8] A. Z. Broder, A. M. Frieze, and E. Upfal, *Existence and construction of edge-disjoint paths on expander graphs*, SIAM J. Comput., 23 (1994), pp. 976–989.
[9] P. Erdös and L. Lovász, *Problems and results on 3-chromatic hypergraphs and some related questions*, in Infinite and Finite Sets, A. Hajnal et al., eds., Colloq. Math. Soc. János Bolyai, 11 (1975), pp. 609–627.
[10] J. Friedman, J. Kahn, and E. Szemerédi, *On the second eigenvalue in random regular graphs*, in Proc. 21st Annual ACM Symposium on Theory of Computing, Seattle, WA, May 1989, pp. 587–598.
[11] D. Gale, *A theorem on flows in networks*, Pacific J. Math., 7 (1957), pp. 1073–1082.
[12] S. Hochbaum, *An exact sublinear algorithm for the max flow, vertex-disjoint paths and communication problems on random graphs*, Oper. Res. 40 (1992), pp. 923–935.
[13] J. Kleinberg and É. Tardos, *Approximations for the disjoint paths problem in high-diameter planar networks*, in Proc. 27th Annual ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 26–35.
[14] C. J. H. McDiarmid, *On the method of bounded differences*, in Surveys in Combinatorics, Proc. 12th British Combinatorial Conference, J. Siemons, ed., Vol. 141 of London Math. Soc. Lecture Series, Cambridge University Press, Cambridge, England, 1989, pp. 148–188.
[15] D. Peleg and E. Upfal, *The token distribution problem*, SIAM J. Comput., 18 (1989), pp. 229–243.
[16] N. Robertson and P. D. Seymour, *Graph minors-XIII: The disjoint paths problem*, J. Combinatorial Theory Ser. B, 63 (1995), pp. 65–110.
[17] E. Shamir and E. Upfal, *A fast construction of disjoint paths in networks*, Ann. Discrete Math., 24 (1985), pp. 141–154.
[18] A. Sinclair and M. Jerrum, *Approximate counting, uniform generation and rapidly mixing Markov chains*, Inform. and Comput., 82 (1989), pp. 93–133.
[19] L. G. Valiant and G. J. Brebner, *Universal schemes for parallel communication*, in Proc. 13th Annual ACM Symposium on Theory of Computation, Milwaukee, WI, May 1981, pp. 263–277.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 574–611

FULLY DYNAMIC ALGORITHMS FOR BIN PACKING: BEING (MOSTLY) MYOPIC HELPS*

ZORAN IVKOVIĆ† AND ERROL L. LLOYD‡

Abstract. The problem of maintaining an approximate solution for *one-dimensional bin packing* when items may arrive and depart dynamically is studied. In accordance with various work on fully dynamic algorithms, and in contrast to prior work on bin packing, it is assumed that the packing may be arbitrarily rearranged to accommodate arriving and departing items. In this context our main result is a *fully dynamic* approximation algorithm for bin packing *MMP* that is $\frac{5}{4}$-competitive and requires $\Theta(\log n)$ time per operation (i.e., for an *Insert* or a *Delete* of an item). This competitive ratio of $\frac{5}{4}$ is nearly as good as that of the best practical off-line algorithms. Our algorithm utilizes the technique (introduced here) whereby the packing of an item is done with a total disregard for already packed items of a smaller size. This *myopic* packing of an item may then cause several smaller items to be repacked (in a similar fashion). With a bit of additional sophistication to avoid certain "bad" cases, the number of items (either individual items or "bundles" of very small items treated as a whole) that needs to be repacked is bounded by a constant.

Key words. bin packing, fully dynamic algorithm

AMS subject classifications. 68P05, 68Q25, 68R05

PII. S0097539794276749

1. Introduction.

In the (one-dimensional) bin packing problem, a list $L = (a_1, a_2, ..., a_n)$ of items of size $size(a_i)$ in the interval (0,1] is given. The goal is to find the minimum k such that all of the items a_i can be packed into k unit-size bins. Bin packing was shown to be *NP*-complete in [15].

For the past quarter century, bin packing has been a central area of research activity in the algorithms and operations research communities (see [3, 7]). Despite its advanced age, bin packing has retained its appeal by being a fertile ground for the study of approximation algorithms (more than a decade ago, bin packing was labeled "the problem that wouldn't go away" [3]). In this paper, we consider *fully dynamic* bin packing, where

- items may arrive and depart from the packing dynamically, and
- items may be moved from bin to bin as the packing is adjusted to accommodate arriving and departing items.

In general, *fully dynamic* algorithms are aimed at situations where the problem instance is changing over time. Fully dynamic algorithms incorporate these incremental changes without any knowledge of the existence and nature of future changes.

Each of the earlier works on *on-line* and *dynamic* bin packing differ from this notion of fully dynamic bin packing in either of two ways: either they do not allow an item to be moved from a bin (of course, this has a predictably bad effect on the achievable quality of the packing), or they restrict themselves to dynamic arrivals of items—there are no departures.

*Received by the editors November 7, 1994; accepted for publication (in revised form) September 10, 1996; published electronically July 28, 1998. This research was partially supported by National Science Foundation grant CCR-9120731.

http://www.siam.org/journals/sicomp/28-2/27674.html

†School of Management, Yale University, 135 Prospect Street, New Haven, CT 06520 (ivkovich@isis.som.yale.edu).

‡Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716 (elloyd@cis.udel.edu).

Although most of the existing work on fully dynamic algorithms has been directed toward problems known to be in P, some recent attention has been paid to fully dynamic *approximation* algorithms for problems that are *NP*-complete [9, 16].

In this paper we develop a fully dynamic approximation algorithm for bin packing that is "competitive" with existing off-line algorithms. In this case, being competitive with off-line algorithms means that the quality of the approximation produced by the fully dynamic approximation algorithm should be as good as that produced by the off-line algorithms. Further, the running time per operation (i.e., change) of the fully dynamic algorithm should be as small as possible.

1.1. Bin packing—Existing results. The usual measure of the quality of a solution produced by a bin packing algorithm A is its *competitive ratio* $R(A)$ defined as

$$R(A) = \lim_{n\to\infty} \sup_{OPT(L)=n} \frac{A(L)}{OPT(L)},$$

where $A(L)$ and $OPT(L)$ denote, respectively, the number of bins used for packing of the list L by A and some optimal packing of L. Here, we say that A is $R(A)$-*competitive*.

In the domain of off-line algorithms, the value of R has been successively improved [3, 19, 5, 13, 4]. Indeed, it has been shown that for any value of $R \geq 1$, there is an $\mathcal{O}(n \log n)$-time algorithm with that competitive ratio [14]. Unfortunately, the running times for these algorithms involve exceedingly large constants (actually, these "constants" depend on how close R is to 1). Among algorithms of practical importance, the best result is an $\mathcal{O}(n \log n)$ algorithm for which R is $\frac{71}{60}$ [13].

With respect to on-line bin packing, the problem has been defined strictly in terms of arrivals (*Inserts*)—items never depart from the packing (i.e., there are no *Deletes*). Further, most on-line algorithms have operated under the restriction that each item must be packed into some bin, and it should remain in that bin permanently. In this context, it is shown that for every on-line linear time algorithm A, $R(A) \geq 1.536...$ [3]. Further, the upper bound has been improved over the years to roughly 1.6 [10, 11, 12, 17, 18].

The work reported in [6] focused on a variant of on-line bin packing, again supporting *Inserts* only, in which each item may be moved a constant number of times (from one bin to another). Two algorithms were provided: one with a linear running time (linear in n, the number of *Inserts*, which is also the number of items) and a competitive ratio of 1.5, and one with an $\mathcal{O}(n \log n)$ running time and a competitive ratio of $\frac{4}{3}$.

Another notion that is related to, but distinct from, fully dynamic bin packing is dynamic bin packing of [2], where each item is associated with not only its size, but also with an arrival time and a departure time (interpreted in the natural way). Here, again (and unlike [6]), items cannot be moved once they are assigned to some bin, unless they depart from the system permanently (at their departure time). This variant differs from fully dynamic bin packing in that items are not allowed to be moved once they are assigned to a bin and through the departure time information. It was shown in [2] that for any such algorithm A, $R(A) \geq 2.5$, and that for their dynamic first fit (FF), $2.770 \leq R(FF) \leq 2.898$.

1.2. Competitive ratio and running time for fully dynamic approximation algorithms. In this section we discuss the notions of competitiveness and running time in the context of developing fully dynamic approximation algorithms.

We begin by noting that with respect to the definition of *competitive ratio* there is no need to make a distinction between fully dynamic and off-line algorithms. In each case, these measures reflect the size of the packing produced by the algorithm relative to the size of optimal packing.

With respect to running times, we say that a fully dynamic approximation algorithm B for bin packing has *running time* $\mathcal{O}(f(n))$ if the time taken by B to process a change (an *Insert* or *Delete*) to an instance of n items is $\mathcal{O}(f(n))$. If $\mathcal{O}(f(n))$ is a *worst-case* time bound, then B is *uniform*. If $\mathcal{O}(f(n))$ is an *amortized* time bound, then B is *amortized*.

Our general goal in developing fully dynamic approximation algorithms for bin packing is to design algorithms with competitive ratios close to those of the best off-line algorithms such that the changes are processed quickly. Of particular interest are algorithms that are, in a sense, the best possible relative to the existing off-line methods. For bin packing the best known off-line algorithms require time $\Theta(n \log n)$. Thus, a fully dynamic algorithm that runs in time $\Theta(\log n)$ per operation is, in that sense, the best possible. Indeed, the fully dynamic algorithm *MMP* that we give in this paper runs in precisely this time per operation.

The algorithms that we present process a sequence of *Inserts* (arrivals) and *Deletes* (departures) of items. Further, our algorithm is designed to handle "lookup" queries of the following form:

- *size*—returns in $\mathcal{O}(1)$ time the number of bins in the current packing;
- *packing*—returns a description of the packing in the form of a list of pairs $(x, \mathrm{Bin}(x))$, where $\mathrm{Bin}(x)$ denotes the bin into which an item x is packed, in time linear in the number of items in the current instance.

These queries may be interspersed in the *Insert/Delete* sequence as follows.

1.3. What's to come. The main result of this paper is a fully dynamic algorithm *MMP* that is $\frac{5}{4}$-competitive and requires $\Theta(\log n)$ time per operation. Relative to the best off-line algorithms, *MMP* has a running time that is best possible, and it has a competitive ratio that is nearly the equal of the best practical off-line algorithms. This is a surprising result even in terms of off-line bin packing, since it is the first practical bin packing algorithm that has a competitive ratio of less than $\frac{4}{3}$ that does not rely on packing the items in sorted order (as discussed in section 2, dynamically maintaining a packing based on a sorted list is problematic). That the algorithm is fully dynamic is all the more remarkable.

With the preliminaries concluded, the remainder of the paper is organized as follows. In the next section we review the basic definitions and define two key properties of *MMP* packing: *LLS-maximality* and *M-thoroughness*. We further provide a sketch of *MMP*'s *Insert* and *Delete* operations, and we focus on the techniques *MMP* utilizes to maintain the above properties: *myopic packing*, *bundles*, and *LLS-coalitions*.

In section 3 we describe the (rather complex) data structure, and the details of *MMP*. In section 4 we prove the competitive ratio of $\frac{5}{4}$ and the uniform logarithmic running time per *Insert/Delete* operation. Finally, in section 5 we furnish some concluding remarks.

2. Toward full dynamization of bin packing. Motivated by the notions of competitiveness introduced in the preceding section, a natural approach to the development of fully dynamic bin packing algorithms is to adapt existing methods to work in the fully dynamic situation. Unfortunately, this is easier said than done. The difficulty is that most of the off-line algorithms perform bin packing in two distinct stages. First, there is a *preprocessing stage* in which the items are organized in some

fashion (this reorganization should have a positive effect on the resulting packing). This is followed by a *packing stage* where the actual packing is accomplished. In the off-line situation, this two stage approach is quite natural since the entire list of items is available at the outset. However, in a dynamic environment a two-stage process becomes awkward. Consider, for example, the algorithm first fit decreasing (*FFD*), which is $\frac{11}{9}$-competitive. This algorithm first sorts the items and then packs them in order of decreasing size using the *FF* packing rule.[1] What about a fully dynamic version of *FFD*? There is, of course, no difficulty in maintaining a sorted list of the elements. *But* there is great difficulty in maintaining the packing based on that sorted list, since the insertion (or deletion) of a single item can result in a large number of changes to that packing. It would seem that the packing induced by the sorted list of items is "too specific" to be maintained dynamically, and that perhaps a less specific rule might be of use. Indeed, in this paper we utilize a weaker notion: Johnson's *grouping* [10, 11].

2.1. Some definitions. Before proceeding, we require a few definitions that will be used throughout the remainder of the paper. In particular, for a bin B, $level(B)$ is the sum of the sizes of the items packed in B; $gap(B)$ is $1 - level(B)$, i.e., the amount of empty space in B; and $content(B)$ is the set of items packed in B.

We can assume that the bins are numbered in such a way that every bin has a unique number with the property that, for any two bins, the bin with the lower number is placed "to the left" of the bin with the higher number. In other words, we assume that, for conceptual purposes, the bins are numbered in increasing order from left to right.

Following Johnson's *grouping* [10, 11], we partition the items according to their respective sizes. In particular, an item a is a B-item (big) if $size(a) \in (\frac{1}{2}, 1]$, an L-item (large) if $size(a) \in (\frac{1}{3}, \frac{1}{2}]$, an S-item (small) if $size(a) \in (\frac{1}{4}, \frac{1}{3}]$, a T-item (tiny) if $size(a) \in (\frac{1}{5}, \frac{1}{4}]$, and an M-item (miniscule) if $size(a) \in (0, \frac{1}{5}]$.

Let $\mathcal{B}$, $\mathcal{L}$, $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{M}$ denote the number of B-items, L-items, S-items, T-items, and M-items in L, respectively.

When the meaning is otherwise clear, the fact that a is a B-item (L-item, S-item, T-item, M-item) will be abbreviated as $a \in$ B (L,S,T,M). A bin is a B-bin (L-bin, S-bin, T-bin, M-bin) if its largest item is a B-item (L-item, S-item, T-item, M-item). There are several types of B-bins: bins containing one B-item and one L-item, and no more B-items, L-items, S-items, or T-items will be called bins of type BL; bins of type BST, BS, BTT, BT, and B are defined analogously. Likewise, there are several types of L-bins, several types of S-bins, and several types of T-bins. The possible types of B-bins, L-bins, S-bins, and T-bins are illustrated in Figure 1. Note that we did not take into consideration the M-items: while it is certainly the case that bins may contain M-items, accounting for them will have no substantive effect on the competitive ratio of *MMP*.

By way of preliminaries, we introduce a binary relation of *superiority* over types of bins. First, all of the types of B-bins, L-bins, S-bins, and T-bins are superior to M-bins. Second, we consider non-M-bins. Here the following ordering of relevant types of items is assumed: B $\prec$ L $\prec$ S $\prec$ T $\prec$ Z, where Z denotes a fictitious item of size 0. We imagine that each bin contains, on top of its B-items, L-items, S-items, and T-items (M-items may be present but are being ignored), a fictitious item of type

[1] Informally, bins are ordered from left to right, and an item is packed into the leftmost bin into which it will fit.

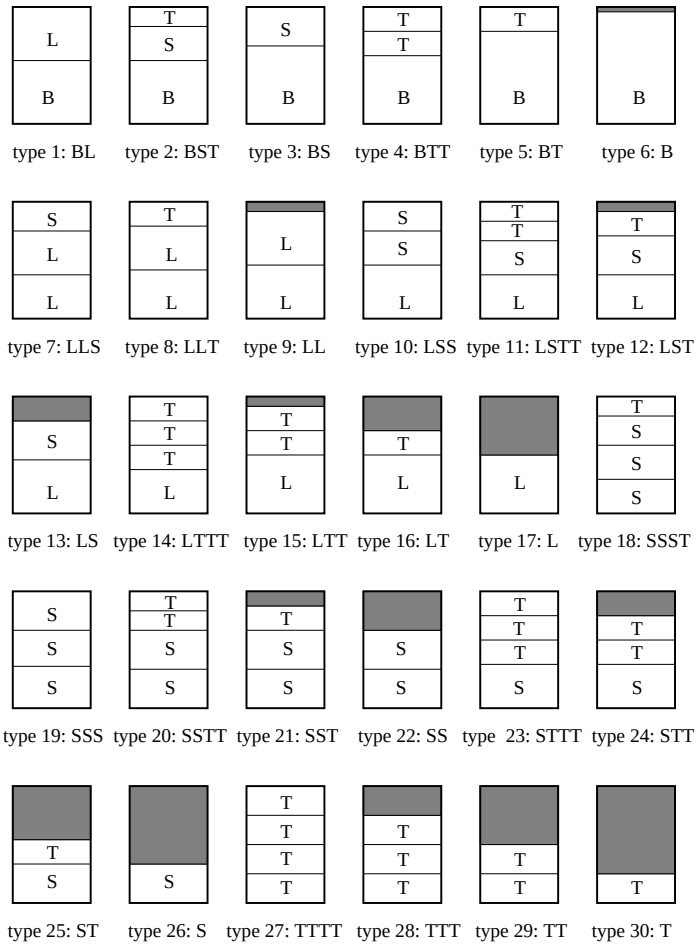


FIG. 1. *Possible types of bins in* MMP.

Z (zero), of size 0. Zero items are introduced solely for technical convenience, as their presence will enable us to impose the desired ordering on the types of bins. Thus, in view of the introduction of Z-items, the types of bins are BLZ, BSTZ, BSZ, ..., TTZ, and TZ. For these types of bins, the relation of superiority is defined as the lexicographical ordering over the types of bins. For example, a bin of type BLZ is superior to a bin of type BSTZ. In the remainder of this paper, we omit Z from the notation describing the types of bins.[2] Finally, we will sometimes find it convenient to refer to these types of bins according to their canonical index in this lexicographical ordering, as depicted in Figure 1: a bin of type 1 is a bin of type BL, a bin of type 2 is a bin of type BST, ..., a bin of type 30 is a bin of type T. We assert naturally that if B_j is superior to B_i, then B_i is *inferior* to B_j.

The *allowed* types of bins in the packings produced by *MMP* are BL, BST, BS, BTT, BT, B, LLS, LLT, LL, SSST, SSS, and TTTT, and, of course, M-bins. This restriction may result in at most six unpacked items: one L-item, two S-items, and

[2]Although Z is omitted, it is needed to ensure that, e.g., BLZ is superior to BZ. The reader should keep in mind that the relation of superiority relies on the presence of Z in each non-M-bin.

three T-items. Clearly, these items could be packed into at most two additional bins (a bin of type LTT and a bin of type SST). *MMP* will utilize the *regular packing*, consisting at all times only of bins of the allowed types, and the *auxiliary storage*, containing the items that are not (currently) packed into a bin from the regular packing.

2.2. LLS-maximality and M-thoroughness. We next define the properties of packings that play a key role for the competitive ratio of *MMP*. We first define the *thoroughness property*. Next, we define the *LLS-maximality property*, a property that is similar to, and (much) stronger than, the thoroughness property. Finally, we define the *M-thoroughness property* aimed at the M-items and their role in the packing. Intuitively, maintaining the LLS-maximality property leads to the competitive ratio of $\frac{5}{4}$ for packings of lists of non-M-items; maintaining LLS-maximality *and* the M-thoroughness property leads to the competitive ratio of $\frac{5}{4}$ for packings of arbitrary lists. We begin with two definitions.

DEFINITION 1. *Let $\mathcal{P}_{B,L,S,T}$ be a set of packings of B-items, L-items, S-items, and T-items such that each packing $P \in \mathcal{P}_{B,L,S,T}$ consists only of the allowed types of bins (BL, BST, BS, BTT, BT, B, LLS, LLT, LL, SSST, SSS, and TTTT), where all of the bins of type BL are to the left of all the non-BL-bins, all of the bins of type BST are to the left of all the non-BL-bins and non-BST-bins, etc.*

DEFINITION 2. *Let a packing $P \in \mathcal{P}_{B,L,S,T}$. Then*

1. *Bins of type BL are thorough in P iff there does not exist a B-item b and an L-item l such that $size(b) + size(l) \leq 1$, and item b is either in a bin of type inferior to BL in the packing P or in the auxiliary storage, and item l is either in a bin of type inferior to BL in the packing P or in the auxiliary storage, i.e., iff it is not possible to pack a B-item from a bin of type inferior to BL or from the auxiliary storage, and an L-item from a bin of type inferior to BL or from the auxiliary storage into a bin.*
2. *Bins of type BST are thorough in P iff there does not exist a bin B of type BS in P, where b and s are the B-item and the S-item packed into B, and a T-item t such that $size(b) + size(s) + size(t) \leq 1$, and the item t is in a bin of type inferior to BST in the packing P or the auxiliary storage.*
3. *Bins of type BS are thorough in P iff there does not exist a B-item b and an S-item s such that $size(b) + size(s) \leq 1$, and the item b is either in a bin of type inferior to BS in the packing P or in the auxiliary storage, and the item s is in a bin of type inferior to BS in the packing P or in the auxiliary storage.*
4. *Bins of type BTT are thorough in P iff there does not exist a bin B of type BT in P, where b and t_1 are the B-item and the T-item packed into B, and a T-item t_2 such that $size(b) + size(t_1) + size(t_2) \leq 1$, and the item t_2 is in a bin of type inferior to BTT in the packing P or in the auxiliary storage.*
5. *Bins of type BT are thorough in P iff there does not exist a B-item b and a T-item t such that $size(b) + size(t) \leq 1$, and the item b is either in a bin of type inferior to BS in the packing P or in the auxiliary storage, and the item t is in a bin of type inferior to BT in the packing P or in the auxiliary storage.*
6. *Bins of type LLS are thorough in P iff there does not exist a bin B of type LLT or LL in P, where l_1 and l_2 are the L-items packed into B, and an S-item s such that $size(l_1) + size(l_2) + size(s) \leq 1$, and the item s is either in a bin of type inferior to LLS in the packing P or in the auxiliary storage.*

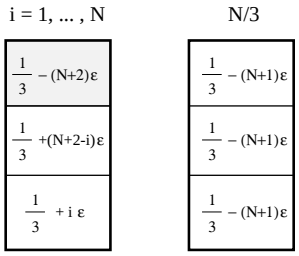


FIG. 2. *An example of a thorough but not LLS-maximal packing. In the packing above, N is an arbitrary integer. The bottom L-item a_1 from the first bin ($i = 1$), $size(a_1) = \frac{1}{3} + \epsilon$, and the top L-item a_2 from the second bin ($i = 2$), $size(a_2) = \frac{1}{3} + N\epsilon$ can fit together with an S-item (all S-items have the size of $\frac{1}{3} - (N+1)\epsilon$). The same is true of the bottom L-item from the second bin ($i = 2$) and the top L-item from the third bin ($i = 3$), ..., the bottom L-item from the $(N-1)$st bin and the top L-item from the Nth bin. Thus, although the packing above is thorough, it is far from LLS-maximal, since many bins of type LLS could be packed from the items in the packing, and all of the items are packed into bins of type inferior to LLS.*

7. *Bins of type LLT are thorough in P iff there does not exist a bin B of type LL in P, where l_1 and l_2 are the L-items packed into B, and a T-item t such that $size(l_1) + size(l_2) + size(t) \leq 1$, and the item t is either in a bin of type inferior to LLT in the packing P or in the auxiliary storage.*
8. *Bins of type SSST are thorough in P iff there does not exist a bin B of type SSS in P, where s_1, s_2, and s_3 are the S-items packed into B, and a T-item t such that $size(s_1) + size(s_2) + size(s_3) + size(t) \leq 1$, and the item t is either in a bin of type inferior to SSST in the packing P or in the auxiliary storage.*

Finally, a packing $P \in \mathcal{P}_{B,L,S,T}$ is thorough iff all of the above types of bins are thorough in P.

LLS-maximality. *MMP* will take some pains to be guaranteed of packing a certain portion of certain L-items and S-items into bins of type LLS (we will call this endeavor "seeking *LLS-coalitions*"). Leading toward this guarantee, we define *LLS-maximality*. LLS-maximality strengthens thoroughness: maintenance of thoroughness does not require LLS-coalitions, and the absence of coalitions leads to a competitive ratio of at least $\frac{4}{3}$ (see the lower bound example for *FFG* in [10]).

DEFINITION 3. *Let a packing $P \in \mathcal{P}_{B,L,S,T}$. Then P is LLS-maximal iff P is thorough and bins of type LLS are LLS-maximal in P; i.e., there does not exist an L-item l_1, another L-item l_2, and an S-item s such that $size(l_1)+size(l_2)+size(s) \leq 1$, and the item l_1 is either in a bin of type inferior to LLS in the packing P or in the auxiliary storage, and the item l_2 is in a bin of type inferior to LLS in the packing P or in the auxiliary storage, and s is in a bin of type inferior to LLS in the packing P or in the auxiliary storage.*

Note that LLS-maximality is a (much) stronger property than thoroughness: there might be packings that are thorough but not maximal. In Figure 2 we give an example, a variant of the lower bound example for *FFG* from [10], of a packing that is thorough but not maximal.

The key factor that distinguishes between thoroughness and LLS-maximality is that when considering whether or not it is possible to pack two L-items and an S-item from bins of type inferior to LLS or the auxiliary storage into a bin, LLS-maximality, unlike thoroughness, does *not* insist that the two L-items must come from the same bin. We note that it can be shown that the maintenance of thoroughness, but not

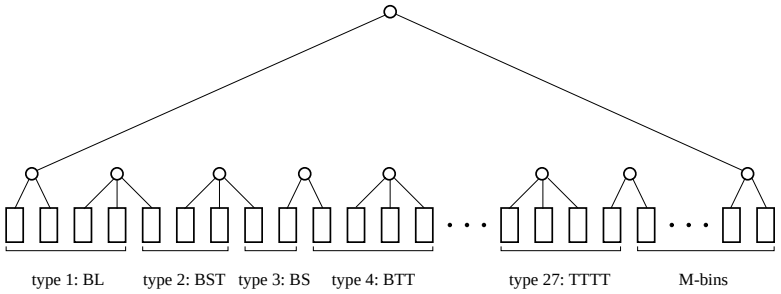


FIG. 3. *A sketch of the* 2-3 *tree of bins in* MMP. *Note that it contains only the allowed types of bins.*

LLS-maximality, leads to a simpler algorithm that also runs in uniform logarithmic time per *Insert/Delete* operation, and is $\frac{4}{3}$–competitive (see [8]).

M-thoroughness. M-thoroughness is the third property we require. It pertains to the role of M-items in the *MMP* packings. Ideally, we would like to be able to develop a method that would enable *MMP* to pack as many M-items into non-M-bins as possible. However, this is not necessary, as it turns out that maintaining M-thoroughness (a much weaker goal), coupled with LLS-maximality, of course, is quite sufficient to guarantee a competitive ratio of $\frac{5}{4}$. Later in this section we show that *MMP* maintains the M-thoroughness property.

DEFINITION 4. *A packing P is M-thorough iff precisely one of the following two conditions is satisfied:*

1. *there are no M-bins in P, or*
2. *there is at least one M-bin in P, and all of the non-M-bins have a level exceeding* $\frac{4}{5}$ *(i.e., a gap less than* $\frac{1}{5}$*), and all of the M-bins, except for possibly the rightmost bin in the packing, also have a level exceeding* $\frac{4}{5}$.

2.3. A sketch of *Insert* and *Delete*. In the next section we describe the data structure of *MMP* in detail. Here we briefly note that all of the bins in the packing will be stored at the leaves of the 2-3 tree of bins, with the bins of type BL placed in the leftmost leaves of the 2-3 tree of bins, with the bins of type BST placed in the leftmost remaining leaves (those not holding bins of type BL) of the 2-3 tree of bins, etc. for all of the other allowed types of bins, and, finally, with the M-bins placed in the rightmost leaves of the 2-3 tree of bins,[3] as depicted in Figure 3.

We now consider, somewhat informally, how to *Insert/Delete* an item. This is done using three major ideas: *myopic packing*, *bundles*, and *LLS-coalitions*.

***Insert* and *Delete* of non-M-items.** We first consider how *MMP Insert*s an item $a \in \mathrm{B} \cup \mathrm{L} \cup \mathrm{S}$. We begin by explaining how *myopic packing* is used to maintain thoroughness. Based on Johnson's grouping [10, 11], when an item a is being packed, a should be more insensitive to previously packed items of "smaller" types than the type of a. Thus, what would a "see" in the bins? Only the items of its own type or of "larger" type. In this sense, a K-item (K is B, L, S, or T) is *myopic* in that it can "see" relatively large items (K-items or larger), and it cannot "see" relatively small ones (smaller than K-items). Based on that view of the packing, a is packed in an *FF*

[3]Recall that the bins are numbered in increasing order from left to right. A numbering of bins that ranges $1, \ldots, MMP(L)$ corresponds to the graph theoretic notion of a preorder numbering of leaves of the 2-3 tree of bins.

fashion (in a's "K or larger" world) using the information stored in the internal nodes and leaves of the 2-3 tree of bins.[4] Let B be the bin into which a was packed. This packing of a results in a forceful eviction of items of smaller types from B, if there are such items at all. The evicted items will be temporarily "set aside" into the auxiliary storage and will eventually be reinserted. Next, an attempt is made to restore the thoroughness of the packing by trying to pack additional items into B, starting from the available items of the largest type that are smaller than K, i.e., the items of the largest type that are smaller than K from the auxiliary storage and from the bins that are inferior to the type of bin the algorithm is trying to reconstruct for B. This effort continues until there are no more available items of that type that can fit with the current bin content. Next, *MMP* continues with the available items (auxiliary storage or inferior bins) of the next largest type, until there are no more available items of that type that would fit into the bin, etc. Here, if an item is taken from some bin, that bin is deleted from the packing, and its contents, except for the item that was taken, are temporarily moved into the auxiliary storage. Upon completing the filling of B and inserting B into the packing, *MMP* reinserts the items from the auxiliary storage into the packing. Their reinsertion may, of course, disturb some other bins and move their contents to the auxiliary storage for later reinsertion. Eventually, all of the items from the auxiliary storage (except perhaps at most one L-item, two S-items, and/or three T-items, of course) are reinserted into the packing, and that packing is thorough.

In addition to thoroughness, *MMP* maintains LLS-maximality. This is done by using LLS-coalitions. To avoid the situation of a list that is thorough but far from LLS-maximal (see Figure 2), an amendment is made to the myopic discipline outlined above. Namely, the insertion of an L-item a is carried out as follows: first, packing a into a B-bin is attempted in a standard myopic fashion. If this attempt fails, a is authorized to try to form an *LLS-coalition* with another L-item and an S-item. The latter two can each be sought in any, and not necessarily the same, bin whose type is inferior to LLS or in the auxiliary storage. If such a coalition is possible, a and the two items are packed into a bin of type LLS, and that bin is inserted into the regular packing. The bins that yielded some or all of these two items need to be deleted, and their remaining content will eventually be reinserted. If the coalition is not possible, the packing of a is completed by resuming the standard myopic steps. Similarly, insertion of an S-item a would involve first packing a into a B-bin in a standard myopic fashion. If this fails, a will seek two L-items coming from any, and not necessarily the same, bins whose type is inferior to LLS or from the auxiliary storage. If two such items are found, an LLS-coalition is formed, and the bin of type LLS is inserted into the regular packing. If not, the packing of a is completed by resuming the standard myopic steps. A careful implementation can guarantee that the added complexity of this *mostly myopic* discipline does not asymptotically add to the running time.

Deletes are implemented as follows: the bin in which a (the item that needs to be deleted) resides is emptied and is deleted. Upon discarding a, the remaining contents of the deleted bin are temporarily moved to the auxiliary storage, from which they are reinserted into the packing as a part of this *Delete* operation.

We show later that *Inserts* and *Deletes* can be carried out in $\Theta(\log n)$ uniform running time, since the number of bins inserted and deleted by an *Insert/Delete*

[4]Note that we do not provide for all the details here. A more detailed description of the data structure and the algorithm *MMP* will be furnished in the coming sections.

operation is bounded by a fixed constant. Intuitively, the discipline of "touching" only the inferior types of bins provides for the desired running time.

Handling M-items. We now consider how *MMP* packs the lists that contain M-items. The goal here is to utilize *bundles* to manipulate many M-items at once within logarithmic uniform running time. At the same time, a proper manipulation of M-items will be important for the M-thoroughness property.

In general, the simplest approach would be to pack the M-items independently of the B-items, L-items, S-items, and T-items by packing them into totally separate bins. This would, however, lead to a competitive ratio greater than $\frac{5}{4}$. Rather, the M-items need to be packed, whenever possible, into non-M-bins. Thus, *MMP* inserts M-items just like any other items, according to their myopic view of the packing (of course, they actually "see" the entire packing). However, the presence of M-items in the packing gives rise to several important considerations.

First, upon insertion of an M-item a into a bin no items will be evicted—M-items are the smallest items! This makes the insertion of an M-item very efficient.

Second, the insertion of B-items, L-items, S-items, and T-items in situations where the input lists contain M-items needs to be examined very carefully. In particular, if the algorithm were to follow only the simple logic of myopic packing, its striving to maintain a thorough packing might require relocation of as many as $\mathcal{O}(n)$ M-items, leading to $\mathcal{O}(n \log n)$ time per *Insert/Delete* operation. This would happen during both insertions and deletions that require relocation of items from the bins of type inferior to that of the bin that is currently being filled. Furthermore, the number of bins that could be inserted and deleted per operation would be huge: it would be possible, for example, to delete as many as $\mathcal{O}(n)$ bins of type BST for the sake of taking a few M-items from each of them and packing those M-items into a single bin of type BL. The disaster does not stop here: each of the items from those many bins of type BST needs to be reinserted, and each reinsertion may again cause an avalanche of deleted bins.

Third, in case the simple myopic discipline is followed, the deletion of an M-item would cause the temporary relocation of B-items, L-items, S-items, T-items, and potentially many M-items into the auxiliary storage. Packing all of these items back into the bins might be very costly: following the same argument as above, $\mathcal{O}(n \log n)$ time might be required to reinsert a single non-M-item, with many inserted and deleted bins.

Thus handling M-items in the same manner as the other items will not do. We solve this apparent difficulty by introducing the technique of *bundling* (see [1]). The idea is that the M-items in each bin (in the auxiliary storage) are collected into *bundles* g_i. All of the bundles in a bin (in the auxiliary storage) have the cumulative size of $\frac{1}{10} < size(g_i) \leq \frac{1}{5}$, except for at most one bundle whose cumulative size is $\leq \frac{1}{10}$. The former kind of bundles is called *closed*, while the latter kind is called *open*.

The purpose of bundles is to allow efficient manipulation of large numbers of M-items at one time: in response to the need to move M-items from a bin to the auxiliary storage, or from the auxiliary storage to a bin, the algorithm will only move entire bundles. During this process, when a bundle is inserted into a bin (or temporarily stored into auxiliary storage), it is first checked to see whether it could be merged with the open bundle, if any, from that bin (or from the auxiliary storage), and, if so, the merging is carried out. While this does not asymptotically increase the running time required for the insertion of an M-item, it drastically decreases the running time of other operations involving M-bundles and makes *MMP* fast ($\Theta(\log n)$ running time per *Insert/Delete* operation).

The algorithm will treat bundles of M-items like any other item (except for the occasional merging of bundles to maintain the property that each bin (auxiliary storage) can have at most one open bundle). Note that a bin can contain at most 10 bundles; hence, we say that no bin can contain more than 10 items. Bundling is one of the tools used to accomplish *M-thoroughness*. It is natural to ask whether or not the technique of bundling is essential for *MMP*; the answer is in the affirmative, since it can be shown that moving only a constant number of very small items per *Insert/Delete* operation disallows competitive ratios below $\frac{4}{3}$, regardless of the running time (see [8]).

3. The data structure and the details of *MMP*.

3.1. The data structure of *MMP*. In this section we describe the data structure utilized by *MMP*. The data structure is rather complex, and it consists of several components.

1. *The regular packing*. The regular packing consists of the bins of the allowed types and some of the information required for the maintenance of an *MMP* packing (LLS-maximal and M-thorough). As mentioned before, the regular packing is maintained via a 2-3 tree. The leaves represent the bins, while the internal nodes store some of the information required for the proper maintenance of the packing (LLS-maximal and M-thorough).

Each leaf contains a record with the following information that provides for a full description of a bin B:

- *content*(B)—Five doubly linked circular lists are utilized to record the set of items currently packed in B, one list for the B-items, one for the L-items, one for the S-items, one for the T-items, and one for the M-bundles packed in B. The entries of these lists are the individual records associated with the B-items, L-items, S-items, T-items, and M-bundles that are currently packed in B. Recall that each bin can contain at most one B-item, two L-items, three S-items, four T-items, and ten M-bundles (at most nine closed bundles and at most one open bundle). Thus these lists are very short.
- Each leaf contains a pointer to the open bundle in B, if any.
- Five numbers that record the myopic levels of B include the following: $level_{\mathrm{B}}(B) = \sum_{a \in content(B) \wedge a \in \mathrm{B}} size(a)$, $level_{\mathrm{L}}(B) = \sum_{a \in content(B) \wedge a \in \mathrm{B} \cup \mathrm{L}} size(a)$, $level_{\mathrm{S}}(B) = \sum_{a \in content(B) \wedge a \in \mathrm{B} \cup \mathrm{L} \cup \mathrm{S}} size(a)$, $level_{\mathrm{T}}(B) = \sum_{a \in content(B) \wedge a \in \mathrm{B} \cup \mathrm{L} \cup \mathrm{S} \cup \mathrm{T}} size(a)$, and $level_{\mathrm{M}}(B) = level(B)$.
- Each leaf contains the type of B.

Each internal node contains the following information about its left, middle, and right (if present) subtree: (1) the largest gaps in the subtree ($gap_{\mathrm{B}} = 1 - level_{\mathrm{B}}$, $gap_{\mathrm{L}} = 1 - level_{\mathrm{L}}$, $gap_{\mathrm{S}} = 1 - level_{\mathrm{S}}$, $gap_{\mathrm{T}} = 1 - level_{\mathrm{T}}$, and $gap_{\mathrm{M}} = 1 - level_{\mathrm{M}}$) and (2) the most inferior type of bin in the subtree.

2. *Inferior trees*. One of the consequences of executing an *Insert* or *Delete* operation is that, in order to maintain a packing that is LLS-maximal, the packing of a bin B may require that some items be removed from bins inferior to B and packed into B. In order to efficiently search for such items in bins inferior to B, *MMP* maintains for each allowed bin type i a set of items of each type of item that appears in a bin of type i. These sets are each represented by a heap (implemented as a 2-3 tree), and we call them the *inferior trees*. To accomplish this efficient search, *MMP* utilizes 15 min-heaps implemented as 2-3 trees: (1) L-items in bins of type BL, (2) L-items in bins of type LLS, (3) L-items in bins of type LLT, (4) L-items in bins of type LL,

(5) S-items in bins of type BST, (6) S-items in bins of type BS, (7) S-items in bins of type LLS, (8) S-items in bins of type SSST, (9) S-items in bins of type SSS, (10) T-items in bins of type BST, (11) T-items in bins of type BTT, (12) T-items in bins of type BT, (13) T-items in bins of type LLT, (14) T-items in bins of type SSST, and (15) T-items in bins of type TTTT.

For example, if the algorithm attempts to pack a B-item into a bin of type BST, it needs to search for an S-item and a T-item from bins whose type is inferior to BST. Thus, an S-item will be searched for in *Aux* (see below) and in the following inferior trees: the tree of S-items in bins of type BS, the tree of S-items in bins of type LLS, the tree of S-items in bins of type SSST, and the tree of S-items in bins of type SSS. Similarly, a T-item will be searched for in *Aux* and in the following inferior trees: the tree of T-items in bins of type BTT, the tree of T-items in bins of type BT, the tree of T-items in bins of type LLT, the tree of T-items in bins of type SSST, and the tree of T-items in bins of type TTTT.

In an inferior tree, each leaf contains an item, or more precisely the record associated with that item, and the internal nodes contain, for left, middle, and (if present) right subtree, the size of the smallest item in the subtree. This enables an easy search for the smallest item in that inferior tree.

3. *Aux*. Admitting only allowed types of bins to the regular packing possibly results in a few excess items that cannot be packed into the regular packing: at most one L-item, two S-items, and three T-items. These items are stored in *Aux*. In addition, in the course of maintaining an *MMP* packing, all of the bins that lose an item(s) a (so that a could be packed into a superior bin), or simply had an item deleted, need to be deleted from the regular packing, and their content, except for the lost/deleted item(s), temporarily stored into *Aux*. The items that are temporarily stored into *Aux* within an operation must all, except for at most one L-item, two S-items, and three T-items, be reinserted into the packing as an integral part of an *Insert/Delete* operation. Later in this section we show that the number of items stored in *Aux* at any time during the execution of *MMP* is bounded by a rather small constant.

Aux is implemented using five min-heaps (again, each heap is implemented as a 2-3 tree), one for each type of item: Aux_{B} for B-items, Aux_{L} for L-items, Aux_{S} for S-items, Aux_{T} for T-items, and Aux_{M} for M-bundles. Each leaf contains an item (bundle), or more precisely the record associated with that item (bundle), and the internal nodes contain, for left, middle, and (if present) right subtree, the size of the smallest item in the subtree. This enables an easy search for the smallest B-item, L-item, S-item, T-item, or M-bundle in *Aux*.

As it turns out, only closed M-bundles will be stored in the 2-3 tree of M-bundles. *Aux* will contain at any time at most one open M-bundle that will be stored separately from the 2-3 tree of (closed) M-bundles.

4. *M-items/bundles*. As mentioned earlier, M-items will be collected in bundles. These bundles may undergo a number of changes in the course of execution of *MMP*: M-items may be inserted to/deleted from bundles; two bundles could be merged into one bundle of M-items; further, there is a need to allow at most one open bundle per bin (at most one open bundle in *Aux*) and to ensure that all of the other bundles in a bin (in *Aux*) are closed. Each M-bundle b will be represented by a 2-3 tree of M-items that are collected into b. With each M-bundle b there will be an associated record that stores detailed information about b.

2-3 trees are particularly suitable for the maintenance and manipulation of bun-

dles. Insertions and deletions of individual M-items into/from a bundle correspond to the operations *Insert* and *Delete*, which are supported by 2-3 trees in logarithmic time. Furthermore, if there is a need to pack a bundle b into a bin B (or store it into *Aux*), b can be easily added onto the list of M-bundles in B (inserted into the 2-3 tree of M-bundles in *Aux* by means of an *Insert* operation on that tree in case b is closed or designated as open and stored into *Aux* in case b is open). If there is a need to merge two bundles, this can be easily accomplished by executing the operation *Union*, also supported by 2-3 trees in logarithmic time, on the two 2-3 trees representing the two bundles.

5. *The list of items in L.* Each item a will have, for the duration of its presence in L (i.e., from operation *Insert(a)* to operation *Delete*(a)), an associated record that maintains detailed information about a ($size(a)$ and a few pointers for manipulation of a in the data structures utilized by *MMP*). In addition, *MMP* will maintain L by storing its items at the leaves of a 2-3 tree of the current items of L. The leaves of that tree are each associated with an item currently in L. Each leaf stores, next to the pointers required for the manipulation of the 2-3 tree of the current items, a unique identifier associated with the item and the pointer to the corresponding record associated with that item.

When an operation *Insert(a)* is initiated, the new item a is inserted into the 2-3 tree of items, and its associated record is created. This is followed by other actions described in subsequent sections. Conversely, the operation *Delete(a)* involves certain actions (described in subsequent sections) and, finally, removes a from the 2-3 tree of the current items and destroys a's associated record. The processing of an *Insert/Delete* operation will typically cause changes to the packing and will affect several items in the packing. None of those changes will, however, have any impact on the 2-3 tree of the current items.

Finally, we note that, in the course of executing an *Insert/Delete* of an item a, the 2-3 tree of current items is used to locate the bin in which a is packed (or to realize that a is in *Aux*) and, furthermore, if a is an M-item, to locate the bundle b that a belongs to. Given an identifier associated with a, $\Theta(\log n)$ running time is required to locate the leaf in the 2-3 tree of current items that is associated with a. Then the pointer to the corresponding record associated with a is followed to gain access to that record within $\mathcal{O}(\log n)$ uniform running time.

3.2. Details of *MMP*. In this section we furnish the details of *MMP*. In the following subsections we will first provide a top level description of *MMP* and then furnish the details of *clear_Aux*, a key function from that description. In the following, b denotes a B-item, l denotes an L-item, s denotes an S-item, t denotes a T-item, and m denotes a M-bundle.

3.2.1. Top level description of *MMP*. We now describe *MMP*. Simply put, both *Insert*(x) and *Delete*(x) rely heavily on the function *clear_Aux*. The idea behind *Insert* is to insert the item x into *Aux* and then let *clear_Aux* complete the insertion of x in a manner that will maintain LLS-maximality and M-thoroughness. The idea behind *Delete* is similar, except that before invoking *clear_Aux*, the *Delete* operation needs to remove x from all of the data structures utilized by *MMP*.

In the description below, we utilize the following functions:

- *store_Aux*(x)—store the item x into *Aux*.
- *store_Aux*(X)—store the items and M-bundles from the set X into *Aux*. Then, for each L-item, S-item, and T-item from X, delete that item from the inferior tree

to which it belonged (if any). The set X will be either all of the content of some bin B or a part of it. In the latter case, adjust the myopic levels of B.

• *locate_structure*(x)—return a pointer to the structure to which x belongs (either a bin B or Aux).

• *terminate*(x,p)—delete x from all of the data structures. A pointer p points to the structure containing x (either a bin B or Aux). If x is an M-item, then delete x from the M-bundle m to which x belongs; m is contained in the structure pointed to by p (either a bin B or Aux). If x is a non-M-item, then (1) delete x from the structure to which p points (either a bin B or Aux), and then (2) if p points to a bin B, then delete x from the inferior tree to which x belongs (if any). At the conclusion of *terminate*, adjust the myopic levels of B, delete x from L, and delete the record associated with x.

• *remove*(B)—delete a bin B from the 2-3 tree representing the regular packing. Then destroy the record associated with B.

• *add*(B)—insert a bin B into the 2-3 tree representing the regular packing. Let TYPE be the type of B. *add* will insert B immediately after the highest numbered bin of type superior or equal to TYPE. This is accomplished via the information stored at the internal nodes (the most inferior type of bin in the left, middle, and right (if present) subtree). *add* starts at the root and walks down the tree, always to the subtree that stores the largest type that is still inferior to TYPE, until a leaf q is reached, or no subtree with a type that is inferior to TYPE could be found. In the latter case, B is the most inferior bin in the packing, and it will be inserted as the rightmost leaf of the 2-3 tree of bins. In the former case, B is inserted immediately to the left of q.

Both *remove* and *add* will update the information in the interior nodes on the path from B to the root of the 2-3 tree representing the regular packing.

• *open_new_bin*(B)—create a record associated with a new, unpacked bin; call it B. Initialize *content*(B) as empty, and B's myopic levels to 0. Leave the type of B unspecified.

• *pack*(B, $x_1, \ldots, x_k$, TYPE)—pack items/bundles $x_1, \ldots, x_k$ into B. Set the type of B to TYPE, if the value of TYPE is one of the allowed bin types (including M-bins). If TYPE is 0, then *pack* does not set the type of B. The latter option will be convenient when packing M-bundles into non-M, nonempty bins. Note that B need not be empty prior to the execution of the *pack* operation.

• *search_myopic*_K(x) (K $\in$ {L, S, T, M}, x is a K-item)—perform a search for the leftmost bin into which x can fit, with a myopic "K or larger" view of the packing. The search utilizes the information on the largest gap_{K} in the left, middle, and right (if present) subtree stored at the internal nodes of the 2-3 tree representing the regular packing and searches that tree in an *FF* fashion. *search_myopic*_K(x) returns a pointer p to the bin that can accommodate x (in the myopic sense). In this case we let B_p refer to that bin. If the value of the pointer p is *null*, then x could not fit into any of the bins in the (current) regular packing.

• *unload_Aux*$_{\mathrm{M}}$(B)—pack the following M-bundles into a bin B for as long as they fit or until Aux_{M} is empty: (1) the open M-bundle from Aux_{M}, (2) one by one, the smallest remaining closed M-bundle from Aux_{M}. *unload_Aux*$_{\mathrm{M}}$ terminates when either an M-bundle from Aux that cannot fit into B is found or Aux_{M} is empty.

• *top_with*_M(B)—fill up a bin B with M-bundles from Aux_{M}. If Aux_{M} is emptied in the process, the rightmost M-bin B_r of the regular packing (if any) is removed from the regular packing, and all of the M-bundles from B_r are inserted into Aux_{M}, at which point filling B with M-bundles from Aux_{M} is resumed.

```
    Insert(x):
1       store_Aux(x);
2       clear_Aux;

    Delete(x):
3       p =locate_structure(x);
4       terminate(x,p);
5       if (x ∈ M) then
6           begin
7               top_with_M(B_p); /* B_p is a bin pointed to by p */
8               reload_M;
9           end;
10      else /* x is a non-M-item */
11          if p points to a bin B_p then
12              begin
13                  store_Aux(content(B_p)); /* x is already deleted from B_p */
14                  remove(B_p);
15                  clear_Aux;
16              end;
```

FIG. 4. *A top level description of MMP's Insert and Delete.*

```
    top_with_M(B):
        unload_Aux_M(B);
        while Aux_M = ∅ and B_r is a M-bin do /* B_r—rightmost bin in packing */
            begin
                store_Aux(content(B_r));
                remove(B_r);
                unload_Aux_M(B);
            end;
```

• *discharge*($x_1, \ldots, x_k$,TYPE)—creates a new bin B, packs the items/bundles $x_1, \ldots, x_k$ into B, sets the type of B to TYPE, and then inserts B into the regular packing:

```
    discharge(x_1, ..., x_k, TYPE):
        open_new_bin(B);
        pack(B,x_1, ..., x_k,TYPE);
        add(B);
```

• *reload*_M—empty Aux_M by packing each of its bundles into the regular packing via *FF*:

```
    reload_M:
        while Aux_M ≠ ∅ do
            begin
                b=delete_min(Aux_M); /* Extract the smallest M-bundle m from Aux_M */
                p =search_myopic_M;
                if p = null then /* pack m into a new bin */
                    discharge(m,M-bin);
                else pack(B_p,0); /* pack m into B_p, the leftmost possible bin */
            end;
```

We complete this subsection with an observation that it is easy to see that all of the functions outlined above run in logarithmic uniform running time (using the data structures outlined in section 3.1).

We now summarize the top level description of *Insert* and *Delete* in Figure 4.

clear_Aux:
17 phase 1: clear all of the B-items from Aux_B;
18 phase 2: pack L-items, S-items, and T-items from *Aux* into B-bins of the regular packing;
19 phase 3: form LLS-coalitions;
20 phase 4: pack the remaining L-items, S-items, and T-items from *Aux* into non-B-bins;
21 phase 5: *reload*_M;

FIG. 5. *The five phases of clear_Aux.*

3.2.2. Details of *clear_Aux*. In this subsection we provide a description of *clear_Aux*, the most involved function invoked by *Insert* and *Delete* operations. *clear_Aux* proceeds in five phases. The first four phases (Figures 6 through 9) are aimed at the maintenance of LLS-maximality, while the last phase is a simple invocation of *reload*_M aimed at the maintenance of M-thoroughness. The five phases are listed in Figure 5.

We proceed with a detailed description of the first four phases. We utilize the functions defined in the previous subsection, as well as the function *seek* defined below.

• *seek*$(x_1, \ldots, x_k, y_1[s_1^1, \ldots, s_{y1}^1], \ldots, y_j[s_1^j, \ldots, s_{yj}^j])$—given items $x_1, \ldots, x_k$, *seek* searches for an item y_1 in the sources from the list $[s_1^1, \ldots, s_{y1}^1]$, ... , and for an item y_j in the sources from the list $[s_1^j, \ldots, s_{yj}^j]$ such that all of the x's and y's fit together; i.e., their cumulative size does not exceed 1. The first source on each list is always Aux_{K_y} (the type of K_y is the same as the type of the corresponding y item), while the remaining sources are the appropriate inferior trees (see code). The length of each list is always bounded by a small constant. This search gives preference to the "recruitment" of as many y's from *Aux* as possible. When an item currently packed into a bin must be utilized, it is preferred to search for items from the second source, and if that fails from the third source, etc. The search in any source is nothing more than checking the minimum size item in that source, which is easily accomplished in logarithmic time, since each source is maintained as a min-heap.

If the search is successful and the desired y's are found, *seek* proceeds by deleting all of the entries associated with the y's that are currently packed into bins from their respective inferior trees. Further, *seek* deletes such y's from their bins, stores the remaining content of these bins into *Aux*, and finally deletes these bins from the regular packing and destroys the records associated with these bins.

Note that *seek* runs in logarithmic uniform running time, since the number of searches of various balanced trees in the data structure from section 3.1 is bounded by a small constant (depending on the lengths of the lists of sources).

3.2.3. Implementation of queries. In this subsection we briefly comment on the implementation of queries *size* and *packing*. The query *size* asks for the number of bins in the (current) *MMP* packing. It is easy to implement it in $\mathcal{O}(1)$ uniform running time by maintaining a global integer variable *number* that reflects the number of bins in the regular packing. When a query *size* is processed, the algorithm will first look into *Aux*, which will at that time contain at most one L-item, two S-items, and three T-items. *MMP* will then compute (in constant time) z, the number of bins needed to pack these items from *Aux* ($z \leq 2$), and return *number* $+ z$ as the response to query *size*.

Recall that the query *packing* requests a description of the packing in the form of

```
    phase 1:
22      while Aux_B ≠ ∅ do
23          begin
24              b = delete_min(Aux_B);
25              if seek(b, l[Aux_L,4,3,2]) then
26                  discharge(b, l,BL);
27              else if seek(b, s[Aux_S,9,8,7,6]) then
28                  if seek(b, s, t[Aux_T,15,14,13,12,11]) then
29                      discharge(b, s, t,BST);
30                  else discharge(b, s,BS);
31              else if seek(b, t[Aux_T,15,14,13,12]) then
32                  if seek(b, t, t'[Aux_T,15,14,13,12]) then
33                      discharge(b, t, t',BTT);
34                  else discharge(b, t,BT);
35              else discharge(b,B);
36          end;
```

FIG. 6. *Phase* 1 *of clear_Aux: clear all of the B-items from* Aux_B.

a list of pairs $(x, \mathrm{Bin}(x))$, where $\mathrm{Bin}(x)$ denotes the bin into which an item x is packed, in time linear in the number of items in the current instance. Such a description of the regular packing can be obtained by a preorder traversal of the 2-3 tree representing the regular packing, and a computation of the packing of the items from *Aux* (without actually packing the items from *Aux* into bins, thereby removing these items from *Aux*). It is easy to see that this processing requires a uniform running time that is linear in n, the size of the (current) list L.

4. Competitive ratio and running time of *MMP*.

4.1. *MMP* is $\frac{5}{4}$-competitive. In this section the proof of the upper bound on the competitive ratio of *MMP* is presented. Lower bound examples with a $\frac{5}{4}$ ratio can be given; hence, this bound is tight. See Figure 10 for details.

4.1.1. Overview of the proof of the upper bound on the competitive ratio of *MMP*. The proof of the upper bound on the competitive ratio of *MMP* consists of several parts. First, we establish that *MMP* maintains regular packings that are LLS-maximal. Second, we show that *MMP* maintains M-thoroughness. Once these two important facts about *MMP* are proved (see subsections 4.1.2 and 4.1.3 below), we consider only lists of non-M-items and prove that the upper bound on the competitive ratio of *MMP* for such lists is $\frac{5}{4}$. This is the most difficult part of the proof. We then prove that *MMP* is $\frac{5}{4}$-competitive for arbitrary lists by an easy application of M-thoroughness.

4.1.2. *MMP* maintains LLS-maximality. In this subsection we prove the LLS-maximality of packings produced by *MMP*. In the proof, we will appeal to the code presented in the previous section.

LEMMA 1. *MMP maintains LLS-maximality of packings of lists of non-M-items.*

Proof. The proof proceeds by induction on the number of *Insert* and *Delete* operations processed by *MMP*. Suppose that the packing P produced by *MMP* is LLS-maximal immediately before an *Insert* or *Delete* operation is requested from *MMP*. We proceed to show that *MMP* will process that operation in a manner that maintains LLS-maximality.

```
    phase 2:
37      do
39          l = min(Aux_L);
40          p = search_myopic_L(l);
41          if p points to a B-bin, call it B_p then
38              begin
42                  success = true;
43                  l = delete_min(Aux_L);
44                  store_Aux({y ∈ content(B_p)| y of type ≺ L});
45                  pack(B_p, l,BL);
46                  top_with_M(B_p);
47              end
48      while Aux_L ≠ ∅ and success;
49      do
50          s = min(Aux_S);
51          p = search_myopic_S(s);
52          if p points to a B-bin, call it B_p then
53              begin
54                  success = true;
55                  s = delete_min(Aux_S);
56                  store_Aux({y ∈ content(B_p)| y of type ≺ S});
57                  if seek(b, s, t[Aux_T,15,14,13,12,11]) then /* b ∈ content(B_p) */
58                      pack(B_p, s, t,BST);
59                  else pack(B_p, s,BS);
60                  top_with_M(B_p);
61              end
62      while Aux_S ≠ ∅ and success;
63      do
64          t = min(Aux_T);
65          p = search_myopic_T(t);
66          if p points to a B-bin, call it B_p then
67              begin
68                  success = true;
69                  t = delete_min(Aux_T);
70                  store_Aux({y ∈ content(B_p)| y of type ≺ T});
71                  if seek(b, t, t'[Aux_T,15,14,13,12]) then /* b ∈ content(B_p) */
72                      pack(B_p, t, t',BTT);
73                  else pack(B_p, t,BT);
74                  top_with_M(B_p);
75              end
76      while Aux_T ≠ ∅ and success;
```

FIG. 7. *Phase* 2 *of clear_Aux: pack L-items, S-items, and T-items from Aux into B-bins of the regular packing.*

LLS-maximality concerns only non-M-items. Thus, operations of interest here are *Inserts* and *Deletes* of non-M-items. To complete the proof, it suffices to examine the first four phases of *clear_Aux* and to verify that their execution maintains LLS-maximality. This is in turn easy to verify by inspection of the definitions of thoroughness and LLS-maximality (Definitions 2 and 3, respectively) and the code of *clear_Aux*. □

```
    phase 3:
77      do
78          l = min(Aux_L);
79          success = seek(l, l'[Aux_L,4,3], s[Aux_S,9,8]);
80          if success then discharge(l, l', s,LLS);
81      while Aux_L ≠ ∅ and success;
82      do
83          s = min(Aux_S);
84          success = seek(s, l_1[Aux_L,4,3], l_2[Aux_L,4,3]);
85          if success then discharge(l_1, l_2, s,LLS);
86      while Aux_S ≠ ∅ and success;
```

FIG. 8. *Phase* 3 *of clear_Aux: form LLS–coalitions.*

```
    phase 4:
87      while |Aux_L| ≥ 2 do
88          begin
89              l_1 = delete_min(Aux_L);
90              l_2 = delete_min(Aux_L);
91              if seek(l_1, l_2, t[Aux_T,15,14]) then discharge(l_1, l_2, t,LLT);
92              else discharge(l_1, l_2,LL);
93          end;
94      while |Aux_S| ≥ 3 do
95          begin
96              s_1 = delete_min(Aux_S);
97              s_2 = delete_min(Aux_S);
98              s_3 = delete_min(Aux_S);
99              if seek(s_1, s_2, s_3, t[Aux_T,15]) then discharge(s_1, s_2, s_3, t,SSST);
100             else discharge(s_1, s_2, s_3,SSS);
101         end;
102     while |Aux_T| ≥ 4 do
103         begin
104             t_1 = delete_min(Aux_T);
105             t_2 = delete_min(Aux_T);
106             t_3 = delete_min(Aux_T);
107             t_4 = delete_min(Aux_T);
108             discharge(t_1, t_2, t_3, t_4,TTTT);
109         end;
```

FIG. 9. *Phase* 4 *of clear_Aux: pack the remaining L-items, S-items, and T-items from Aux into non-B-bins.*

4.1.3. *MMP* maintains M-thoroughness. In this subsection we prove that *MMP* maintains M-thorough packings. This proof also proceeds by induction on the number of *Insert* and *Delete* operations processed by *MMP*. Suppose that the packing P produced by *MMP* is M-thorough immediately before an *Insert* or *Delete* operation is requested from *MMP*. We proceed to show that *MMP* will process that operation in a manner that maintains M-thoroughness.

We first consider the operation *Insert*(x): the item x is simply stored into *Aux*, and then *clear_Aux* is invoked. Within *clear_Aux*, each bin B whose content is changed (B can be a new bin or a bin whose content partially changes in the course of execution

$$L = \Big(\frac{1}{2}+\epsilon, \frac{1}{2}+2\epsilon, \ldots, \frac{1}{2}+\frac{\mathrm{N}}{2}\epsilon, \ldots, \frac{1}{2}+\mathrm{N}\epsilon,$$
$$\frac{1}{2}-\mathrm{N}\epsilon, \frac{1}{2}-(\mathrm{N}-1)\epsilon, \ldots, \frac{1}{2}-\frac{\mathrm{N}}{2}\epsilon, \ldots, \frac{1}{2}-\epsilon \Big)$$

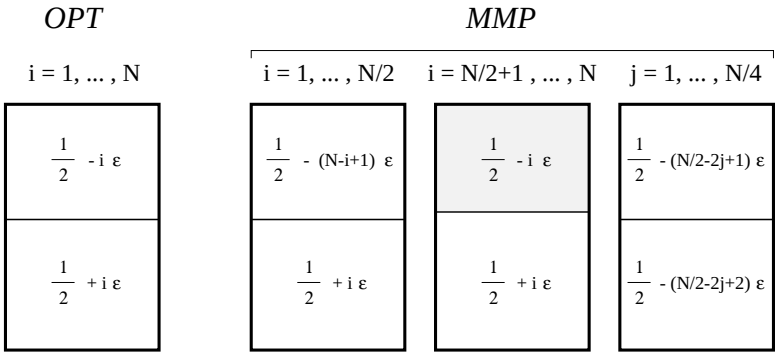


FIG. 10. *A lower bound example for MMP. Here the items are inserted precisely in the order in which they are listed in L above. This example proves that there are arbitrarily large lists L (N is an arbitrarily large integer divisible by 4) for which OPT(L) = N, and MMP(L) = $\frac{5}{4}N$, thus establishing a lower bound of $\frac{5}{4}$ on the competitive ratio of MMP, i.e., R(MMP) $\geq \frac{5}{4}$.*

of *clear_Aux*) is filled with M-bundles via the *top_with_*M function. This function continues filling B until an M-bundle that cannot fit into B is found, in which case *level*$(B) > \frac{4}{5}$ or until there are no more M-bins and Aux_{M} is empty. Finally, phase 5 of *clear_Aux* is an execution of the function *reload_*M. This function will, in case Aux_{M} is *not* empty at the conclusion of maintaining LLS-maximality (phases 1 through 4), accomplish M-thoroughness by packing the M-bundles from Aux_{M} into the regular packing in an *FF* fashion. This will guarantee that (1) there could only be M-bins if each non-M-bin has a level $> \frac{4}{5}$ and (2) any M-bin, except, perhaps, for the rightmost bin in the packing, has a level $> \frac{4}{5}$.

Next, we consider the operation *Delete*(x), where x is a non-M-item. If x is packed into a bin B_p, x is deleted from B_p, the remaining content of B_p is inserted into *Aux*, B_p is deleted from the regular packing, and *clear_Aux* is invoked. (As argued above *clear_Aux* maintains the M-thoroughness of the packing.) Note that deletion of non-M-items from *Aux* has no effect on M-thoroughness of *MMP*. Finally, deletion of M-items (see code: lines 3–9 in *Delete*) also preserves M-thoroughness via functions *top_with_*M and *reload_*M.

Thus we proved the following lemma.

LEMMA 2. *MMP maintains M-thoroughness of packings of arbitrary lists.*

4.1.4. Consideration of lists with no M-items. We fix an arbitrary list L that contains no M-items. Recall that L may be obtained by an arbitrary sequence of *Inserts* and *Deletes* of items. This arbitrariness may lead to various *MMP* packings of L. We thus fix an arbitrary *MMP* packing P_L of L. P_L consists of the regular packing P_0 and the auxiliary storage *Aux*.

We then fix an arbitrary optimal packing OPT_0 of L and derive from it another optimal packing *OPT* of L. *OPT* is a reordering of the bins from OPT_0 such that the bins of type BL are the leftmost bins of *OPT*, the bins of type BST are the next

leftmost bins of OPT, ... , and the bins of type T are the rightmost bins of OPT.

We note that it will be convenient to fix P, a reordering of the bins from P_0. P is defined as follows: the kth leftmost B-bin in OPT and the kth leftmost B-bin in P must contain the same B-item; the order of non-B-bins of P_0 and P is identical.

For both OPT and P, let the *index* of a B-bin B in the packing $OPT(P)$ be the number assigned to B in the "left to right" numbering of the bins from $OPT(P)$. Clearly, for an arbitrary B-item b from L, the indices of B-bins into which b is packed in OPT and P are equal.

Intuitively, the above construction will enable us to view OPT and P in a special way: we may imagine that the B-items of L are "static"—they "remain in the same bin" in an imaginary transformation between OPT and P, and the L-items, S-item, and T-items "migrate," since they may be packed into the k_1th leftmost bin in OPT and into the k_2th leftmost bin in P, where $k_1 \neq k_2$.

The underlying idea. The underlying idea of the proof is to develop an elegant way of capturing the following imaginary series of events. At the outset, someone "glued" the B-items from L into $\mathcal{B}$ bins, one B-item per bin. That person is then required to take the non-B-items from L and pack them into those $\mathcal{B}$ bins and as many additional bins as necessary (these will be non-B-bins), so as to create OPT. Next, that person is required to remove all of the non-B-items from bins and retain the bins containing the glued B-items in the "left to right" order produced by OPT. Finally, that person is again required to take the non-B-items from L and pack them into those $\mathcal{B}$ bins and as many additional bins as necessary (again, these will be non-B-bins), so as to create P. As indicated in the course of defining P, we are interested in "migrations" of non-B-items *from* bins of all types in OPT *to* B-bins in P. The notion of glued B-items motivated the definitions of OPT and P: B-items may be viewed as "static"; they do not move from bin to bin when the packing is changed from OPT to P. L-items, S-items, and T-items, on the other hand, may "migrate"; i.e., the indices of the bins they are packed into in OPT and P may differ. The remainder of the proof explores this "itemographic process."

We perform an extensive analysis of the structure of P in terms of different types of bins in P and OPT and their respective multiplicity f_i, $1 \leq i \leq 30$, where f_i denotes the number of bins of type i in OPT (i is the canonical index of bin types with respect to the superiority relation. See Figure 1.) The proof will *not at all* depend on the particular sequence of *Inserts* and *Deletes* that led to L and its *MMP* packing P_L. The proof will depend *only* on the fact, proved in subsection 4.1.2, that *MMP* produces packings that are LLS-maximal. Based on that property, we derive lower bounds on the quantities L_B, S_B, and T_B, and N_{LLS} defined thus.

DEFINITION 5. *Let L_B (S_B, T_B) denote the number of L-items (S-items, T-items) in B-bins in P. Let N_{LLS} denote the number of bins of type LLS in P.*

A preliminary result. We begin with a preliminary lemma. We establish an upper bound on the number of bins *MMP* would require to pack a given number of L-items, S-items, and T-items.

LEMMA 3. *Suppose L contains only L-items, S-items, and T-items.* MMP *will pack L into at most $\frac{\mathcal{L}}{2} + \frac{\mathcal{S}}{3} + \frac{\mathcal{T}}{4} + 2$ bins.*

Proof. Recall that *MMP* maintains LLS-maximality (see Lemma 1). P_0 will require precisely $\lfloor \frac{\mathcal{L}}{2} \rfloor$ bins with two L-items (these bins may each contain an additional S-item or an additional T-item), followed by at most $\lfloor \frac{\mathcal{S}}{3} \rfloor$ bins with three S-items (analogously, these bins may each contain an additional T-item), followed by at most

$\lfloor \frac{\mathcal{T}}{4} \rfloor$ bins with four T-items. In addition, *Aux* might contain at most one L-item, two S-items, and three T-items. All of these items can fit into at most two bins (one bin of type LTT and one bin of type SST). Hence the number of bins required by *MMP* to pack all of the items of L is at most

$$\left\lfloor \frac{\mathcal{L}}{2} \right\rfloor + \left\lfloor \frac{\mathcal{S}}{3} \right\rfloor + \left\lfloor \frac{\mathcal{T}}{4} \right\rfloor + 2 \leq \frac{\mathcal{L}}{2} + \frac{\mathcal{S}}{3} + \frac{\mathcal{T}}{4} + 2. \qquad \square$$

Several technical results. We now proceed with several technical results that are important in estimating the quantities L_B, S_B, T_B, and N_{LLS}. A precise estimate of these quantities is the key ingredient in the proof. Each of the following lemmas contains three statements that are quite analogous and are moreover proved analogously. We adopt this particular style of presentation to avoid unnecessary repetition.

LEMMA 4. *Consider an optimal packing P_{opt} consisting only of k bins of type BL (BS; LLS). Consider a list L whose optimal packing could be P_{opt} and is constructed in stages:*[5]

stage 0: $X =$ *the set of items in* P_{opt}*;*
$L = \emptyset$*;*
go to stage 1;

stage i: ($1 \leq i \leq k$)
$Y =$ (*the smallest B-item in* X (*B-item; two L-items*),
the smallest L-item in X (*S-item; S-item*));
$L = L \cdot Y$*;*
$X = X -$ *set of items in* Y*;*
go to stage $i + 1$*;*

stage $k + 1$: *return* L*;*

Then the B-item and the L-item (the B-item and the S-item; the two L-items and the S-item) assigned to Y in stages 1 *through* $\alpha = \lceil \frac{k}{2} \rceil$ ($\lceil \frac{k}{2} \rceil$; $\lceil \frac{k}{3} \rceil$), *respectively, must fit together into a bin of type BL (BS, LLS).*

Proof. Suppose by way of contradiction that it is *not* the case that the B-item and the L-item (the B-item and the S-item; the two L-items and the S-item) assigned to Y in all of the stages 1 through $\alpha = \lceil \frac{k}{2} \rceil$ ($\lceil \frac{k}{2} \rceil$; $\lceil \frac{k}{3} \rceil$), respectively, fit together into a bin of type BL (BS, LLS).

Then there exists a positive integer $1 \leq \beta < \alpha$ such that only the B-items and the L-items (the B-items and the S-items; the L-items and the S-items) assigned to Y in stages 1 through β, respectively, fit into a bin of type BL (BS; LLS).

Consider L arranged in a table with 2 (2;3) rows and k columns as follows: ith column contains exactly the items added to L in the ith stage of the construction of L, where within a column the items are arranged in successive rows according to the order in which they were added to L. Refer to Figure 11 for details.

Note that only the items from the first β columns of this table can, according to the supposition, fit together into bins: one bin per column of type BL (BS; LLS).

Let $\mathcal{C}_1$ be the set of all of the items that lie in the first row (first row; first two rows) and are also in the first β columns. Let $n_1 = card(\mathcal{C}_1)$. Clearly $n_1 = \beta$ (β; 2β). Let $\mathcal{C}_2$ be the set of all of the items in the last row that belong to the columns $\beta + 1, \beta + 2, ..., k$. Let $n_2 = card(\mathcal{C}_2)$. Clearly $n_2 = k - \beta$.

Any item $a \in \mathcal{C}_2$ could only be packed into a bin of type BL (BS; LLS) if b, the B-item (the B-item; at least one of the two L-items) with which a would be packed,

[5] In the construction below, the operator $\cdot$ denotes concatenation; i.e., $\cdot$ is a binary operator that appends its second operand to its first operand.

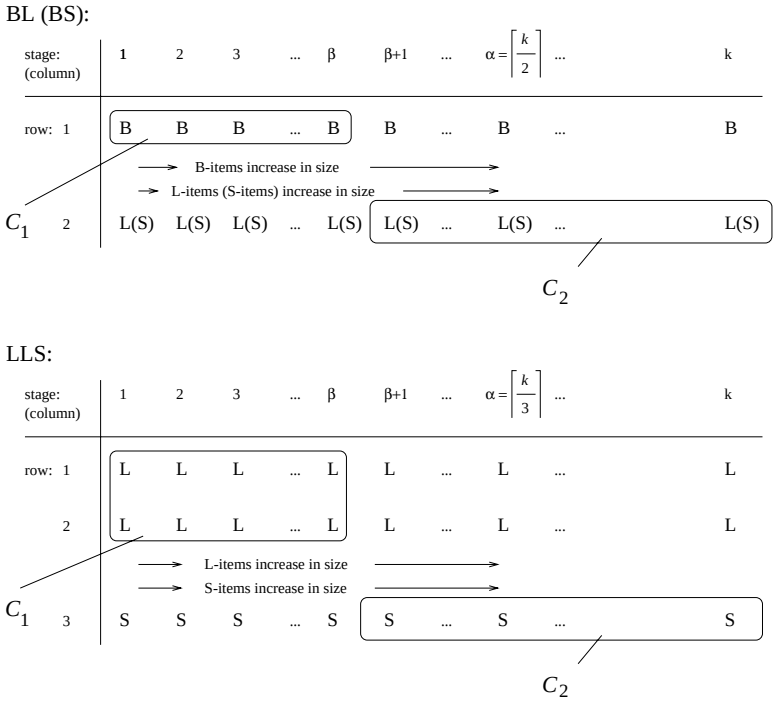


FIG. 11. *A tabular arrangement of L. Letters B, L, and S denote B-items, L-items, and S-items of L, respectively.*

would be from $\mathcal{C}_1$; otherwise the level of such a bin would exceed 1. Thus, since the optimal packing of L contains only bins of type BL (BS; LLS), there should be sufficiently many items in $\mathcal{C}_1$ to ensure the packing of each $a \in \mathcal{C}_2$; i.e., it should be the case that $n_1 \geq n_2$, i.e., $n_1 - n_2 \geq 0$. However,

$$\begin{aligned} n_1 - n_2 = (c-1)\beta - (k-\beta) \;&=\; c\beta - k \;\leq c\left(\left\lceil \frac{k}{c} \right\rceil - 1\right) - k \\ &\leq \left(\frac{k+c-1}{c}\right) - c - k \;=\; -1, \end{aligned}$$

where c=2 (2;3) bins of type BL (BS; LLS). This is a contradiction. □

Clearly, if *MMP* packs this particular list L in a series of *Inserts* of items in the order in which the items appear in L, *MMP* will pack at least α bins of type BL (BS; LLS).

COROLLARY 1. *The MMP packing of the list L defined in the statement of Lemma* 4*, carried out as a series of Inserts of items in the order in which the items appear in L, must contain at least* $\alpha = \lceil \frac{k}{2} \rceil$ ($\lceil \frac{k}{2} \rceil$; $\lceil \frac{k}{3} \rceil$) *bins of type BL (BS, LLS).*

Thus we have shown that a very particular sequence of *Inserts* will pack an L with a very specific optimal packing in the manner that will, informally, "salvage" about one half (one half; one third) of the bins of type BL (BS; LLS). The following lemma shows that this particular order of packing the items of L, inserted one by one in the order in which they appear in L, is not at all essential. Furthermore, it is not essential that an optimal packing of L consists only of bins of type BL (BS; LLS). In other words, to determine the lower bound on the number of bins of type BL (BS; LLS) in

an *MMP* packing of a list L containing only B-items, L-items, S-items, and T-items (B-items, S-items, and T-items [note a careful avoidance of L-items); L-items, S-items, and T-items (note a careful avoidance of B-items)), the only relevant information is the maximum number of bins of type BL (BS; LLS) that can be constructed from items in L. *MMP* will, informally, "salvage" about one half (one half; one third) of that maximum number of bins of type BL (BS; LLS). We again choose this particular style of presentation to avoid the unnecessary repetition.

LEMMA 5. *Let L' be a list containing B-items, L-items, S-items, and T-items (B-items, S-items, and T-items; L-items, S-items, and T-items). Let k be the maximum number of bins of type BL (BS; LLS) that could be packed from the items of L'. Then any MMP packing of L', obtained by an arbitrary sequence of Inserts and Deletes leading to L', contains at least $\alpha = \lceil \frac{k}{2} \rceil$ ($\lceil \frac{k}{2} \rceil$; $\lceil \frac{k}{3} \rceil$) bins of type BL (BS; LLS).*

Proof. Suppose by way of contradiction that there is an *MMP* packing P_{bad} of L' that contains only $\gamma < \alpha$ bins of type BL (BS; LLS). Let $P_{desired}$ be a maximum cardinality packing of B-items and L-items (B-items and S-items; L-items and S-items) from L' into bins of type BL (BS; LLS) (note the correspondence of $P_{desired}$ with P_{opt} from Lemma 11). By a hypothesis, the maximum number of bins of type BL (BS; LLS) in the $P_{desired}$ packing is precisely k. Let L be the list of items derived from $P_{desired}$ in the manner of Lemma 11. Let P' be the *MMP* packing of L obtained by a sequence of *Inserts* of items from L in the order in which the items appear in L (note the correspondence with the situation from Corollary 1). Since there are only $\gamma < \alpha$ bins of type BL (BS; LLS) in P_{bad}, there must be at least one B-item b (one B-item b; two L-items l_1, l_2) and one L-item l (one S-item s; one S-item s) that are packed in one of the first α bins of type BL (BS; LLS) in P', and are not packed in a bin of type BL (BS; LLS) in P_{bad}. Note that the size of each of b and l (b and s; l_1, l_2, and s) is no greater than the size of the B-item and the L-item (the B-item and the S-item; the two L-items and the S-item) from the αth bin in P_{bad}. Thus b and l (b and s; l_1, l_2, and s) can fit into a bin. This is a contradiction, since *MMP* produces packings that are thorough (thorough; LLS-maximal) (Lemma 1) for bins of type BL (BS; LLS), and would therefore have packed at least one more bin of type BL (BS; LLS) in P_{bad}. □

Toward a lower bound on N_{LLS}. We proceed by stating several properties that will be used to obtain a lower bound on the value of N_{LLS}. We begin by considering bins of type LSTT. We show that two L-items and one S-item, each from any, and not necessarily the same, bin of type LSTT must fit into a bin.

LEMMA 6. *Let l_1, l_2, and s be two L-items and an S-item, each from any, and not necessarily the same, bin of type LSTT. Then l_1, l_2, and s can fit into a bin.*

Proof. It suffices to show that $2L_{max} + S_{max} \leq 1$, where L_{max} (S_{max}) denotes the maximum size of L-items (S-items) in any bin of type LSTT;

$$L_{max} = 1 - S_{min} - 2T_{min} < 1 - \frac{1}{4} - 2 \cdot \frac{1}{5} = \frac{7}{20},$$
$$S_{max} = 1 - L_{min} - 2T_{min} < 1 - \frac{1}{3} - 2 \cdot \frac{1}{5} = \frac{4}{15},$$
$$2L_{max} + S_{max} < 2 \cdot \frac{7}{20} + \frac{4}{15} = \frac{29}{30} < 1,$$

where S_{min} and T_{min} denote the minimum size of an S-item and an T-item, respectively. □

Establishing a good lower bound on the number of bins of type LLS that would be

produced by *MMP* is very important later in the proof, since that bound is a measure of the success of the technique of LLS-coalitions utilized by *MMP*. To that end, we state and prove the following lemma.

LEMMA 7. *Suppose that L is a list of non-B-items. Let P_{opt} be an arbitrary optimal packing of L satisfying the following three conditions:*

1. *P_{opt} contains at least n_1 bins of type LLS;*
2. *at least n_2 L-items from L are not packed into bins of type LLS in P_{opt}, and each of these L-items can fit into a bin of type LSTT;*
3. *at least n_2 S-items from L are not packed into bins of type LLS in P_{opt}, and each of these S-items can fit into a bin of type LSTT.*

Then the number of bins of type LLS in any MMP *packing of L is at least $\lceil \frac{n_1+n_2}{4} \rceil - 1$.*

Proof. First, the following fact follows immediately from Lemma 6.

FACT 1. *The maximum number of bins of type LLS that can be packed from the items of L is at least $n_0 = n_1 + \lfloor \frac{n_2}{2} \rfloor$.*

Now to prove this lemma, we consider two cases.

Case 1. $n_1 \geq n_2$.

By Lemma 5, *MMP* will pack at least $\lceil \frac{n_0}{3} \rceil$ bins of type LLS:

$$\left\lceil \frac{n_0}{3} \right\rceil = \left\lceil \frac{n_1 + \lfloor \frac{n_2}{2} \rfloor}{3} \right\rceil \geq \left\lceil \frac{n_1 + \frac{n_2-1}{2}}{3} \right\rceil = \left\lceil \frac{\frac{3}{4}n_1 + \frac{1}{4}n_1 + \frac{1}{2}n_2 - \frac{1}{2}}{3} \right\rceil$$
$$= \left\lceil \frac{n_1+n_2}{4} - \frac{1}{6} \right\rceil \geq \left\lceil \frac{n_1+n_2}{4} \right\rceil - 1.$$

Case 2. $n_1 < n_2$.

By Lemma 6 and the fact that *MMP* maintains LLS-maximality of bins of type LLS (Lemma 1), *MMP* will pack at least $\lfloor \frac{n_2}{2} \rfloor$ bins of type LLS. Further, note that $n_1 + n_2 < 2n_2$:

$$\left\lfloor \frac{n_2}{2} \right\rfloor \geq \left\lfloor \frac{\frac{n_1+n_2}{2}}{2} \right\rfloor = \left\lfloor \frac{n_1+n_2}{4} \right\rfloor \geq \left\lceil \frac{n_1+n_2}{4} \right\rceil - 1. \qquad \square$$

Toward lower bounds on L_B, S_B, and T_B. The next several lemmas establish important properties that will be used later to bound L_B, S_B, and T_B.

LEMMA 8. *A B-item from a bin of type BL can be packed together with any S-item.*

Proof. It suffices to show that $B_{max} + S_{max} < 1$, where B_{max} denotes the maximum size of B-items in any bin of type BL, and S_{max} denotes the maximum size of S-items:

$$B_{max} = 1 - L_{min} < 1 - \frac{1}{3} = \frac{2}{3}, \quad S_{max} = \frac{1}{3}, \quad B_{max} + S_{max} < \frac{2}{3} + \frac{1}{3} = 1,$$

where L_{min} denotes the minimum size of L-items. $\square$

The next nine lemmas are proved quite analogously.

LEMMA 9. *A B-item from a bin of type BL can be packed together with any T-item.*

LEMMA 10. *A B-item from a bin of type BST can be packed together with an L-item from a bin of type LLS.*

LEMMA 11. *A B-item from a bin of type BST can be packed together with an L-item from a bin of type LSTT.*

LEMMA 12. *A B-item from a bin of type BST can be packed together with any S-item.*

LEMMA 13. *A B-item from a bin of type BST can be packed together with any T-item.*

LEMMA 14. *A B-item from a bin of type BS can be packed together with any T-item.*

LEMMA 15. *A B-item from a bin of type BTT can be packed together with an L-item from a bin of type LSTT.*

LEMMA 16. *A B-item from a bin of type BTT can be packed together with any S-item.*

LEMMA 17. *A B-item from a bin of type BTT can be packed together with any T-item.*

***MMP* is $\frac{5}{4}$-competitive for lists with no M-items.** Recall that there are 30 possible types of bins involving B-items, L-items, S-items, and T-items: BL, BST, ... , T (see Figure 1). We will find it convenient to refer to these types of bins according to their canonical index in the lexicographical ordering based on the relation of superiority. For example, a bin of type 1 is a bin of type BL, a bin of type 2 is a bin of type BST, ... , a bin of type 30 is a bin of type T, as depicted in Figure 1.

Next, recall that only a subset of the above types of bins is *allowed* in *MMP* packings. The allowed bin types are BL, BST, BS, BTT, BT, B, LLS, LLT, LL, SSST, SSS, and TTTT. Alternatively, using the canonical indices, the allowed bin types are 1, 2, 3, 4, 5, 6, 7, 8, 9, 18, 19, and 27.

Intuitively, certain types of bins will be more important for the proof of the upper bound on the competitive ratio than the others. We therefore classify the bin types into the *front* (more important) types of bins and the *back* (less important) types of bins.

DEFINITION 6. *The* front *types of bins are BL, BST, BS, BTT, BT, B, LLS, and LSTT* (*types* $1, 2, \ldots, 7, 11$). *All of the other types are* back *types of bins. Let* $\mathcal{L}_{back}$ ($\mathcal{S}_{back}$, $\mathcal{T}_{back}$) *denote the number of L-items (S-items, T-items) in the bins of back types in OPT.*

Recall that f_i, $1 \le i \le 30$, denotes the number of bins of type i in *OPT*. The following equations hold (see Figure 1):

$$\mathcal{L}_{back} = 2(f_8 + f_9) + f_{10} + \sum_{i=12}^{17} f_i,$$

$$\mathcal{S}_{back} = 3(f_{18} + f_{19}) + 2(f_{10} + f_{20} + f_{21} + f_{22}) + f_{12} + f_{13} + \sum_{i=23}^{26} f_i,$$

$$\begin{aligned}\mathcal{T}_{back} = {} & 4f_{27} + 3(f_{14} + f_{23} + f_{28}) + 2(f_{15} + f_{20} + f_{24} + f_{29}) \\ & + f_8 + f_{12} + f_{16} + f_{18} + f_{21} + f_{25} + f_{30},\end{aligned}$$

$$\mathcal{L} = f_1 + 2f_7 + f_{11} + \mathcal{L}_{back},$$

$$\mathcal{S} = f_2 + f_3 + f_7 + f_{11} + \mathcal{S}_{back},$$

$$\mathcal{T} = f_2 + 2f_4 + f_5 + 2f_{11} + \mathcal{T}_{back}.$$

We next furnish a technical definition of the *list of indices*.

DEFINITION 7. *A list of indices* $l = i_1, i_2, \ldots, i_k$ *is a list of positive integers.* $*$ *will be used as an abbreviation for the list* 8, 9, 10, 12, 13, ..., 30.[6]

[6]Note that this list corresponds to the back types of bins.

We would like to use this notation to count the number of bins in OPT whose type is an element of the list of indices l. To that end, we furnish the following definition.

DEFINITION 8. *Let l denote a list of indices. Let f_l denote the number of bins in OPT whose type is an element of l. Then $f_l = \sum_{i \in l} f_i$.*

For example, the number of bins in OPT that contains a B-item and an S-item (bins of type BST (2) or BS (3)) is $f_{2,3} = f_2 + f_3$.

Let us define another technical definition of *ranges of bins*. Each range of bins r_i captures precisely the indices of B-bins of type i in OPT.

DEFINITION 9. *Ranges of bins r_i, $1 \leq i \leq 6$, are segments of positive integers defined as follows: $r_1 = [1, \ldots, f_1]$, $r_2 = [f_1 + 1, \ldots, f_{1,2}]$, $r_3 = [f_{1,2} + 1, \ldots, f_{1,2,3}]$, $r_4 = [f_{1,2,3} + 1, \ldots, f_{1,2,3,4}]$, $r_5 = [f_{1,\ldots,4} + 1, \ldots, f_{1,\ldots,5}]$, and $r_6 = [f_{1,\ldots,5} + 1, \ldots, f_{1,\ldots,6}]$.*

We would like to count the number of items of a certain type (L-items, S-items, or T-items) that are packed into a bin of type i in OPT and into a B-bin with an index from the range r_j in P. Intuitively, we would like to count how many items of a certain type "migrated" from bins of type i in OPT (intuitively, the source bins) to B-bins in P that "used to be" bins of type j in OPT; i.e., the index of such bins in both OPT and P is from the range r_j (intuitively, the destination bins).

We would also like to count the number of items of a certain type (L-items, S-items, or T-items) that are packed into a bin in OPT whose type is an element of the list of indices l_s (intuitively, the types of source bins), and into a B-bin in P whose index is from a range of bins indexed by an element of the list of indices l_d (intuitively, the ranges of destination bins).

DEFINITION 10. *Let L_d^s (S_d^s, T_d^s), $s \in \{1, 2, \ldots, 5, 7, 8, \ldots, 30\}$ and $d \in \{1, 2, \ldots, 6\}$, denote the number of L-items (S-items, T-items) that are packed into bins of type s (source) in OPT, and into B-bins with an index from r_d (destination) in P.*

Let l_s and l_d denote lists of indices. Let $L_{l_d}^{l_s}$ ($S_{l_d}^{l_s}$, $T_{l_d}^{l_s}$) denote the number of L-items (S-items, T-items) that are packed into a bin in OPT whose type is an element of the list of indices l_s, and into a B-bin in P whose index is from a range of bins indexed by an element of the list of indices l_d. Then

$$L_{l_d}^{l_s} = \sum_{i \in l_s} \sum_{j \in l_d} L_j^i \quad \left(S_{l_d}^{l_s} = \sum_{i \in l_s} \sum_{j \in l_d} S_j^i \,, \quad T_{l_d}^{l_s} = \sum_{i \in l_s} \sum_{j \in l_d} T_j^i \right).$$

We illustrate this notation with two examples.

First, L_1^7 denotes the number of L-items that are packed into bins of type 7 (LLS) in OPT, and into bins with an index from r_1 in P, that is, into one of the f_1 leftmost bins, bins that contain a B-item that is packed into a bin of type 1 (BL) in OPT.

Second, the number of T-items that are packed into a bin of one of the types BST (2), BT (4), or LLT (8) in OPT, and into a B-bin whose index is from one of the ranges r_2 or r_3 (that is, into one of the bins that contains a B-item that is packed into a bin of type 2 (BST) or a bin of type 3 (BS) in OPT) is written as follows:

$$T_{2,3}^{2,4,8} = \sum_{i \in 2,4,8} \sum_{j \in 2,3} T_j^i = T_2^2 + T_2^4 + T_2^8 + T_3^2 + T_3^4 + T_3^8.$$

We now state and prove the main result of this subsection.

LEMMA 18. *$MMP(L) \leq \frac{5}{4} OPT(L) + 3$.*

Proof. We begin with a series of four claims that establish lower bounds on the values of L_B, S_B, T_B, and N_{LLS}, respectively.

CLAIM 1. $L_B \geq L^{1,7,11,*}_{1,3,\ldots,6} + L^{1,*}_2 + \min(2f_7 + f_{11} - L^{7,11}_{1,3,\ldots,6}, f_2 - L^{1,*}_2)$.

Proof. The key to the proof of this lemma is an observation that any B-item from a bin of type 2 (BST) can be packed together with any L-item from a bin of type 7 or 11 (LLS or LSTT) (Lemmas 10 and 11).

The right-hand side of the above inequality consists of the follwing three additive terms.

1. The first term accounts for all of the L-items that are packed into one of the bins of type 1, 7, 11, or into one of the bins of back type in *OPT*, and into one of the B-bins in P that contains a B-item that is packed into a bin of type 1, 3, 4, 5, or 6 in *OPT* (note a careful avoidance of bins of type 2 (BST)).

2. The second term accounts for all of the L-items that are packed into one of the bins of type 1, or into one of the bins of back type in *OPT*, and into one of the B-bins in P that contains a B-item that is packed into a bin of type 2 (again, note a careful avoidance of bins of type 2 (BST)).

3. The third term, the min term, is somewhat more involved.

The first operand of the min term is $2f_7 + f_{11} - L^{7,11}_{1,3,\ldots,6}$. This number denotes the number of L-items from bins of type 7 (LLS) and 11 (LSTT) in *OPT* that are *not* packed into a bin in P together with a B-item that is packed into a bin of type 1, 3, 4, 5, or 6 in *OPT* (note a careful avoidance of B-items from bins of type 2 (BST) in *OPT*).

The second operand of the min term is $f_2 - L^{1,*}_2$. This number denotes the number of B-items that are (1) packed into a bin of type 2 (BST) in *OPT* and (2) are *not* packed into a bin in P together with an L-item that is packed into a bin of type 1 (BL) or one of the back-type bins in *OPT* (note a careful avoidance of L-items from bins of type 7 (LLS) and 11 (LSTT) in *OPT*).

Thus, by Lemmas 10 and 11, any B-item accounted for by the second operand can be packed together with any L-item accounted for by the first operand of the min expression. By the LLS-maximality of *MMP* (see Lemma 1), P will feature $\min(2f_7 + f_{11} - L^{7,11}_{1,3,\ldots,6}, f_2 - L^{1,*}_2)$ bins of type 1 (BL) containing the above B-items and L-items. □

The following two lemmas are proved quite analogously.

CLAIM 2. $S_B \geq S^{2,3,7,11,*}_{3,5,6} + \min(f_2 + f_3 + f_7 + f_{11} - S^{2,3,7,11}_{3,5,6}, f_1 + f_2 + f_4 - (L_B - L^{1,7,11,*}_{3,5,6}))$.

CLAIM 3. $T_B \geq T^{2,4,5,11,*}_{5,6} + \min(f_2 + 2f_4 + f_5 + 2f_{11} - T^{2,4,5,11}_{5,6},\ f_{1,\ldots,4} - (L_B - L^{1,7,11,*}_{5,6}) - (S_B - S^{2,3,7,11,*}_{5,6}))$.

CLAIM 4. $N_{LLS} \geq \frac{1}{4}(\max(0, f_7 - (L^7_{1,\ldots,6} + S^7_{1,\ldots,6})) + f_{11} - \max(L^{11}_{1,\ldots,6}, S^{11}_{1,\ldots,6})) - 1$.

Proof. The key to the proof of this lemma is the statement of Lemma 7. Let the set of items packed into non-B-bins in *OPT* play the role of L in Lemma 7. Let the non-B-bins of *OPT* play the role of P_{opt} in Lemma 7. Further, note the following conditions.

1. $\max(0, f_7 - (L^7_{1,\ldots,6} + S^7_{1,\ldots,6}))$ is a (conservative) lower bound on the number of bins of type LLS in *OPT* that do *not* contain an item that is packed into a B-bin in P. These $\max(0, f_7 - (L^7_{1,\ldots,6} + S^7_{1,\ldots,6}))$ bins of type LLS play the role of n_1 bins of type LLS in Lemma 7.

2. $f_{11} - \max(L^{11}_{1,\ldots,6}, S^{11}_{1,\ldots,6})$ is a (conservative) lower bound on the number of L-items and the number of S-items from bins of type LSTT in *OPT* that are *not* packed into B-bins in P. These $f_{11} - \max(L^{11}_{1,\ldots,6}, S^{11}_{1,\ldots,6})$ L-items and $f_{11} - \max(L^{11}_{1,\ldots,6}, S^{11}_{1,\ldots,6})$ S-items play the role of n_2 L-items and S-items in Lemma 7.

This claim, then, follows directly from Lemma 7. Note that, unlike in Lemma 7, we did not choose to apply the ceiling operator, as this will have no substantive effect on the analysis of the competitive ratio of *MMP*. □

Next, we apply Lemma 5 to establish lower bounds on the values of L_1^1 and S_3^3. Note that we do not provide a lower bound for T_5^5. Again, while a lower bound on T_5^5 could be established in an analogous manner, accounting for T_5^5 will have no substantive effect on the analysis of the competitive ratio of *MMP*.

CLAIM 5. $L_1^1 \geq \frac{1}{2}(f_1 - \min(f_1, L_1^{7,11,*} + L_{2,\ldots,6}^1))$.

Proof. The proof is immediate from Lemma 5. Let Γ be the set of B-items from L that are packed into bins of type BL in *OPT* and at the same time satisfy the following two properties.

1. They are packed into a bin in *OPT* together with L-items which satisfy the property that they are *not* packed into a bin of type BL in P together with a B-bin with an index from $r_2 \cup r_3 \ldots \cup r_6$. Informally, the B-items from Γ cannot be packed in *OPT* together with L-items that "migrate" from bins with indices from r_1 in *OPT* to other B-bins in P.

2. They are *not* packed into a bin in P together with L-items that are packed into non-B-bins in *OPT*.

A conservative estimate of the lower bound on the cardinality of Γ is $f_1 - \min(f_1, L_1^{7,11,*} + L_{2,\ldots,6}^1)$. We say that the estimate is conservative, since there may be B-items that violate both properties, whereas this bound assumes that each B-item violates at most one property; i.e., it is assumed that there are no "overlaps."

The list consisting of B-items of Γ and L-items that are packed together with a B-item from Γ in *OPT* now plays the role of L' from Lemma 5. Further, the above conservative estimate of the cardinality of Γ plays the role of k from Lemma 5.

The claim follows directly from Lemma 5. Note that, unlike in Lemma 5, we did not apply the ceiling operator, as this will have no substantive effect on the analysis of the competitive ratio of *MMP*. □

The following claim is proved quite analogously.

CLAIM 6. $S_3^3 \geq \frac{1}{2}(f_3 - \min(f_3, L_3^{1,7,11,*} + S_3^{2,7,11,*} + S_{1,2,4,5,6}^3))$.

Next, we note that the number of bins in *OPT* is precisely $OPT(L) = \sum_{i=1}^{30} f_i$. Let us now bound the value of $MMP(L)$ in terms of f_i's, L_B, S_B, T_B, and N_{LLS}. The number of B-bins, $\mathcal{B} = \sum_{i=1}^{6} f_i = f_{1,2,3,4,5,6}$, is identical for *OPT* and P. Further, *MMP* packs L_B (S_B, T_B) L-items (S-items, T-items) into B-bins, and N_{LLS} bins of type LLS. Finally, we assume that the L-items, S-items, and T-items that are not explicitly accounted for via L_B, S_B, T_B, and N_{LLS} are packed in the "worst possible way." Hence, we bound the value of $MMP(L)$ as follows:

$$\begin{aligned} MMP(L) &\leq f_{1,\ldots,6} + N_{LLS} + \frac{1}{2}(\mathcal{L} - L_B - 2N_{LLS}) \\ &\quad + \frac{1}{3}(\mathcal{S} - S_B - N_{LLS}) + \frac{1}{4}(\mathcal{T} - T_B) + 2 \\ &= f_{1,\ldots,6} + \frac{1}{2}(f_1 + 2f_7 + f_{11} + \mathcal{L}_{back} - L_B) \\ &\quad + \frac{1}{3}(f_2 + f_3 + f_7 + f_{11} + \mathcal{S}_{back} - S_B - N_{LLS}) \\ &\quad + \frac{1}{4}(f_2 + 2f_4 + f_5 + 2f_{11} + \mathcal{T}_{back} - T_B) + 2. \end{aligned}$$

Note that

$$\frac{\mathcal{L}_{back}}{2} + \frac{\mathcal{S}_{back}}{3} + \frac{\mathcal{T}_{back}}{4} \leq \frac{5}{4} f_*,$$

which can be verified by a straightforward substitution. Hence, to complete the proof it suffices to show that

$$\mathcal{K} \leq \frac{5}{4} f_{1,\ldots,7,11} + 1,$$

where $\mathcal{K}$ is defined as follows:

$$\begin{aligned}\mathcal{K} = f_{1,\ldots,6} &+ \frac{1}{2}(f_1 + 2f_7 + f_{11} - L_B) \\ &+ \frac{1}{3}(f_2 + f_3 + f_7 + f_{11} - S_B - N_{LLS}) \\ &+ \frac{1}{4}(f_2 + 2f_4 + f_5 + 2f_{11} - T_B).\end{aligned}$$

We outline four conditions, (I),...,(IV), by observing Claims 1 through 4. Our case analysis will be guided by these conditions:

$$\begin{aligned}&\text{(I) holds} \iff 2f_7 + f_{11} - L^{7,11}_{1,3,\ldots,6} \leq f_2 - L^{1,*}_2, \\ &\text{(II) holds} \iff f_2 + f_3 + f_7 + f_{11} - S^{2,3,7,11}_{3,5,6} \leq f_1 + f_2 + f_4 - (L_B - L^{1,7,11,*}_{3,5,6}), \\ &\text{(III) holds} \iff f_2 + 2f_4 + f_5 + 2f_{11} - T^{2,4,5,11}_{5,6} \\ &\qquad\qquad \leq f_{1,\ldots,4} - (L_B - L^{1,7,11,*}_{5,6}) - (S_B - S^{2,3,7,11,*}_{5,6}), \\ &\text{(IV) holds} \iff L^7_{1,\ldots,6} + S^7_{1,\ldots,6} \leq f_7.\end{aligned}$$

Before proceeding with the analysis, we note several useful facts:

1. $L_B \geq \frac{1}{2} f_1$ (immediate from Lemma 5).
2. If (I) holds,

$$\begin{aligned}L_B &\geq L^{1,7,11,*}_{1,3,\ldots,6} + L^{1,*}_2 + 2f_7 + f_{11} - L^{7,11}_{1,3,\ldots,6} \\ &= L^{1,*}_{1,3,\ldots,6} + L^{1,*}_2 + 2f_7 + f_{11} \\ &\geq 2f_7 + f_{11}.\end{aligned}$$

3. If (I) does not hold (¬(I) holds),

$$\begin{aligned}L_B &\geq L^{1,7,11,*}_{1,3,\ldots,6} + L^{1,*}_2 + f_2 - L^{1,*}_2 \\ &= f_2 + L^{1,7,11,*}_{1,3,\ldots,6} \\ &= f_2 + L^1_1 + L^{1,7,11,*}_{3,\ldots,6} + L^{7,11,*}_1 \\ &\geq f_2 + \frac{1}{2}(f_1 - \min(f_1, L^{7,11,*}_1 + L^1_{2,\ldots,6})) + L^{1,7,11,*}_{3,\ldots,6} + L^{7,11,*}_1 \\ &= f_2 + \frac{1}{2}(f_1 - \min(f_1, L^{7,11,*}_1 + L^1_{2,\ldots,6})) + \left(\frac{1}{2} + \frac{1}{2}\right) L^1_{3,\ldots,6} \\ &\quad + L^{7,11,*}_{3,\ldots,6} + \left(\frac{1}{2} + \frac{1}{2}\right) L^{7,11,*}_1 + \left(\frac{1}{2} - \frac{1}{2}\right) L^1_2 \\ &= f_2 + \frac{1}{2} f_1 + \frac{1}{2}(L^{7,11,*}_1 + L^1_{2,\ldots,6}) - \frac{1}{2}\min(f_1, L^{7,11,*}_1 + L^1_{2,\ldots,6})\end{aligned}$$

$$
\begin{aligned}
&+\frac{1}{2}L^1_{3,\dots,6}+L^{7,11,*}_{3,\dots,6}+\frac{1}{2}L^{7,11,*}_1-\frac{1}{2}L^1_2\\
&\geq \frac{1}{2}f_1+f_2-\frac{1}{2}L^1_2+\frac{1}{2}L^1_{3,\dots,6}+L^{7,11,*}_{3,\dots,6}+\frac{1}{2}L^{7,11,*}_1\\
&\geq \frac{1}{2}f_1+f_2-\frac{1}{2}L^1_2+\frac{1}{2}L^{7,11}_{1,3,\dots,6}.
\end{aligned}
$$

4. If (II) holds,

$$
\begin{aligned}
S_B &\geq S^{2,3,7,11,*}_{3,5,6}+f_2+f_3+f_7+f_{11}-S^{2,3,7,11,*}_{3,5,6}\\
&= f_2+f_3+f_7+f_{11}.
\end{aligned}
$$

5. If ¬(II) holds,

$$
\begin{aligned}
S_B &\geq S^{2,3,7,11,*}_{3,5,6}+f_1+f_2+f_4-L_B+L^{1,7,11,*}_{3,5,6}\\
&= f_1+f_2+f_4-L_B+S^{2,7,11,*}_{5,6}+L^{1,7,11,*}_{5,6}+S^3_3+S^{2,7,11,*}_3+S^3_{5,6}+L^{1,7,11,*}_3\\
&\geq f_1+f_2+f_4-L_B+S^{2,7,11,*}_{5,6}+L^{1,7,11,*}_{5,6}+\frac{1}{2}f_3\\
&\quad-\frac{1}{2}\min(f_3,L^{1,7,11,*}_3+S^{2,7,11,*}_3+S^3_{1,2,4,5,6})+S^{2,7,11,*}_3+S^3_{5,6}+L^{1,7,11,*}_3\\
&= f_1+f_2+f_4-L_B+S^{2,7,11,*}_{5,6}+L^{1,7,11,*}_{5,6}+\frac{1}{2}f_3\\
&\quad-\frac{1}{2}\min(f_3,L^{1,7,11,*}_3+S^{2,7,11,*}_3+S^3_{1,2,4,5,6})+\left(\frac{1}{2}+\frac{1}{2}\right)S^{2,7,11,*}_3\\
&\quad+\left(\frac{1}{2}+\frac{1}{2}\right)S^3_{5,6}+\left(\frac{1}{2}+\frac{1}{2}\right)L^{1,7,11,*}_3+\left(\frac{1}{2}-\frac{1}{2}\right)S^3_{1,2,4}\\
&= f_1+f_2+f_4-L_B+S^{2,7,11,*}_{5,6}+L^{1,7,11,*}_{5,6}+\frac{1}{2}f_3\\
&\quad+\frac{1}{2}(L^{1,7,11,*}_3+S^{2,7,11,*}_3+S^3_{1,2,4,5,6})\\
&\quad-\frac{1}{2}\min(f_3,L^{1,7,11,*}_3+S^{2,7,11,*}_3+S^3_{1,2,4,5,6})\\
&\quad+\frac{1}{2}S^{2,7,11,*}_3+\frac{1}{2}S^3_{5,6}+\frac{1}{2}L^{1,7,11,*}_3-\frac{1}{2}S^3_{1,2,4}\\
&\geq f_1+f_2+\frac{1}{2}f_3+f_4-L_B+\frac{1}{2}L^{1,7,11,*}_3+L^{1,7,11,*}_{5,6}+\frac{1}{2}S^{2,7,11,*}_3\\
&\quad+S^{2,7,11,*}_{5,6}+\frac{1}{2}S^3_{5,6}-\frac{1}{2}S^3_{1,2,4}\\
&\geq f_1+f_2+\frac{1}{2}f_3+f_4-L_B-\frac{1}{2}S^3_{1,2,4}.
\end{aligned}
$$

6. If (III) holds,

$$
\begin{aligned}
T_B &\geq T^{2,4,5,11,*}_{5,6}+f_2+2f_4+f_5+2f_{11}-T^{2,4,5,11}_{5,6}\\
&= f_2+2f_4+f_5+2f_{11}.
\end{aligned}
$$

7. If ¬ (III) holds,

$$
\begin{aligned}
T_B &\geq T^{2,4,5,11,*}_{5,6}+f_{1,\dots,4}-(L_B-L^{1,7,11,*}_{5,6})-(S_B-S^{2,3,7,11,*}_{5,6})\\
&\geq f_{1,\dots,4}-(L_B-L^{1,7,11,*}_{5,6})-(S_B-S^{2,3,7,11,*}_{5,6})\\
&\geq f_{1,\dots,4}-L_B-S_B.
\end{aligned}
$$

Based on the bounds on L_B, S_B, T_B, and N_{LLS}, the proof of Lemma 18 is completed by an case analysis.

Case 1. (II) holds; (III) holds.

$$\begin{aligned}
\mathcal{K} &= f_{1,\ldots,6} + \frac{1}{2}(f_1 + 2f_7 + f_{11} - L_B) \\
&\quad + \frac{1}{3}(f_2 + f_3 + f_7 + f_{11} - S_B - N_{LLS}) \\
&\quad + \frac{1}{4}(f_2 + 2f_4 + f_5 + 2f_{11} - T_B) \\
&\le f_{1,\ldots,6} + \frac{1}{2}(f_1 + 2f_7 + f_{11} - L_B) \\
&\quad + \frac{1}{3}(f_2 + f_3 + f_7 + f_{11} - (f_2 + f_3 + f_7 + f_{11}) - N_{LLS}) \\
&\quad + \frac{1}{4}(f_2 + 2f_4 + f_5 + 2f_{11} - (f_2 + 2f_4 + f_5 + 2f_{11})) \\
&= f_{1,\ldots,6} + \frac{1}{2}(f_1 + 2f_7 + f_{11} - L_B) - \frac{1}{3}N_{LLS} \\
&\le f_{1,\ldots,6} + \frac{1}{2}\left(f_1 + 2f_7 + f_{11} - \frac{1}{2}f_1\right) \\
&= \frac{5}{4}f_1 + f_{2,\ldots,7} + \frac{1}{2}f_{11} \\
&\le \frac{5}{4}f_{1,\ldots,7,11} + 1.
\end{aligned}$$

Case 2. (II) holds; ¬(III) holds.

$$\begin{aligned}
\mathcal{K} &= f_{1,\ldots,6} + \frac{1}{2}(f_1 + 2f_7 + f_{11} - L_B) \\
&\quad + \frac{1}{3}(f_2 + f_3 + f_7 + f_{11} - S_B - N_{LLS}) \\
&\quad + \frac{1}{4}(f_2 + 2f_4 + f_5 + 2f_{11} - T_B) \\
&\le f_{1,\ldots,6} + \frac{1}{2}(f_1 + 2f_7 + f_{11} - L_B) \\
&\quad + \frac{1}{3}(f_2 + f_3 + f_7 + f_{11} - S_B - N_{LLS}) \\
&\quad + \frac{1}{4}(f_2 + 2f_4 + f_5 + 2f_{11} - (f_1 + f_2 + f_3 + f_4 - L_B - S_B)) \\
&= \frac{5}{4}f_1 + \frac{4}{3}f_2 + \frac{13}{12}f_3 + \frac{5}{4}f_4 + \frac{5}{4}f_5 + f_6 + \frac{4}{3}f_7 + \frac{4}{3}f_{11} \\
&\quad - \frac{1}{4}L_B - \frac{1}{12}S_B - \frac{1}{3}N_{LLS} \\
&\le \frac{5}{4}f_1 + \frac{4}{3}f_2 + \frac{13}{12}f_3 + \frac{5}{4}f_4 + \frac{5}{4}f_5 + f_6 + \frac{4}{3}f_7 + \frac{4}{3}f_{11} \\
&\quad - \frac{1}{4}L_B - \frac{1}{12}(f_2 + f_3 + f_7 + f_{11}) - \frac{1}{3}N_{LLS} \\
&\le \frac{5}{4}f_{1,2} + f_3 + \frac{5}{4}f_{4,5} + f_6 + \frac{5}{4}f_{7,11} \\
&\le \frac{5}{4}f_{1,\ldots,7,11} + 1.
\end{aligned}$$

Case 3. ¬(II) holds; ¬(III) holds.

$$\begin{aligned}
\mathcal{K} &= f_{1,\dots,6} + \frac{1}{2}(f_1 + 2f_7 + f_{11} - L_B) \\
&\quad + \frac{1}{3}(f_2 + f_3 + f_7 + f_{11} - S_B - N_{LLS}) \\
&\quad + \frac{1}{4}(f_2 + 2f_4 + f_5 + 2f_{11} - T_B) \\
&\le f_{1,\dots,6} + \frac{1}{2}(f_1 + 2f_7 + f_{11} - L_B) \\
&\quad + \frac{1}{3}(f_2 + f_3 + f_7 + f_{11} - S_B - N_{LLS}) \\
&\quad + \frac{1}{4}(f_2 + 2f_4 + f_5 + 2f_{11} - (f_1 + f_2 + f_3 + f_4 + (L^{1,7,11,*}_{5,6} - L_B) + (S^{2,3,7,11,*}_{5,6} - S_B))) \\
&\quad = \frac{5}{4}f_1 + \frac{4}{3}f_2 + \frac{13}{12}f_3 + \frac{5}{4}f_{4,5} + f_6 + \frac{4}{3}f_{7,11} - \frac{1}{4}L_B - \frac{1}{12}S_B - \frac{1}{3}N_{LLS} \\
&\quad - \frac{1}{4}L^{1,7,11,*}_{5,6} - \frac{1}{4}S^{2,3,7,11,*}_{5,6} \\
&\le \frac{5}{4}f_1 + \frac{4}{3}f_2 + \frac{13}{12}f_3 + \frac{5}{4}f_{4,5} + f_6 + \frac{4}{3}f_{7,11} - \frac{1}{4}L_B \\
&\quad - \frac{1}{12}\left(f_1 + f_2 + \frac{1}{2}f_3 + f_4 - L_B - \frac{1}{2}S^3_{1,2,4}\right) - \frac{1}{3}N_{LLS} - \frac{1}{4}L^{1,7,11,*}_{5,6} - \frac{1}{4}S^{2,3,7,11,*}_{5,6} \\
&\le \frac{7}{6}f_1 + \frac{5}{4}f_2 + \frac{25}{24}f_3 + \frac{7}{6}f_4 + \frac{5}{4}f_5 + f_6 + \frac{4}{3}f_7 + \frac{4}{3}f_{11} - \frac{1}{6}L_B - \frac{1}{3}N_{LLS} \\
&\quad - \frac{1}{4}L^{1,7,11,*}_{5,6} - \frac{1}{4}S^{2,3,7,11,*}_{5,6} \\
&\le \frac{7}{6}f_1 + \frac{5}{4}f_2 + \frac{25}{24}f_3 + \frac{7}{6}f_4 + \frac{5}{4}f_5 + f_6 + \frac{4}{3}f_7 + \frac{4}{3}f_{11} - \frac{1}{6}L_B - \frac{1}{12}S^{7,11}_{5,6} - \frac{1}{3}N_{LLS}.
\end{aligned}$$

Case 3.1. (I) holds.

$$\begin{aligned}
\mathcal{K} &\le \frac{7}{6}f_1 + \frac{5}{4}f_2 + \frac{25}{24}f_3 + \frac{7}{6}f_4 + \frac{5}{4}f_5 + f_6 + \frac{4}{3}f_7 + \frac{4}{3}f_{11} - \frac{1}{6}(2f_7 + f_{11}) \\
&\le \frac{5}{4}f_{1,\dots,7,11} + 1.
\end{aligned}$$

Case 3.2. ¬(I) holds.

$$\begin{aligned}
\mathcal{K} &\le \frac{7}{6}f_1 + \frac{5}{4}f_2 + \frac{25}{24}f_3 + \frac{7}{6}f_4 + \frac{5}{4}f_5 + f_6 + \frac{4}{3}f_7 + \frac{4}{3}f_{11} \\
&\quad - \frac{1}{6}\left(\frac{1}{2}f_1 + f_2 - \frac{1}{2}L^1_2 + \frac{1}{2}L^{7,11}_{1,3,\dots,6}\right) - \frac{1}{12}S^{7,11}_{5,6} - \frac{1}{3}N_{LLS} \\
&= \frac{13}{12}f_1 + \frac{13}{12}f_2 + \frac{25}{24}f_3 + \frac{7}{6}f_4 + \frac{5}{4}f_5 + f_6 + \frac{4}{3}f_7 + \frac{4}{3}f_{11} + \frac{1}{12}L^1_2 - \frac{1}{12}L^{7,11}_{1,3,\dots,6} \\
&\quad - \frac{1}{12}S^{7,11}_{5,6} - \frac{1}{3}N_{LLS} \\
&\le \frac{7}{6}f_1 - \frac{1}{12}S^{7,11}_1 + \frac{1}{12}L^1_2 + \frac{7}{6}f_2 - \frac{1}{12}L^{7,11}_2 - \frac{1}{12}S^{7,11}_2 + \frac{9}{8}f_3 - \frac{1}{12}S^{7,11}_3 \\
&\quad + \frac{5}{4}f_4 - \frac{1}{12}S^{7,11}_4 + \frac{5}{4}f_5 + f_6 + \frac{4}{3}f_7 + \frac{4}{3}f_{11} - \frac{1}{12}L^{7,11}_{1,3,\dots,6} - \frac{1}{12}S^{7,11}_{5,6} - \frac{1}{3}N_{LLS} \\
&\le \frac{5}{4}f_1 + \frac{7}{6}f_2 + \frac{9}{8}f_3 + \frac{5}{4}f_4 + \frac{5}{4}f_5 + f_6 + \frac{4}{3}f_7 + \frac{4}{3}f_{11} - \frac{1}{12}L^{7,11}_{1,\dots,6} - \frac{1}{12}S^{7,11}_{1,\dots,6} - \frac{1}{3}N_{LLS}
\end{aligned}$$

$$\begin{aligned}
&\le \frac{5}{4}f_1+\frac{7}{6}f_2+\frac{9}{8}f_3+\frac{5}{4}f_4+\frac{5}{4}f_5+f_6+\frac{4}{3}f_7+\frac{4}{3}f_{11}-\frac{1}{12}L^{7,11}_{1,\dots,6}-\frac{1}{12}S^{7,11}_{1,\dots,6}\\
&\quad-\frac{1}{3}\left(\frac{1}{4}(\max(0,f_7-(L^7_{1,\dots,6}+S^7_{1,\dots,6}))+f_{11}-\max(L^{11}_{1,\dots,6},S^{11}_{1,\dots,6}))-1\right)\\
&=\frac{5}{4}f_1+\frac{7}{6}f_2+\frac{9}{8}f_3+\frac{5}{4}f_4+\frac{5}{4}f_5+f_6+\frac{4}{3}f_7+\frac{5}{4}f_{11}-\frac{1}{12}L^{7,11}_{1,\dots,6}-\frac{1}{12}S^{7,11}_{1,\dots,6}\\
&\quad-\frac{1}{12}\left(\max(0,f_7-(L^7_{1,\dots,6}+S^7_{1,\dots,6}))+\frac{1}{12}\max(L^{11}_{1,\dots,6},S^{11}_{1,\dots,6})\right)+\frac{1}{3}\\
&\le \frac{5}{4}f_{1,\dots,7,11}+1+\frac{1}{12}f_7-\frac{1}{12}\left(\max(0,f_7-(L^7_{1,\dots,6}+S^7_{1,\dots,6}))-\frac{1}{12}L^{7,11}_{1,\dots,6}-\frac{1}{12}S^{7,11}_{1,\dots,6}\right.\\
&\quad\left.+\frac{1}{12}\max(L^{11}_{1,\dots,6},S^{11}_{1,\dots,6})\right)\\
&\le \frac{5}{4}f_{1,\dots,7,11}+1+\frac{1}{12}\mathcal{K}',
\end{aligned}$$

where $\mathcal{K}'=f_7-(\max(0,f_7-L^7_{1,\dots,6}-S^7_{1,\dots,6})+L^7_{1,\dots,6}+S^7_{1,\dots,6})$.

Case 3.2.1. (IV) holds. $\mathcal{K}'=f_7-(f_7-L^7_{1,\dots,6}-S^7_{1,\dots,6}+L^7_{1,\dots,6}+S^7_{1,\dots,6})=0$.

Case 3.2.2. ¬(IV) holds. $\mathcal{K}'=f_7-L^7_{1,\dots,6}-S^7_{1,\dots,6}<0$.

Case 4. ¬(II) holds; (III) holds.

$$\begin{aligned}
\mathcal{K}&=f_{1,\dots,6}+\frac{1}{2}(f_1+2f_7+f_{11}-L_B)\\
&\quad+\frac{1}{3}(f_2+f_3+f_7+f_{11}-S_B-N_{LLS})\\
&\quad+\frac{1}{4}(f_2+2f_4+f_5+2f_{11}-T_B)\\
&\le f_{1,\dots,6}+\frac{1}{2}(f_1+2f_7+f_{11}-L_B)\\
&\quad+\frac{1}{3}\left(f_2+f_3+f_7+f_{11}-\left(f_1+f_2+\frac{1}{2}f_3+f_4-L_B\right.\right.\\
&\quad\left.\left.+\frac{1}{2}L^{1,7,11,*}_3+L^{1,7,11,*}_{5,6}+\frac{1}{2}S^{2,7,11,*}_3+S^{2,7,11,*}_{5,6}+\frac{1}{2}S^3_{5,6}-\frac{1}{2}S^3_{1,2,4}\right)-N_{LLS}\right)\\
&\quad+\frac{1}{4}(f_2+2f_4+f_5+2f_{11}-(f_2+2f_4+f_5+2f_{11}))\\
&=\frac{7}{6}f_1+f_2+\frac{7}{6}f_3+\frac{2}{3}f_4+f_{5,6}+\frac{4}{3}f_7+\frac{5}{6}f_{11}-\frac{1}{6}L_B-\frac{1}{3}N_{LLS}+\frac{1}{6}S^3_{1,2,4}\\
&\quad-\frac{1}{3}\left(\frac{1}{2}L^{1,7,11,*}_3+L^{1,7,11,*}_{5,6}+\frac{1}{2}S^{2,7,11,*}_3+S^{2,7,11,*}_{5,6}+\frac{1}{2}S^3_{5,6}\right)\\
&\le \frac{7}{6}f_1+f_2+\frac{7}{6}f_3+\frac{2}{3}f_4+f_{5,6}+\frac{4}{3}f_7+\frac{5}{6}f_{11}-\frac{1}{6}L_B+\frac{1}{6}S^3_{1,2,4}-\frac{1}{12}S^7_{3,5,6}-\frac{1}{3}N_{LLS}.
\end{aligned}$$

Case 4.1. (I) holds.

$$\begin{aligned}
\mathcal{K}&\le \frac{7}{6}f_1+f_2+\frac{7}{6}f_3+\frac{2}{3}f_4+f_{5,6}+\frac{4}{3}f_7+\frac{5}{6}f_{11}-\frac{1}{6}(2f_7+f_{11})+\frac{1}{6}S^3_{1,2,4}\\
&\le \frac{5}{4}f_1-\frac{1}{12}S^3_1+\frac{13}{12}f_2-\frac{1}{12}S^3_2+\frac{5}{4}f_3-\frac{1}{12}S^3_{1,2,4}+\frac{3}{4}f_4-\frac{1}{12}S^3_4\\
&\quad+f_{5,6,7}+\frac{2}{3}f_{11}+\frac{1}{6}S^3_{1,2,4}
\end{aligned}$$

$$= \frac{5}{4}f_1 + \frac{13}{12}f_2 + \frac{5}{4}f_3 + \frac{3}{4}f_4 + f_{5,6,7} + \frac{2}{3}f_{11}$$
$$\leq \frac{5}{4}f_{1,\ldots,7,11} + 1.$$

Case 4.2. ¬ (I) holds.

$$\begin{aligned}
\mathcal{K} &\leq \frac{7}{6}f_1 + f_2 + \frac{7}{6}f_3 + \frac{2}{3}f_4 + f_{5,6} + \frac{4}{3}f_7 + \frac{5}{6}f_{11} - \frac{1}{6}\left(\frac{1}{2}f_1 + f_2 - \frac{1}{2}L_2^1 + \frac{1}{2}L_{1,3,\ldots,6}^7\right) \\
&\quad + \frac{1}{6}S_{1,2,4}^3 - \frac{1}{12}S_{3,5,6}^7 - \frac{1}{3}N_{LLS} \\
&\leq \frac{5}{4}f_1 - \frac{1}{12}S_1^3 - \frac{1}{12}S_1^7 + \frac{7}{6}f_2 - \frac{1}{12}L_2^1 - \frac{1}{12}S_2^3 - \frac{1}{12}L_2^7 - \frac{1}{12}S_2^7 \\
&\quad + \frac{5}{4}f_3 - \frac{1}{12}S_{1,2,4}^3 + \frac{3}{4}f_4 - \frac{1}{12}S_4^3 - \frac{1}{12}S_4^7 + f_{5,6} + \frac{4}{3}f_7 + \frac{5}{6}f_{11} \\
&\quad + \frac{1}{12}L_2^1 - \frac{1}{12}L_{1,3,\ldots,6}^7 + \frac{1}{6}S_{1,2,4}^3 - \frac{1}{12}S_{3,5,6}^7 - \frac{1}{3}N_{LLS} \\
&= \frac{5}{4}f_1 + \frac{7}{6}f_2 + \frac{5}{4}f_3 + \frac{3}{4}f_4 + f_{5,6} + \frac{4}{3}f_7 + \frac{5}{6}f_{11} - \frac{1}{12}L_{1,\ldots,6}^7 - \frac{1}{12}S_{1,\ldots,6}^7 - \frac{1}{3}N_{LLS} \\
&\leq \frac{5}{4}f_1 + \frac{7}{6}f_2 + \frac{5}{4}f_3 + \frac{3}{4}f_4 + f_{5,6} + \frac{4}{3}f_7 + \frac{5}{6}f_{11} - \frac{1}{12}L_{1,\ldots,6}^7 - \frac{1}{12}S_{1,\ldots,6}^7 \\
&\quad - \frac{1}{3}\left(\frac{1}{4}(\max(0, f_7 - (L_{1,\ldots,6}^7 + S_{1,\ldots,6}^7)) + f_{11} - \max(L_{1,\ldots,6}^{11}, S_{1,\ldots,6}^{11})) - 1\right) \\
&\leq \frac{5}{4}f_1 + \frac{7}{6}f_2 + \frac{5}{4}f_3 + \frac{3}{4}f_4 + f_{5,6} + \frac{4}{3}f_7 + \frac{11}{12}f_{11} - \frac{1}{12}L_{1,\ldots,6}^{7,11} - \frac{1}{12}S_{1,\ldots,6}^{7,11} \\
&\quad - \frac{1}{12}\left(\max(0, f_7 - (L_{1,\ldots,6}^7 + S_{1,\ldots,6}^7)) + \frac{1}{12}\max(L_{1,\ldots,6}^{11}, S_{1,\ldots,6}^{11})\right) + \frac{1}{3} \\
&\leq \frac{5}{4}f_{1,\ldots,7,11} + 1 + \frac{1}{12}f_7 - \frac{1}{12}\left(\max(0, f_7 - (L_{1,\ldots,6}^7 + S_{1,\ldots,6}^7)) - \frac{1}{12}L_{1,\ldots,6}^{7,11} - \frac{1}{12}S_{1,\ldots,6}^{7,11}\right. \\
&\quad \left. + \frac{1}{12}\max(L_{1,\ldots,6}^{11}, S_{1,\ldots,6}^{11})\right) \\
&\leq \frac{5}{4}f_{1,\ldots,7,11} + 1 + \frac{1}{12}\mathcal{K}',
\end{aligned}$$

where $\mathcal{K}' = f_7 - (\max(0, f_7 - L_{1,\ldots,6}^7 - S_{1,\ldots,6}^7) + L_{1,\ldots,6}^7 + S_{1,\ldots,6}^7)$.

Case 4.2.1. (IV) holds. $\mathcal{K}' = f_7 - (f_7 - L_{1,\ldots,6}^7 - S_{1,\ldots,6}^7 + L_{1,\ldots,6}^7 + S_{1,\ldots,6}^7) = 0$.

Case 4.2.2. ¬(IV) holds. $\mathcal{K}' = f_7 - L_{1,\ldots,6}^7 - S_{1,\ldots,6}^7 < 0$. □

4.1.5. Consideration of arbitrary lists. We established that the competitive ratio of *MMP* packings of lists of non-M-items is $\frac{5}{4}$. We now use the M-thoroughness of *MMP*, established in Lemma 2, to establish $\frac{5}{4}$ as the competitive ratio of *MMP* for arbitrary lists.

THEOREM 1. *Let L be a list of items of arbitrary size from $(0, 1]$. Then*

$$MMP(L) \leq \frac{5}{4}OPT(L) + 3.$$

Proof. We consider the following two cases.

Case 1. There are no M-bins in any *MMP* packing of L.

We may disregard all of the M-items from L and directly apply Lemma 18.

Case 2. There exists at least one *MMP* packing of L that contains an M-bin.

Fix an arbitrary *MMP* packing P_L of L that contains an M-bin. Let P_0 and *Aux* be the regular packing and the auxiliary storage of P_L, respectively. By M-thoroughness of *MMP* (Lemma 2), all of the bins of the regular packing have a level of at least $\frac{4}{5}$, except for, possibly, the last bin. This immediately implies that the regular packing P_0 requires at most $\frac{5}{4}OPT(L)+1$ bins to pack all of the items of L, *except* for the excess items from *Aux*: at most one L-item, two S-items, and three T-items, all of which can be packed into two bins. Thus the overall number of bins required by P_L is at most $\frac{5}{4}OPT(L)+3$. □

4.2. The running time of *MMP* is $\Theta(\log n)$.

THEOREM 2. MMP *can be implemented to run in* $\Theta(\log n)$ *uniform running time per* Insert/Delete *operation.*

Proof. Recall from sections 3.2.1 and 3.2.2 that each of the functions utilized by *MMP* runs in logarithmic time. It is easy to see from the code that there is a small bounded number of calls to these functions preceding each packing of an item and in particular each insertion and deletion of a whole bin. Note that this in itself is not sufficient to prove the logarithmic running time of *MMP*. However, it does suffice to show that the number of bins that are inserted and deleted as a consequence of a single *Insert* or *Delete* operation is bounded by a constant.

The proof proceeds by a case analysis. The nontrivial cases are the following: *Insert* of a B-item, *Insert* of an L-item, *Insert* of an S-item, and *Insert* of a T-item; *Delete* of a B-item, *Delete* of an L-item, *Delete* of an S-item, and, finally, *Delete* of a T-item.

We only show the case of an *Insert* of an S-item and note that the proofs are quite analogous for each of the other cases.

To facilitate a simple exposition of the proof, we denote items by upper case letters corresponding to their type (e.g., a T-item will be denoted as T in this proof). Items that could form an LLS-coalition will be decorated with an apostrophe.

What is the longest possible sequence of bin insertions/deletions an *Insert* of an S-item S' could generate?

We consider two subcases. First, S' is inserted so as to form an LLS' bin. Second, S' is inserted into a B-bin. It can be shown that the more difficult case is the second case and with that case the subcase leading to a BS'T bin is more difficult than the subcase leading to a BS' bin.

We proceed with an analysis of the case of a BS'T bin. Before the *Insert* of S', the content of *Aux* was in the worst case: {L, 2S, 3T}.

To construct BS'T, S' may have in the worst case evicted two T-items from a BTT bin and "recruited" a T-item from another BTT bin. At this moment, one bin (BS'T) is inserted, one bin (BTT) is deleted, and the content of *Aux* is in the worst case: {B, L, 2S, 6T}.

The B-item from *Aux* could now, in the worst case, form a BTT bin with two T-items, each from a BT bin. At this moment, one bin (BTT) is inserted and two bins (BT, BT) are deleted. *Aux* is now {2B, L, 2S, 8T}.

The two B-items from *Aux* could now, in the worst case, each form a BT bin with a T-item from an LLT bin. At this moment, two bins (BT, BT) are inserted, and two bins (LLT, LLT) are deleted. *Aux* is now {5L, 2S, 8T}.

By LLS-maximality of *MMP*, none of the L-items from *Aux* can form LLS-coalitions. Thus, in the worst case, each of the five L-items from *Aux* can form

an LLT bin with a T-item from an SSST bin. At this moment, five LLT bins are inserted, and five SSST bins are deleted. *Aux* is now {L, 17S, 8T}.

Again, by LLS-maximality of *MMP*, none of the S-items from *Aux* can form LLS-coalitions. Thus, in the worst case, five SSST bins are formed out of 15 S-items from *Aux*, coupled with five T-items, each from a TTTT bin. At this moment, five SSST bins are inserted, and five TTTT bins are deleted. *Aux* is now {L, 2S, 23T}.

Finally, five TTTT bins are inserted, each containing four T-items from *Aux*. At the conclusion, the content of *Aux* is {L, 2S, 3T}.

Thus, the overall number of inserted bins is 18, and the overall number of deleted bins is 15. Note that we ignored the possibility that M-bins may be deleted to execute *top_with*_M, and at the end of the *Insert* of S' *reload*_M. Clearly, consideration of M-bins could contribute only very few inserted or deleted M-bins. For example, a (generous) upper bound on the number of deleted M-bins is 18. □

5. Conclusion. We have studied the problem of maintaining an approximate solution for *one-dimensional bin packing* when items may arrive and depart dynamically and when the packing may be rearranged to accommodate arriving and departing items. Our main result is a fully dynamic bin packing algorithm *MMP* that is $\frac{5}{4}$-competitive and requires $\Theta(\log n)$ uniform running time per *Insert/Delete* operation. Relative to the best practical off-line algorithms, our algorithm is the best possible with respect to its running time and is nearly approximation-competitive with those algorithms (losing but a factor of $\frac{1}{15}$ to the best of those [13]).

The major unresolved issue is whether there exist fully dynamic bin packing algorithms (accommodating both *Inserts* and *Deletes*) that attain better competitive ratios, i.e., are there algorithms that are α-competitive for some $\alpha < \frac{5}{4}$, and require $o(n)$ time per operation. Here, both uniform and amortized algorithms are of interest.

Other unresolved issues are (1) what is the nature of the trade-off between running times and competitive ratios of fully dynamic bin packing algorithms for bin packing (both uniform and amortized), and (2) is there a competitive ratio for which there are no fully dynamic approximation algorithms for bin packing featuring sublinear running times (uniform or amortized)?

REFERENCES

[1] R. J. Anderson, E. W. Mayr, and M. K. Warmuth (1983), *Parallel approximation algorithms for bin packing*, Inform. and Comput., 82, pp. 262–277.

[2] E. G. Coffman, M. R. Garey, and D. S. Johnson (1983), *Dynamic bin packing*, SIAM J. Comput., 12, pp. 227–258.

[3] E. G. Coffman, M. R. Garey, and D. S. Johnson (1984), *Approximation algorithms for bin packing: An updated survey*, in Algorithm Design for Computer System Design, G. Ausiello, M. Lucertini, and P. Serafini, eds., Springer-Verlag, New York, pp. 49–106.

[4] W. Fernandez de la Vega and G. S. Lueker (1981), *Bin packing can be solved within* $1+\epsilon$ *in linear time*, Combinatorica, 1, pp. 349–355.

[5] D. K. Friesen and M. A. Langston (1991), *Analysis of a compound bin packing algorithm*, SIAM J. Discrete Math., 4, pp. 61–79.

[6] G. Gambosi, A. Postiglione, and M. Talamo (1990), *New algorithms for on–line bin packing*, in Algorithms and Complexity, Proceedings of the First Italian Conference, G. Aussiello, D. P. Bovet, and R. Petreschi, eds., World Scientific, Singapore, pp. 44–59.

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

[8] Z. Ivković (1995), *Fully Dynamic Approximation Algorithms*, Ph.D. thesis, University of Delaware, Newark, DE.

[9] Z. Ivković and E. L. Lloyd (1993), *Fully dynamic maintenance of vertex cover*, in Proc. 19th International Workshop on Graph–Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, Springer-Verlag, New York, pp. 99–111.

[10] D. S. Johnson (1973), *Near–Optimal Bin Packing Algorithms*, Ph.D. thesis, MIT, Cambridge, MA.

[11] D. S. Johnson (1974), *Fast algorithms for bin packing*, J. Comput. System Sci., 8, pp. 272–314.

[12] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham (1974), *Worst–case performance bounds for simple one-dimensional packing algorithms*, SIAM J. Comput., 3, pp. 299–325.

[13] D. S. Johnson and M. R. Garey (1985), *A* 71/60 *theorem for bin packing*, J. Complexity, 1, pp. 65–106.

[14] N. Karmarkar and R. M. Karp (1982), *An efficient approximation scheme for the one-dimensional bin-packing problem*, in Proc. 23rd IEEE Symposium on Foundations of Computer Science, pp. 312–320.

[15] R. M. Karp (1972), *Reducibility among combinatorial problems*, in Complexity of Computations, R. E. Miller and J. W. Thatcher, eds., Plenum, New York, pp. 85–103.

[16] P. N. Klein and S. Sairam (1993), *Fully Dynamic Approximation Schemes for Shortest Path Problems in Planar Graphs*, manuscript.

[17] C. C. Lee and D. T. Lee (1985), *A simple on-line bin-packing algorithm*, J. Assoc. Comput. Mach., 3, pp. 562–572.

[18] P. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee (1989), *On-line bin-packing in linear time*, J. Algorithms, 3, pp. 305–326.

[19] A. C.–C. Yao (1980). *New algorithms for bin packing*, J. Assoc. Comput. Mach., 27, pp. 207–277.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 612–636

DYNAMIC TREES AND DYNAMIC POINT LOCATION*

MICHAEL T. GOODRICH† AND ROBERTO TAMASSIA‡

Abstract. This paper describes new methods for maintaining a point-location data structure for a dynamically changing monotone subdivision $\mathcal{S}$. The main approach is based on the maintenance of two interlaced spanning trees, one for $\mathcal{S}$ and one for the graph-theoretic planar dual of $\mathcal{S}$. Queries are answered by using a centroid decomposition of the dual tree to drive searches in the primal tree. These trees are maintained via the link-cut trees structure of Sleator and Tarjan [*J. Comput. System Sci.*, 26 (1983), pp. 362–381], leading to a scheme that achieves vertex insertion/deletion in $O(\log n)$ time, insertion/deletion of k-edge monotone chains in $O(\log n + k)$ time, and answers queries in $O(\log^2 n)$ time, with $O(n)$ space, where n is the current size of subdivision $\mathcal{S}$. The techniques described also allow for the dual operations *expand* and *contract* to be implemented in $O(\log n)$ time, leading to an improved method for spatial point location in a 3-dimensional convex subdivision. In addition, the interlaced-tree approach is applied to on-line point location (where one builds $\mathcal{S}$ incrementally), improving the query bound to $O(\log n \log\log n)$ time and the update bounds to $O(1)$ amortized time in this case. This appears to be the first on-line method to achieve a polylogarithmic query time and constant update time.

Key words. computational geometry, point location, centroid decomposition, dynamic data structures, on-line algorithms

AMS subject classifications. 68U05, 68Q25, 68P05, 68P10

PII. S0097539793254376

1. Introduction.

An exciting direction in algorithmic research has been to show how one can efficiently maintain various properties of a combinatoric or geometric structure while updating that structure in a dynamic fashion (e.g., see [13]). A problem with tremendous potential for dynamization is planar point location, a classic problem in computational geometry (e.g., see [1, 17, 28, 33, 37]). Given a subdivision S of the plane into "cells," described by a total of n line segments, the problem is to preprocess S to allow for efficiently naming the cell containing a query point p. An important special case of the point-location problem occurs when each face in the planar subdivision is a monotone polygon with respect to the y-axis; that is, the boundary of each face is intersected at most twice by any horizontal line. Given such a subdivision, Kirkpatrick [26] shows that one can construct an $O(n)$-space structure in $O(n)$ time that allows $O(\log n)$-time point-location queries. Edelsbrunner, Guibas, and Stolfi [18] show that one can achieve these same bounds by applying the fractional cascading paradigm of Chazelle and Guibas [8, 9] to the chain method of Lee and Preparata [27]. Cole [15] and Sarnak and Tarjan [41] independently show that one can also achieve these bounds after $O(n \log n)$ preprocessing by applying a per-

*Received by the editors August 26, 1993; accepted for publication (in revised form) August 16, 1996; published electronically August 4, 1998. A preliminary version of this paper was presented at the 23*rd Annual ACM Symposium on the Theory of Computing*, New Orleans, 1991.

http://www.siam.org/journals/sicomp/28-2/25437.html

†Department of Computer Science, The Johns Hopkins University, Baltimore, MD 21218 (goodrich@cs.jhu.edu). This research was supported in part by National Science Foundation grants CCR-8810568, CCR-9003299, and CCR-9625289 and by U.S. Army Research Office grant DAAH04-96-1-0013.

‡Department of Computer Science, Brown University, Providence, RI 02912-1910 (rt@cs.brown.edu). This research supported in part by National Science Foundation grants CCR-9007851 and CCR-9423847, by U.S. Army Research Office grants DAAL03-91-G-0035 and DAAH04-96-1-0013, and by Office of Naval Research and the Advanced Research Projects Agency contract N00014-91-J-4052, ARPA order 8225.

sistence technique (e.g., see Driscoll et al. [16]) to a simple plane-sweeping procedure (as an example of a static→dynamic→static conversion). (See [17, 28, 37] for other references on this important problem.)

We are interested in maintaining a monotone subdivision dynamically, subject to edge insertion and deletion, vertex insertion and deletion, as well as the insertion or deletion of a monotone chain of k edges. In addition, we are also interested in operations that are duals to edge insertion and deletion, as in the framework of Guibas and Stolfi [25], where we allow for vertex expansion and contraction: an *expansion* splits a vertex v into two new vertices connected by an edge, and a *contraction* merges two adjacent vertices into a new vertex. These operations are useful in applying a dynamic point location to spatial point location in 3-dimensional subdivisions [40] via persistence [16].

1.1. Previous work. Before we describe our main results, let us briefly review previous work on dynamic point location, which we summarize in Table 1. Early work on dynamic point location includes a method by Overmars [35], which is based on a segment-tree [4] approach to planar-point location, and achieves an $O(\log^2 n)$ query and update time with $O(n \log n)$ space. Fries [20] and Fries, Mehlhorn, and Näher [21] present a data structure with $O(n)$ space, $O(\log^2 n)$ query time, and $O(\log^4 n)$ amortized update time (for edge insertion/deletion only), using an approach based on the static chain method of Lee and Preparata [27]. Neither of these methods seems to extend to the dual update operations of expand and contract, however.

TABLE 1

Previous results for dynamic point location. N denotes the number of possible y-coordinates for edge endpoints in the subdivision. Also, we use $\bar{O}(*)$ to denote an amortized bound.

Type	Queries	Insert	Delete
General [3]	$O(\log n \log\log n)$	$\bar{O}(\log n \log\log n)$	$\bar{O}(\log^2 n)$
Connected [11]	$O(\log^2 n)$	$O(\log n)$	$O(\log n)$
Connected [12]	$O(\log n)$	$\bar{O}(\log^3 n)$	$\bar{O}(\log^3 n)$
Monotone [14]	$O(\log n)$	$\bar{O}(\log^2 n)$	$\bar{O}(\log^2 n)$
Convex [39]	$O(\log n + \log N)$	$O(\log n \log N)$	$O(\log n \log N)$
Staircase [2]	$O(\log n)$	$\bar{O}(\log n)$	$\bar{O}(\log n)$

Preparata and Tamassia [38] have given techniques for maintaining monotone subdivisions that are also based on this chain method, but they improve the bounds of Fries, Melhorn, and Näher by representing the chains topologically rather than geometrically. In their scheme, inserting/deleting vertices on edges requires $O(\log n)$ time, and inserting/deleting monotone chains of edges requires $O(\log^2 n + k)$ time. Moreover, as shown in [40], their scheme can be extended to the dual update operations, which leads, via persistence [16], to a data structure for spatial point location that uses $O(N \log^2 N)$ space, requires $O(N \log^2 N)$ processing time, and allows for queries to be answered in $O(\log^2 N)$ time, where N is the size of the 3-dimensional subdivision.

Cheng and Janardan [11] present two methods for dynamic planar point location that improve the time of edge updates. In their Scheme I they achieve $O(\log^2 n)$ query time, $O(\log n)$ time for inserting/deleting a vertex, and $O(k \log n)$ time for inserting/deleting a chain of k edges, and in their Scheme II they achieve $O(\log n \log\log n + k)$ time for inserting/deleting monotone chains, at the expense of increasing vertex insertion/deletion time to $O(\log n \log\log n)$ and increasing the query time to $O(\log^2 n \log\log n)$. Both of their methods are based on a search strategy derived

from the priority search tree data structure of McCreight [30]. They dynamize this approach with the $BB(\alpha)$ tree data structure (e.g., see [32]) using the approach of Willard and Lueker [43] to spread local updates over future operations and the method of Overmars [34] to perform global rebuilding (at the same time) before the "current" data structure becomes too unbalanced. Their methods do not seem to extend to the dual update operations, however, nor does it seem possible to improve their bounds for the on-line case.

The dynamic data structure by Baumgarten, Jung, and Mehlhorn [3] combines interval trees, segment trees, fractional cascading, and the data structure of [11]. It achieves $O(n)$ space, $O(\log n \log\log n)$ query and insertion time, and $O(\log^2 n)$ deletion time, where the time bounds for updates are amortized.

Chiang and Tamassia [14] present a dynamic data structure for monotone subdivisions, which is based on the static trapezoid method of Preparata [36] and extends previous work by Preparata and Tamassia [39] on dynamic point location in convex subdivisions with vertices on a fixed set of lines. The operations supported are insertion and deletion of vertices and edges and horizontal translation of vertices. They show how to achieve queries in $O(\log n)$ time, while requiring $O(\log^2 n)$ time for updates. The space requirement for their method is $O(n \log n)$. Finally, Atallah, Goodrich, and Ramaiyer [2] show how to apply a new data structure, which they call *biased finger trees* to achieve an $O(\log n)$ query time and $O(\log n)$ amortized update time for a fairly restricted class of subdivisions known as staircase subdivisions, where each face is bounded above and below by "staircase" polygonal chains.

In related work, Chiang, Preparata, and Tamassia [12] have shown that one can achieve an $O(\log n)$ query time in a dynamic environment that allows for ray-shooting queries and subdivision updates in $O(\log^3 n)$ time, and Goodrich and Tamassia [23] show how to maintain a similar environment so as to achieve $O(\log^2 n)$ time for all updates and queries (using a method built upon the scheme of the present paper).

1.2. Our results. In this paper we show how to dynamically maintain a monotone subdivision so as to achieve $O(\log^2 n)$ query time, $O(\log n)$ time for vertex insertion/deletion, and $O(\log n + k)$ time for the insertion/deletion of a monotone chain of k edges. Our methods are based on the maintenance of two interlaced spanning trees, one for the subdivision and one for its graph-theoretic dual, to answer queries. Queries are performed by using a centroid decomposition of the dual tree to drive searches in the primal tree. We dynamize this approach using the edge-ordered dynamic tree data structure of Eppstein et al. [19], which is an extension of the link-cut trees data structure of Sleator and Tarjan [42]. We use the "built-in" operations of *link*, *cut*, *split*, and *merge* to implement both our updates and queries. Our methods improve the previous bounds for dynamically maintaining monotone subdivisions.

We also show how to extend our approach to implement the dual operations of *expand* and *contract*, which, in turn, leads to an improved data structure for spatial point location via the persistence paradigm of Driscoll et al. [16], where one dynamizes the problem to a 3-dimensional space sweep that uses our data structure to maintain the current "slice." This leads to an $O(N \log N)$ space data structure that requires only $O(N \log N)$ preprocessing time while achieving an $O(\log^2 N)$ query time, for a 3-dimensional convex subdivision with N facets, which improves the space and preprocessing of the previous method [40] by a $\log N$ factor.

Finally, we show how to apply our approach to on-line planar point location, where one builds a monotone subdivision incrementally. In this case we show how to maintain the centroid decomposition of the dual tree explicitly (in a $BB(\alpha)$ tree [32])

and apply a simple version of the fractional cascading paradigm of Chazelle and Guibas [8, 9] to improve the query time to $O(\log n \log\log n)$ while also improving the complexity of updates to $O(1)$ amortized time. We believe this is the first on-line point location method to achieve a polylogarithmic query time and constant update time.

2. Preliminaries.

2.1. Monotone subdivisions. A *(planar) subdivision* $\mathcal{S}$ is a partition of the plane into polygons, called the *regions* of $\mathcal{S}$. We assume that $\mathcal{S}$ has one unbounded region, called the external region. A subdivision $\mathcal{S}$ is generated by a planar graph embedded in the plane such that the edges are straight-line segments. We assume a standard representation for the subdivision $\mathcal{S}$ such as doubly connected edge lists [37].

A *monotone chain* is a polygonal chain such that each horizontal line intersects it in at most one point. A polygon is *monotone* if its boundary is partitionable into two monotone chains. A *monotone subdivision* is such that all its regions are monotone polygons (even the external region). A *triangulation* is a subdivision such that the boundary of each region has three edges.

Let us orient each edge of a monotone subdivision $\mathcal{S}$ by decreasing ordinate, i.e., so that it "points down." Because each face in $\mathcal{S}$ is a monotone polygon, in orienting the edges of $\mathcal{S}$ in this way we obtain a planar st-graph, i.e., a planar acyclic digraph with exactly one source (vertex without incoming edges) and one sink (vertex without outgoing edges). The source s and sink t of $\mathcal{S}$ are the highest and lowest vertices of $\mathcal{S}$, respectively. The *left chain* of a region r of $\mathcal{S}$ is the monotone chain on the boundary of r such that r is on the left side of it when traversed from top to bottom. The *right chain* is similarly defined. Note that according to this definition the left (resp., right) chain of the external region appears on the right (resp., left) boundary of the subdivision.

2.2. Centroid decomposition. Let T be free tree with n vertices of degree at most 3. A *centroid edge* of T is an edge e whose removal partitions T into two trees of size at most $1 + 2n/3$ each. It is well known that if $n > 1$, such an edge exists and can be found in time $O(n)$ (e.g., see [6, 31]).

A *centroid decomposition tree* for T is a rooted binary tree B recursively defined as follows: if T has a single vertex v, then B consists of a single leaf node that stores vertex v. Otherwise, let e be a centroid edge of T, and let T' and T'' be the trees that result when removing e from T. The root of B stores edge e, and the left and right subtrees of B are centroid trees for T' and T'', respectively. The centroid decomposition tree B has $O(\log n)$ height and can be constructed in $O(n)$ time (e.g., see [10, 24]).

2.3. Dynamic trees. Dynamic trees [42] are a versatile dynamic data structure for maintaining a forest of rooted trees. We shall use an extension of dynamic trees, called *edge-ordered dynamic trees* [19].

An edge-ordered tree is a rooted tree in which a cyclic order is imposed on the edges incident on each node (including the edge to the parent). The circular sequence of edges incident to node μ is called the *edge ring* of μ. For example, in our application the trees are drawn in the plane and we use the counterclockwise ordering of the edges around each vertex given by the embedding. Edge-ordered dynamic trees support the following repertory of update operations [19].

link(μ', μ'', e', e''). This operation assumes that μ' is the root of a tree T', μ'' is a node of another tree T'', and e'' is an edge incident on μ''. Add a new edge

e' from node μ' to node μ'', thus making T' a subtree of T''. The new edge e' is inserted after edge e'' in the edge ring of μ''.

cut(μ, e). This operation assumes that node μ is not the root of a tree and e is the edge from μ to its parent. Remove edge e, thus separating the subtree rooted at μ.

split$(\mu, \mu', \mu'', e, e_1, e_2)$. Split node μ into two nodes μ' and μ'' connected by a new edge e. If $(\alpha e_1 \beta e_2 \gamma)$ is the edge ring of μ, then $\alpha e \gamma$ and $e e_1 \beta e_2$ are the edge-rings of μ' and μ'', respectively.

merge(μ', μ'', e). Merge adjacent nodes μ' and μ'' connected by edge e into a single node μ. If αe is the edge ring of μ' and βe is the edge ring of μ'', then $\alpha\beta$ is the edge ring of μ.

Let T be a dynamic tree, subject to the above operations. Sleator and Tarjan [42] present two schemes for efficiently performing the link and cut operations on T, and these schemes carry over naturally to edge-ordered dynamic trees [19]. In this paper we assume the scheme that uses *partitioning by size*. In this scheme the edges of T are considered to be directed from the child to the parent, and an edge e from μ to ν is said to be *solid* if the subtree rooted at μ has more than half of the edges of the subtree rooted at ν. Otherwise, edge e is said to be *dashed*. There is at most one solid edge entering any node (from its children). Therefore, every node is in exactly one path of solid edges (of length 0 or more). We refer to these paths as *solid paths*. (See Fig. 1a.) A solid path π is represented as a balanced binary tree P_π, so that T is then stored as a collection of these *path trees*. For more details on partitioning by size and how it can be exploited to efficiently perform dynamic tree updates and queries, see [19, 42].

While link-cut trees support a variety of query operations, such as finding the least-common ancestor of two nodes, we shall use only the following operation that is part of the standard repertory of dynamic trees [19, 42].

expose(μ). Create a solid path π from node μ to the root by converting to solid all the dashed edges of π, and converting to dashed all the solid edges that enter a node of π but are not on π. (See Fig. 1b.) This operation may violate the definition of solid edges, so it is always followed by a procedure that undoes its effects.

Edge-ordered dynamic trees use linear space and support each of the above operations in $O(\log n)$ time, where n is the size of the tree(s) involved in the operation [19, 42].

3. Our approach. In this section we address the problem of performing point location in a triangulation $\mathcal{S}$ with n vertices. Without loss of generality, we assume that $\mathcal{S}$ does not have horizontal edges. General subdivisions can be handled via a preliminary triangulation step, which takes $O(n)$ time if the subdivision is connected [7], and $O(n \log n)$ time otherwise [22].

We describe here a static method that uses $O(n \log n)$ space and preprocessing time and supports point-location queries in $O(\log^2 n)$ time. We will show in the subsequent section how to dynamize this approach so as to achieve $O(n)$ space, the same query time, and an update time that is $O(\log n)$.

3.1. Building the structure. A *monotone spanning tree* T of $\mathcal{S}$ is a rooted spanning tree such that any root-to-leaf path of T is monotone with respect to the y-axis. The root T then is the vertex t in $\mathcal{S}$ with smallest y-coordinate. Such a monotone spanning tree T is obtained by choosing for each vertex v in $\mathcal{S}$ some edge emanating from v, assuming all edges are directed downward. Note that this simple choosing

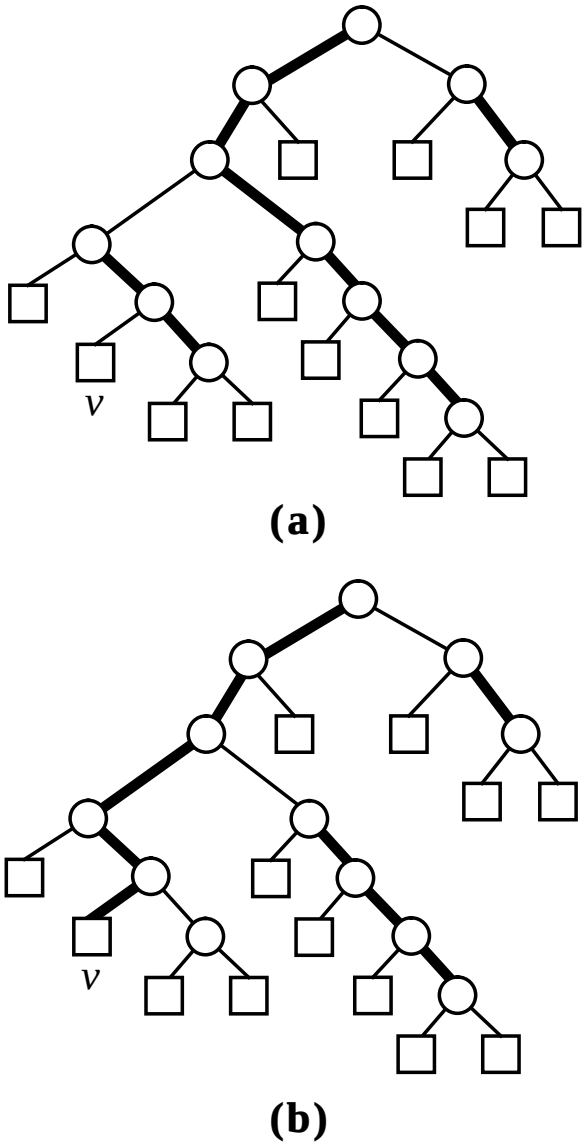


FIG. 1. (a) *A link-cut tree with edges partitioned into solid and dashed.* (b) *Effect of operation* expose(v) *on the tree of part* (a).

operation would not necessarily define a spanning tree if $\mathcal{S}$ were not monotone. (See Fig. 2.) As we show in the next lemma, such a spanning tree has a nice property that can be exploited for point location.

LEMMA 3.1. *For any nontree edge e of $\mathcal{S}$, the fundamental cycle $F(e)$ determined by e and T is a monotone polygon.*

Proof. Let $e = (v', v'')$. Since the spanning tree T is monotone, the paths π' and π'' of T from v' to t and v'' to t are monotone chains. Let v be the least-common ancestor of v' and v''. The cycle $F(e)$ that results when adding e to T is the polygon formed by the following monotone chains: (i) the subpath of π' from v to v' plus edge e and (ii) the subpath of π'' from v to v''. Therefore, we have that $F(e)$ is a monotone polygon. □

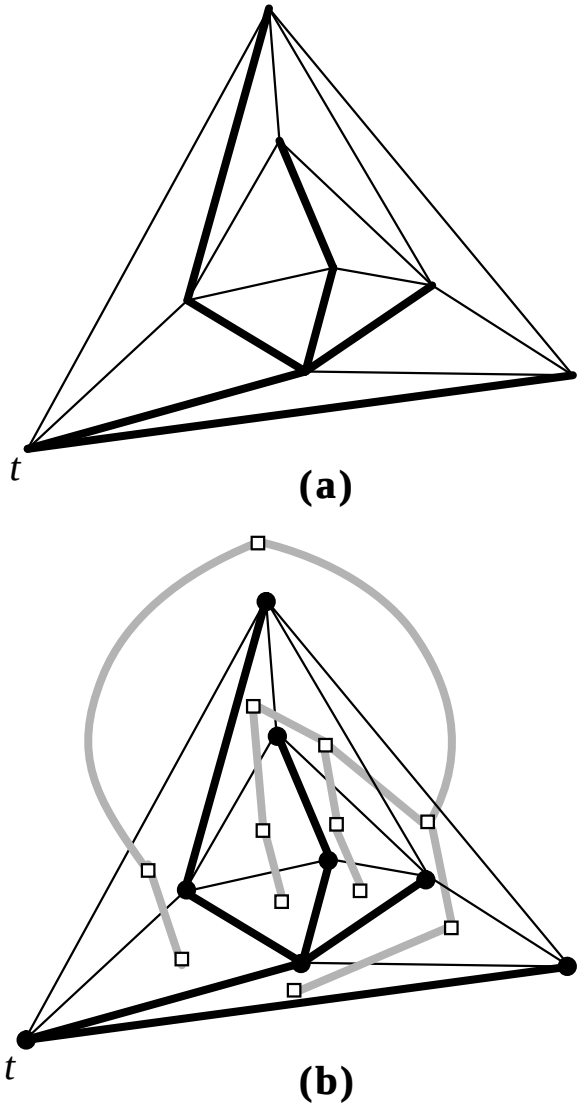


FIG. 2. (a) *A monotone spanning tree.* (b) *Its dual spanning tree.*

This motivates the construction of our point-location structure, which is performed in the following four steps.

Step 1. Construct a monotone spanning tree T for $\mathcal{S}$ and represent T as an edge-ordered tree rooted at the sink vertex t, where the ordering of the edges incident on a vertex is given by the planar embedding.

Step 2. Construct the graph-theoretic planar dual [5] of $\mathcal{S}$, but exclude any edges dual to edges in T. This defines a spanning tree D on the dual graph [19], called the *dual spanning tree* of T (see Fig. 2). Each node of D is a region r of $\mathcal{S}$, and each edge e^* of D corresponds to a nontree edge e in $\mathcal{S}$, which in turn determines a unique cycle $F(e)$ in $\mathcal{S}$ (when e is added to T). We represent D as an edge-ordered tree rooted at the external region, where the ordering of the edges incident on a node (region) is given by the planar embedding.

Step 3. Form a centroid decomposition of D [6, 10, 24]. Recall that a centroid edge in D divides D into two subtrees whose sizes are in the interval $[|D|/3, 2|D|/3]$; hence, a centroid decomposition defines a binary tree B, where each internal node μ in B corresponds to a centroid edge e^*_μ of D, with the left child of μ being the portion of D "below" e_μ (i.e., inside $F(e_\mu)$) and the right child being the portion "above" e_μ (i.e., outside $F(e_\mu)$). (See Fig. 3.)

Step 4. With each node μ in B we store the left and right chains of monotone polygon $F(e_\mu)$ in two sorted arrays $L(\mu)$ and $R(\mu)$, respectively. Note that to avoid confusion in the L and R lists we consider each edge to have two sides, a left side and a right side, which are distinct edges for the sake of this definition. (See Fig. 3.)

LEMMA 3.2. *The above method runs in $O(n \log n)$ time and uses $O(n \log n)$ space.*

Proof. Steps 1 and 2 can be easily implemented in $O(n)$ time. Step 3 takes $O(n)$ time using the method of [10, 24]. Step 4 is the bottleneck step, in that it requires $O(n \log n)$ time and space to copy and store all the L and R lists for the nodes in B (since B has depth $O(\log n)$). □

Having presented our structure, let us describe how it can be used to answer a point-location query.

3.2. Querying the structure. Suppose we are given a query point p, and we wish to locate the cell in $\mathcal{S}$ containing p. Our method for performing this point-location query is actually quite simple. We perform a search down B, where at each node μ we use the lists $L(\mu)$ and $R(\mu)$ to determine if p is inside or outside the polygon $F(e_\mu)$. Since $L(\mu)$ and $R(\mu)$ are stored as arrays, we can perform two binary searches to determine if p is inside $F(e_\mu)$ in $O(\log n)$ time. If p is inside $F(e_\mu)$, then we visit μ's left child next; otherwise, we visit μ's right child next. This procedure continues until we reach the leaf of B corresponding to a single region—the cell of $\mathcal{S}$ containing p. Therefore, we have the following lemma.

LEMMA 3.3. *Our point-location data structure supports point-location queries in $O(\log^2 n)$ time.*

Incidentally, one can improve the query time to $O(\log n)$, while increasing the preprocessing time and space by at most a constant factor, via the *fractional cascading* technique of Chazelle and Guibas [8, 9]. Thus one can modify the above approach to match the query bounds of previous point location methods [18, 26, 41]. Our motivation for designing this new method was not to simply match the performance of previous methods, however, but to design a scheme that leads to an efficient dynamic point-location method. So, let us leave the details of this static data structure to the interested reader and concentrate instead on how this approach can be dynamized.

4. Dynamic planar point location. In this section we show how to implement our point-location method dynamically using dynamic trees. Our dynamic environment supports the following repertory of update operations on a monotone subdivision $\mathcal{S}$ (i.e., we assume that each operation is performed only if it is known to preserve the monotonicity of $\mathcal{S}$).

InsertEdge$(e, r, v, w; r_1, r_2)$. Insert edge e between vertices v and w inside region r, which is then decomposed into regions r_1 and r_2 to the left and right of e, respectively.

DeleteEdge$(e, v, w, r_1, r_2; r)$. Remove edge e between vertices v and w and merge into region r the two regions r_1 and r_2 formerly to the left and right of e, respectively.

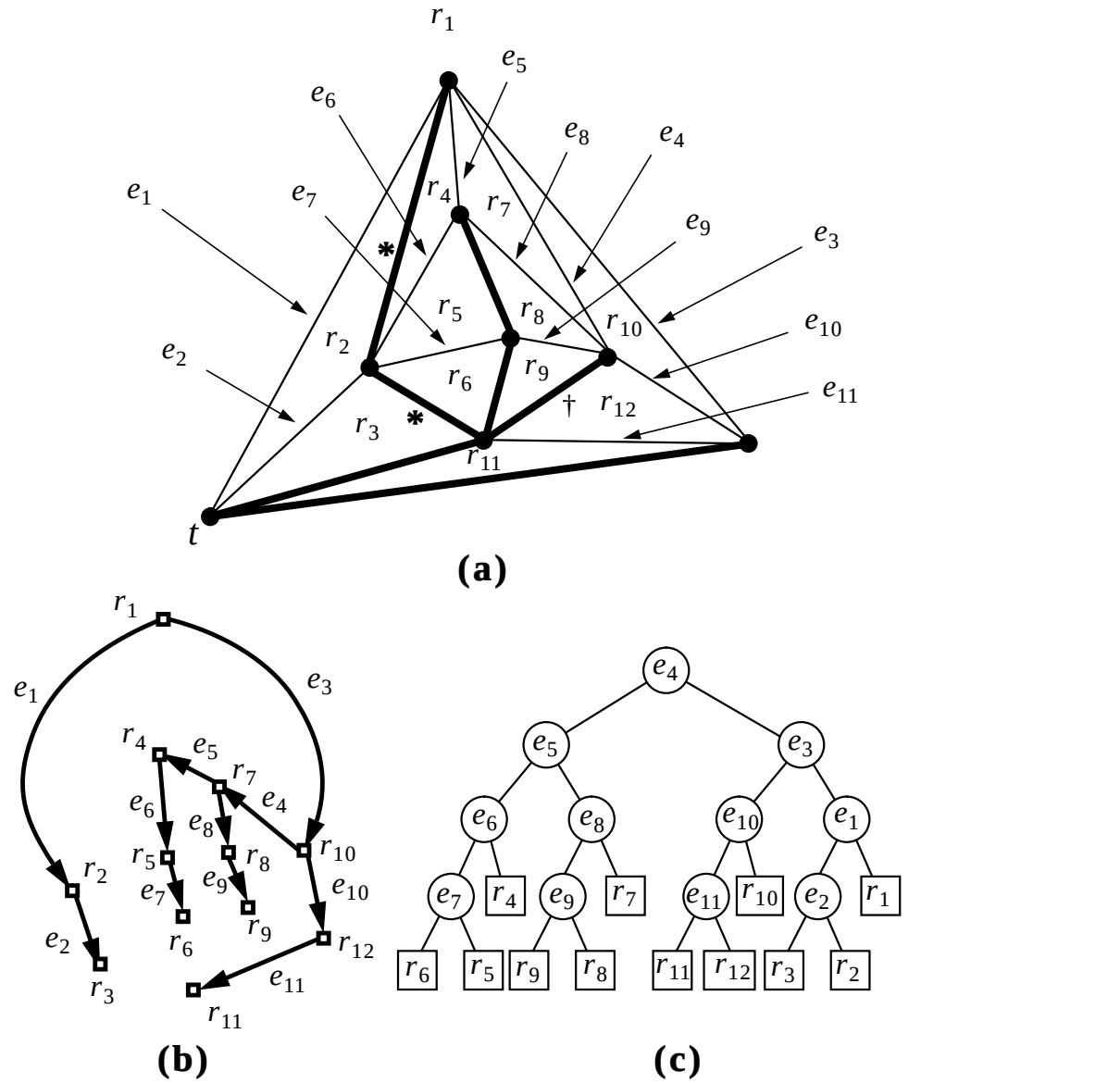


FIG. 3. *A centroid decomposition of the dual spanning tree of the previous figure.* (a) *The subdivision $\mathcal{S}$ and monotone spanning tree T. In this example the edges in $L(\mu)$ are marked with a "$*$" and the edge in $R(\mu)$ (other than e_4) is marked with a "†," where μ is the node in B associated with edge e_4.* (b) *The dual tree D.* (c) *The centroid decomposition tree B.*

Expand$(e, v, r_1, r_2; v_1, v_2)$. Expand vertex v into vertices v_1 and v_2 connected by edge e, which has regions r_1 and r_2 to its left and right, respectively.

Contract$(e, r_1, r_2, v_1, v_2; v)$. Contract edge e between vertices v_1 and v_2 into vertex v. Regions r_1 and r_2 are those formerly to the left and right of e, respectively.

InsertChain$(\gamma, r, v, w; r_1, r_2)$. Insert a monotone chain γ between vertices v and w inside region r, which is then decomposed into regions r_1 and r_2 to the left and right of e, respectively.

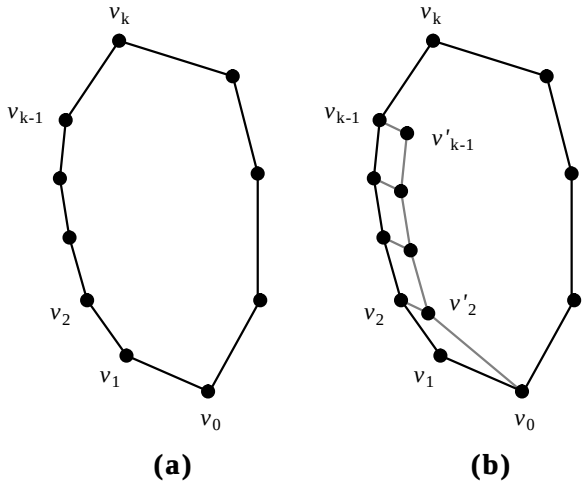


FIG. 4. *Example of refinement of a region.*

DeleteChain$(\gamma, v, w, r_1, r_2; r)$. Let γ be a monotone chain between vertices v and w, whose internal vertices have degree 2. Remove γ and merge into region r the two regions r_1 and r_2 formerly to the left and right of γ, respectively.

Several problems arise in the dynamization of the static structure of section 3. The first is that the static data structure assumes a preliminary triangulation step to ensure that the dual spanning tree has bounded degree and therefore admits a centroid decomposition. But it appears difficult to dynamically maintain a triangulation, since a newly inserted edge could intersect many triangulation edges. So we do not attempt to maintain a triangulation of our current subdivision; instead we refine it so as to maintain a crucial property that such a triangulation would give us.

4.1. A virtual triangulation of the regions in $\mathcal{S}$. Let $\mathcal{S}$ be a monotone subdivision. We refine $\mathcal{S}$ into a new subdivision $\mathcal{R}$ as follows. For each region r of $\mathcal{S}$, let $v_k, v_{k-1}, \ldots, v_0$ be the left chain of r as traversed from top to bottom. If $k \geq 3$, we add inside r a "comb" consisting of $k-2$ new vertices, $v'_2, v'_3, \ldots, v'_{k-1}$, and $2(k-2)$ edges, (v_2, v'_0), (v'_{i+1}, v'_i), $(i = 2, \ldots, k-2)$, and (v_i, v'_i) $(i = 2, \ldots, k-1)$. See an example in Fig. 4. We assume that each new vertex v'_i is placed below and to the right of v_i and infinitesimally close to it. Hence the above refinement affects only the topology of the subdivision, so that a point location query has the same answer in $\mathcal{S}$ and $\mathcal{R}$. Also, it is immediate to verify that the refined subdivision has $O(n)$ vertices.

The *leftist spanning tree* of a monotone subdivision is defined as the monotone spanning tree obtained by selecting the leftmost outgoing edge of every vertex, except the source (see Fig. 5a). In addition to the above refinement of $\mathcal{S}$ into $\mathcal{R}$, we also maintain T as a leftist spanning tree of $\mathcal{R}$, with D being its graph-theoretic planar dual. As we show in the following lemma, this is sufficient to achieve the desired result.

LEMMA 4.1. *The planar dual of the leftist spanning tree in $\mathcal{R}$ has degree at most* 3.

Proof. Let T be the leftist spanning tree, and D its dual (see Fig. 5b). We observe that tree D exactly consists of the dual edges of the topmost edges of the right chain of each region. Also, all the remaining edges of the right chain of each region are in tree T. Hence the degree of a node r of D is at most one plus the number of nontree edges on the left chain of region r.

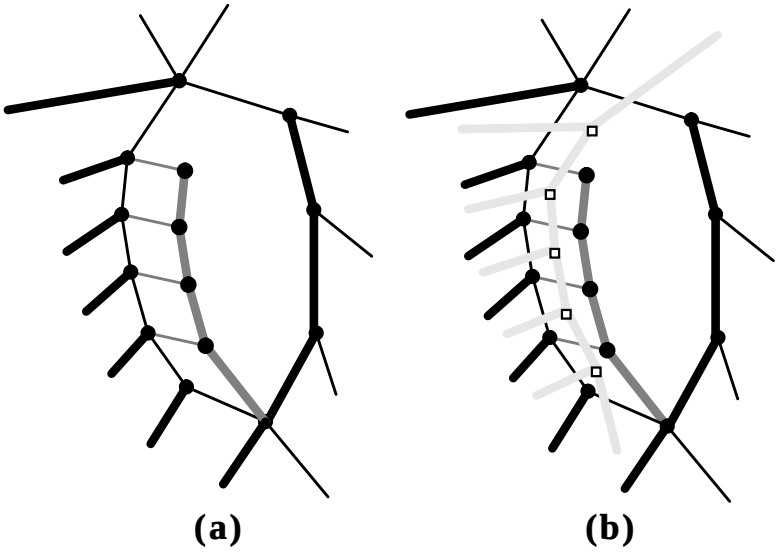


FIG. 5. (a) *Leftist spanning tree of a refined subdivision* $\mathcal{R}$. (b) *Dual of the leftist spanning tree of part* (a). *Tree edges are drawn thick, with the comb edges in this tree being dark gray. The edges of the dual tree are drawn as thick light gray lines, and their dual edges in the comb are drawn as dotted lines. (We use this same convention in all the figures that follow as well.)*

To prove the lemma, we show that every region r of the refined subdivision $\mathcal{R}$ has at most two nontree edges in its left chain. Namely, if region r is to the left of a comb, then it has exactly two edges and thus no more than two nontree edges in its left chain. Else (r is to the right of a comb), only the two topmost edges of the left chain of r may not be in T, since each of the remaining edges (which form the "spine" of the comb) is the only outgoing edge of its end vertex and hence is in T. □

Our dynamic data structure for point location in $\mathcal{S}$ simply consists of the leftist spanning tree T of $\mathcal{R}$, and of its dual spanning tree D, each represented as an edge-ordered dynamic tree [19, 42]. Tree T is rooted at the node associated with the (bottom-most) sink vertex t, and tree D is rooted at the node associated with the external region. In both trees, the ordering of the edges incident on each node is given by the planar embedding. The overall space requirement of the data structure is $O(n)$.

4.2. Finding fundamental cycles. In order to perform queries efficiently we must be able to construct searchable representations of fundamental cycles in T, the monotone spanning tree for $\mathcal{R}$. More significantly, like our triangulation, our representations must be virtual, since an update operation may cause substantial restructurings in the centroid tree and edge lists. Our approach for overcoming this difficulty consists of representing T as an edge-ordered dynamic tree [19] (see also [42]). As we show in the following lemma, this is sufficient for us to be able to quickly perform a point-cycle query in T.

LEMMA 4.2. *Let T be a monotone spanning tree of $\mathcal{S}$, with root t. By representing T as an edge-ordered link-cut tree, one can determine in time $O(\log n)$ whether a query point p is on, inside, or outside the fundamental cycle $F(e)$ induced by a nontree edge e of $\mathcal{S}$.*

Proof. In a link-cut tree [19, 42] representing T the operation *expose*(v) returns a balanced binary tree P_π that represents the path π of T between the root t and vertex v (i.e., the external and internal nodes of P_π store the vertices and edges of π, respectively, such that the in-order visit of P_π yields π). Hence we can determine if a query point p is inside $F(e)$ as follows. Let v' and v'' denote the left and right endpoints of edge e, respectively. We issue an *expose*(v') and perform a binary search on the balanced-tree representation of the left chain of $F(e)$, minus edge e, that is returned, to determine if p is to the left, to the right, or outside the scope of y-coordinates for this chain. After the structural changes in the link-cut representation of T from this expose are undone, we then issue an *expose*(v'') and perform a similar binary search on the balanced-tree representation of the right chain of $F(e)$, minus edge e, that is returned. Whether a point p is on, inside, or outside cycle $F(e)$ can then be easily determined from the results of these two searches and a simple comparison involving the edge e. All of the above steps take $O(\log n)$ time. □

Constructing fundamental cycles in $\mathcal{S}$ is important, but not sufficient, for, in order to achieve an $O(\log^2 n)$ query time, we must also be able to find a centroid edge in the dual tree, D.

4.3. Locating a centroid edge in the dual tree. We do not explicitly maintain a centroid decomposition tree for D, however. Instead, we show in the following lemma that the link-cut representation of D can itself be used to quickly find a centroid edge in D.

LEMMA 4.3. *Let D be a tree of degree 3 represented by a link-cut tree with partitioning by size. Then a centroid edge in D can be located in $O(\log n)$ time.*

Proof. As mentioned above, one of the main ideas of the link-cut tree data structure is to partition the tree D into "solid" paths and "dashed" edges [19, 42] and represent each solid path with a binary search tree. Let π be the solid path containing the root of D. We claim that the set of edges that are either in π or incident on the first node of π contains a centroid edge.

Proof of claim. Let $\pi = (\mu_1, \ldots, \mu_k)$. We denote with S_i the subset of nodes of D consisting of node μ_i and the nodes in the (at most three) subtrees connected to μ_i by dashed edges (see Fig. 6a). Let w_i be the size of S_i, called the *weight* of node μ_i. We have that $\sum_{i=1}^{k} w_i = n$, where n is the number of nodes of D. From the definition of dashed edges [19, 42], we have that $w_i < n/2$ for $i = 2, \ldots, k$. We distinguish two cases. If $w_1 \leq 1 + 2n/3$, then there exists some j such that $n/3 - 1 \leq \sum_{i=1}^{j} w_i \leq 1 + 2n/3$. In this case the solid edge from μ_j to μ_{j+1} is a centroid edge. Otherwise, the dashed edge connecting the largest subtree of μ_1 is a centroid edge, since the largest of the subtrees of μ_1 has at most $n/2$ nodes (because it is connected by a dashed edge) and at least $n/3 - 1$ nodes (because μ_1 has no more than three incident edges). The claim is proved.

Therefore, we can find a centroid edge of D in $O(\log n)$ time by traversing a root-to leaf path in the binary tree of the solid path containing the root of D. (See Fig. 6b.) □

Thus we have shown how to perform the two main components of our point-location procedure.

4.4. Point-location querying. The location of a query point p, therefore, is performed as follows.

1. If D consists of a single node, we return the corresponding region and stop.
2. We find a centroid edge e^* of D using the algorithm of Lemma 4.3.

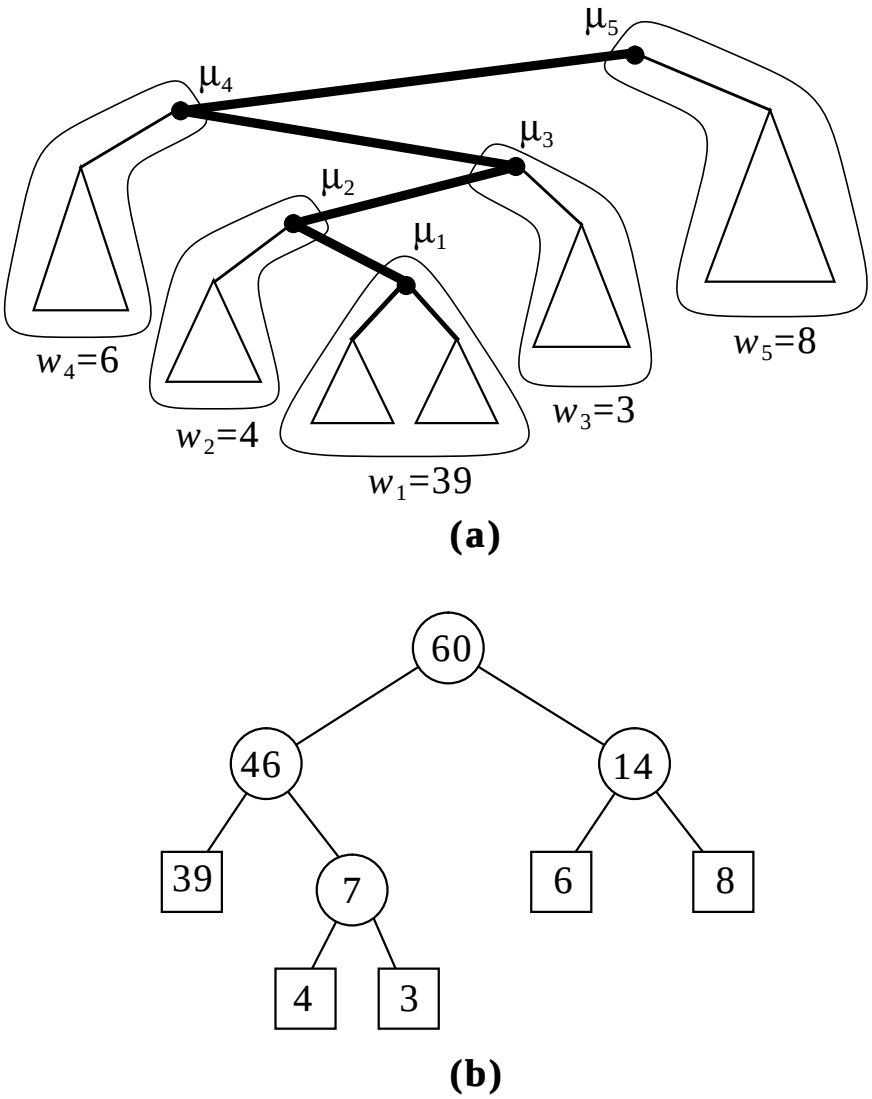


FIG. 6. (a) *Solid path containing the root of a dynamic tree.* (b) *Balanced tree associated with the solid path of part* (a).

3. We cut tree D at edge e^* and let D' and D'' be the resulting trees, where D'' contains the former root of D. Note that since this is a query step, not an update, we also store the edge e^* on a stack in this step, so that after the query is done we can reconstruct the original D via a series of link operations.
4. We determine if p is on, inside, or outside cycle $F(e)$ by applying the algorithm of Lemma 4.2 to tree T.
5. If p is on cycle $F(e)$, return the edge or vertex of $F(e)$ that contains e. Else, if p is inside the cycle, recursively apply the algorithm using D'. Otherwise (p is outside the cycle), recursively apply the algorithm using D''.

Our query operation is completed by reconstructing tree D by means of a sequence of $O(\log n)$ link operations that undo the cuts (by a series of pop operations on the stack used in step 3).

Thus we have the following theorem.

THEOREM 4.4. *The above dynamic point-location data structure supports point-location queries in time $O(\log^2 n)$ and uses $O(n)$ space.*

Proof. The query time bound should be immediately apparent given the above description. For the space bound note that the data structure is no more than $\mathcal{R}$, represented using any standard plane graph representation and T and D represented as link-cut trees (plus cross pointers, so, for example, each nontree edge in $\mathcal{R}$ has a pointer to its dual in D). □

Having given our method for performing queries, let us next address our methods for updating $\mathcal{R}$. We begin with the *Contract*and *Expand*operations.

4.5. Edge contraction and expansion. Recall that in the *Expand*(v, v_1, v_2, e) operation we expand vertex v into vertices v_1 and v_2 connected by edge e with regions r_1 and r_2 being to the left and right of e, respectively (see Fig. 7). There are two cases. In the first case the relative positions of v_1 and v_2 require that e become an edge in the leftist spanning tree T (as illustrated in Fig. 7a). In this case we perform the obvious *split* in T at v and update the corresponding pointer structures in $\mathcal{R}$ and D. We may also have to add an edge to the combs of pseudo edges in r_1 and r_2 so as to maintain our refinement invariant. If this occurs we also need to update the dual tree D (using $O(1)$ *split* and *link* operations) so that it remains a planar dual to $\mathcal{R}$. This can all be done in $O(\log n)$ time.

In the second case the positions of v_1 and v_2 require that e become a nontree edge (see Fig. 7b). In this case we perform the obvious *split* in T at v, forming v_1 and v_1, *cut* T along e, and then link v_2 to its leftmost adjacent node, w in $\mathcal{R}$. We also perform any edge additions to combs in r_1 and r_2, if necessary, as in the first case. Of course, our modifications of T require that we perform changes to D. Specifically, we must cut D at the edge dual to (v_2, w) and perform a link to create a node dual to e. Since this can all be done in $O(\log n)$ time, it implies that we can perform the *Contract*operation in $O(\log n)$ time.

We implement the *Expand*operation by "reversing" the above steps in the obvious manner. Thus we may also perform the *Expand*operation in $O(\log n)$ time.

4.6. Edge insertions and deletions. The next update operation we consider is edge insertion. Recall that in the operation *InsertEdge*$(e, r, v, w; r_1, r_2)$ we insert edge e between vertices v and w inside region r, which is then decomposed into regions r_1 and r_2 to the left and right of e, respectively. There are two cases.

In the first case v and w are on opposite sides of r; i.e., without loss of generality, v is on the left chain of r and w is on the right chain of r (see Fig. 8). We distinguish two subcases.

1.1. Suppose $e = (v, w)$ must become an edge in the leftist spanning tree T (see Fig. 8a). In this case we perform a *cut* in T along the edge going out of w and then link the resulting subtree rooted at w to v. This may also require that we cut the comb in r at the vertex associated with v, deleting its incident edges, and begin a new comb at v (which contains any previous comb edges above v). Likewise, each *cut* in T has a corresponding *link* in D, and each *link* in T has a corresponding *cut* in D. Since the number of needed *cut* and *link* operations is $O(1)$, the total time for this case is $O(\log n)$.

1.2. Suppose $e = (v, w)$ must become a nontree edge (see Fig. 8b). In this case we need only change the comb in r by cutting it at the vertex associated with v and beginning a new comb at w. This can be done with $O(1)$ *cut* and *link* operations in T, but it does not change the topology of D. Thus this case

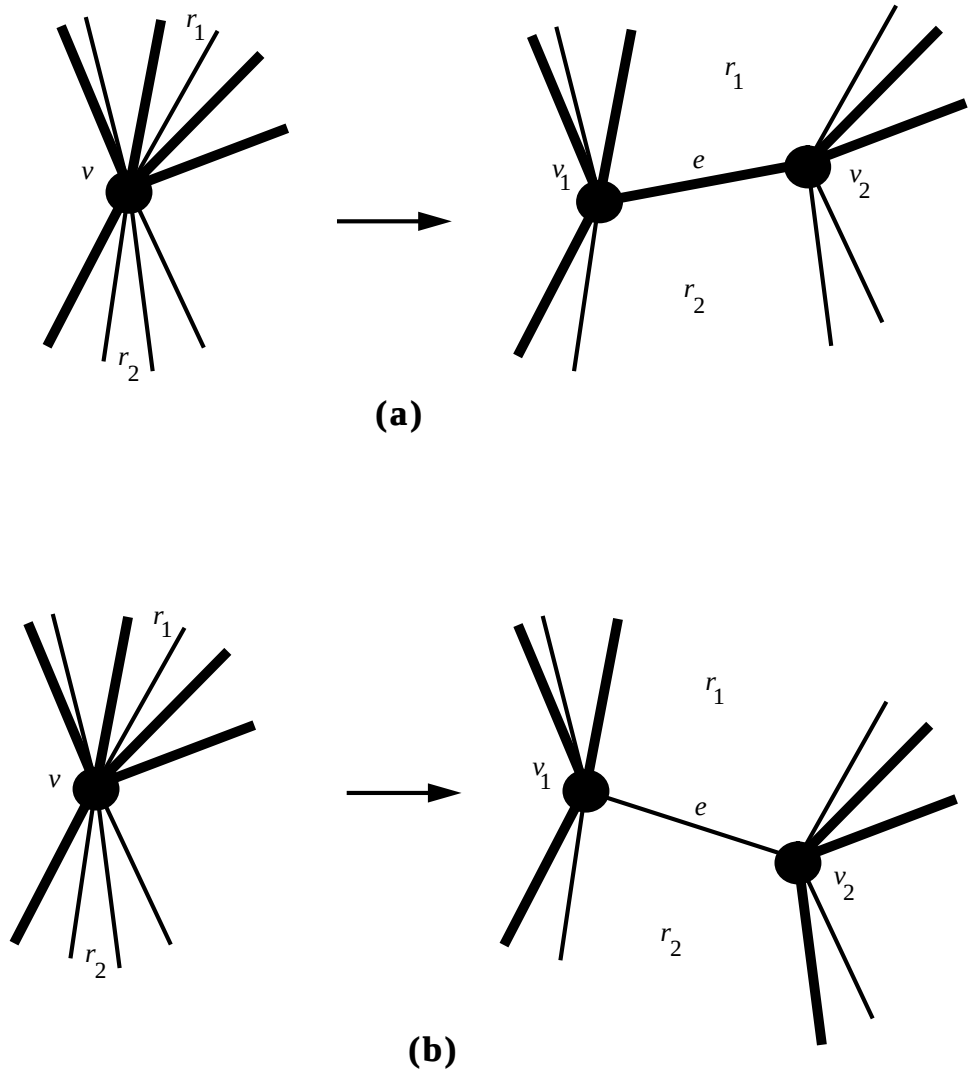


FIG. 7. (a) *An* Expand*operation in which* e *becomes an edge in the leftist spanning tree* T*, and* (b) *an* Expand*in which* e *becomes a nontree edge.*

also takes only $O(\log n)$ time.

In the second case for edge insertion vertices v and w are on the same side of r. There are two obvious subcases.

2.1. Suppose v and w are both in the left chain of r (see Fig. 9a), where, without loss of generality, v has larger y-coordinate than w. In this case we must cut the comb in r at the vertex associated with v and the vertex associated with w, and add the edge $e = (v, w)$ as a nontree edge in $\mathcal{R}$. We begin a new comb at w in r_1 that retains the edges of the old comb between the vertices associated with v and w, and we concatenate the portion of the old comb below the vertex associated with w with the portion of the old comb above the vertex associated with v, to form the new comb for r_2. This can all be done using $O(1)$ *link* and *cut* operations in T. Likewise, cutting the

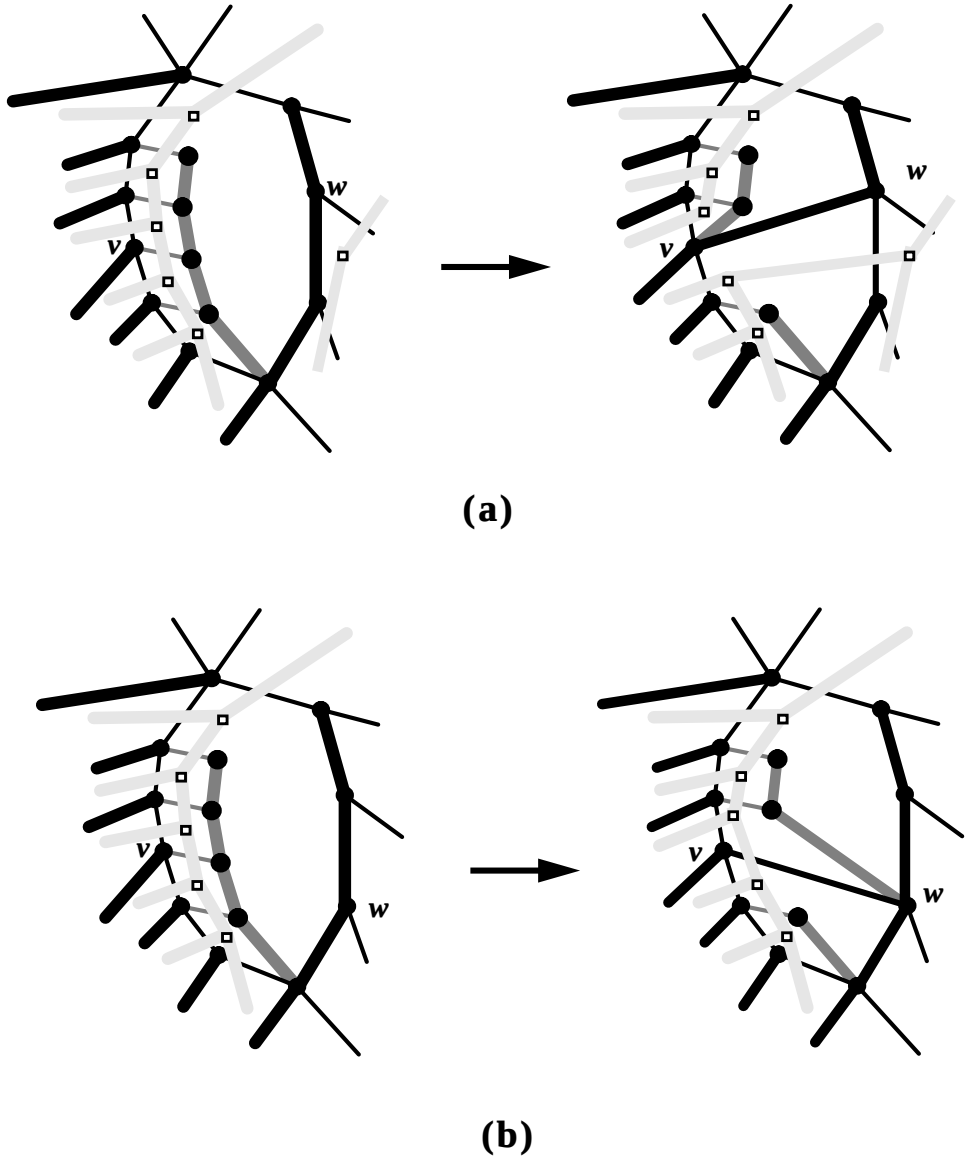


FIG. 8. *Insertions where v and w are on opposite sides of the region r. We illustrate in* (a) *an* InsertEdge*operation in which e becomes an edge in the leftist spanning tree T and in* (b) *an* InsertEdge*in which e becomes a nontree edge.*

comb in r also requires that we perform associated cuts in D, and the comb concatenations have associated links in D. Again, there are only $O(1)$ such operations, however, so this case can be implemented in $O(\log n)$ time.

2.2. Suppose v and w are both in the right chain of r (see Fig. 9b), where, without loss of generality, v has larger y-coordinate than w. In this case we simply cut T at the edge f going out of v and perform a link of the subtree rooted at v along the edge $e = (v, w)$. This also requires that we create a new (leaf) node in D and link it along the edge dual to f. We need not change the comb

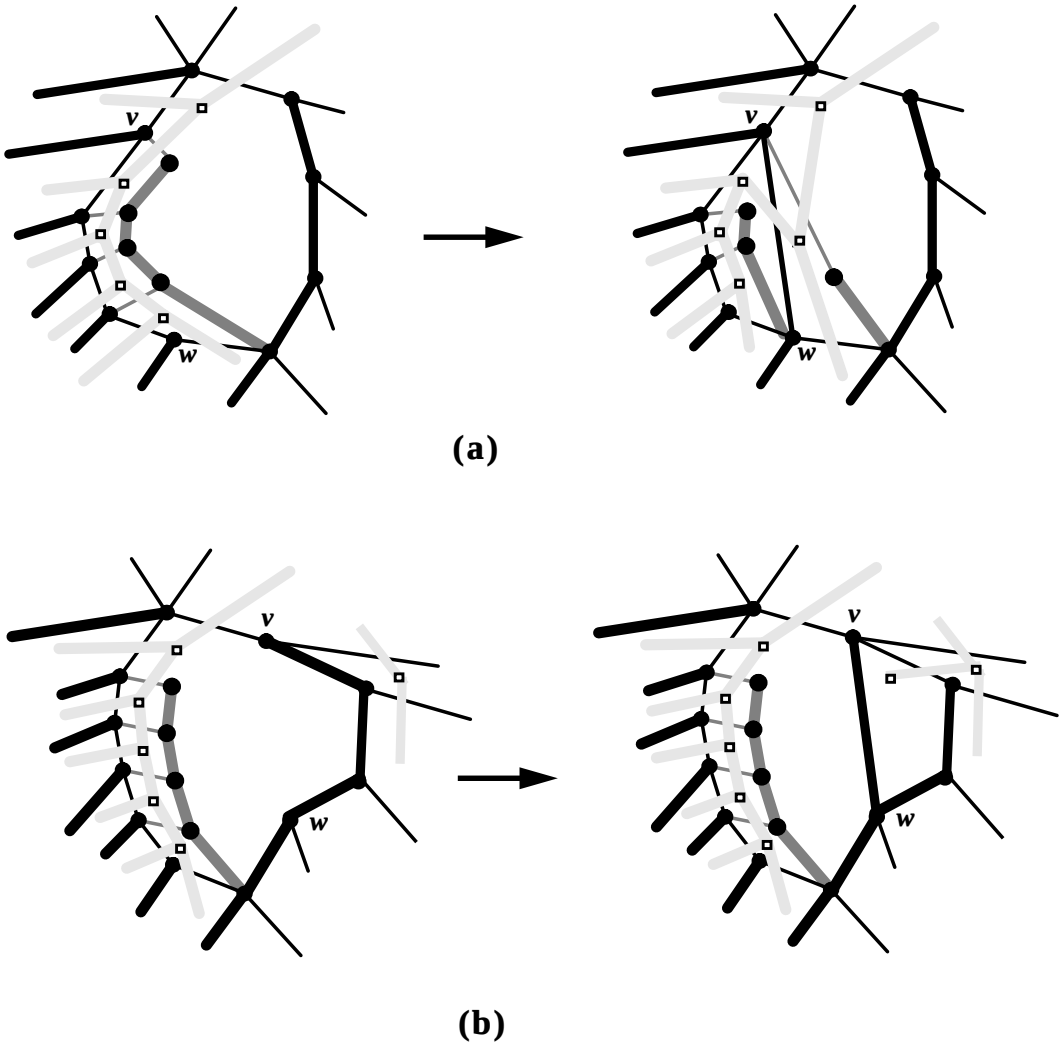


FIG. 9. *Insertions where v and w are on the same side of the region r. We illustrate in* (a) *the case when v and w are in the left chain and in* (b) *the case when v and w are in the right chain.*

in r (which is now the comb for r_1), and the comb in r_2 is the null comb, so this completes the construction. Clearly, this case requires $O(\log n)$ time.

Thus we can perform the *InsertEdge*operation in $O(\log n)$ time. Since the *DeleteEdge*operation is the "reverse" of an *InsertEdge*, this also implies that we can perform the *DeleteEdge*operation in $O(\log n)$ time.

4.7. Chain updates. The only update operations that remain to be described are the chain update operations. Recall that in the operation *InsertChain*$(\gamma, r, v, w; r_1, r_2)$ we insert a monotone chain γ between vertices v and w inside region r, which is then decomposed into regions r_1 and r_2 to the left and right of e, respectively. Note that this is essentially the same as in the case of the *InsertEdge*operation, except that instead of adding a single edge (v, w) we are now inserting a monotone chain. It should not be surprising, then, that our method for performing the *InsertChain*operation

is the same as that for the *InsertEdge*operation, except that where we previously performed a single link in T from v to w, and added a trivial chain in r_2 from v to w, we must now link in an entire chain in T, as well as its corresponding comb. If we perform these link operations in series, then we will require $O(k \log n)$ time, where k is the length of the chain. So, instead, we first build a link-cut tree representation of the chain and its corresponding comb, which takes $O(k)$ time [19, 42], and then we perform the $O(1)$ link operations required to link these chains into T. Likewise, we must build a chain of size $O(k)$ dual to the inserted chain and its comb and link this into D, but again a link-cut tree representation of the chain can be built in $O(k)$ time, and then this can be linked into D with $O(1)$ link operations. Thus the entire time needed for the *InsertChain*operation is $O(\log n + k)$. Since the *DeleteChain*operation amounts to the reversal of this procedure, this also implies that the *DeleteChain*operation can be implemented in $O(\log n + k)$ time (it is actually easier, since we replace the building of link-cut tree representations of $O(k)$-length chains with the garbage collection of the space used by such representations). Therefore, we have the following theorem.

THEOREM 4.5. *Let $\mathcal{S}$ be a monotone subdivision of current size n that is subject to a sequence of on-line updates. Point location in $\mathcal{S}$ can be done with a fully dynamic data structure that uses $O(n)$ space and supports queries in time $O(\log^2 n)$ and update operations* InsertEdge*,* DeleteEdge*,* Expand*, and* Contract*in time $O(\log n)$. Also, update operations* InsertChain*and* DeleteChain*take time $O(\log n + k)$, where k is the size of the monotone chain being inserted or deleted. All the time bounds are worst case.*

5. Spatial point location. We can extend our method further to derive an efficient algorithm for performing point location in 3-dimensional cell complexes whose cells are convex polytopes. Let $\mathcal{C}$ be such a convex cell complex with n vertices and N facets. Note that both n and the number of edges of $\mathcal{C}$ are $O(N)$. Following the same general approach of Preparata and Tamassia [40], we obtain a spatial point-location data structure by combining the persistence-addition technique of Driscoll et al. [16] and our dynamic structure for planar point location.

A conventional dynamic data structure is called *ephemeral* since its instantiation preceding an update is not recoverable after the execution of the update. A *fully persistent* structure supports both accesses and updates to any of its past versions; a *partially persistent* structure supports accesses to any of its past versions but updates only to its most current version. The general technique of Driscoll et al. [16] can be used to add persistence to an ephemeral linked data structure whose records are pointed to by a bounded number of pointers. The resulting persistent data structure uses additional $O(1)$ amortized space per update operation and has the same asymptotic query time (worst case for partial persistence and amortized for full persistence). Since each of our update operations requires a total time of $O(\log n)$ in the worst case, this implies that we make at most $O(\log n)$ pointer updates in any update. Therefore, each of our update operations can be implemented persistently in $O(\log n)$ amortized time, and each one adds $O(\log n)$ amortized additional space to be added to the persistent data structure. In order to implement this persistent strategy, however, we need the following lemma.

LEMMA 5.1. *The dynamic point-location data structure of Theorem* 4.5 *can be implemented with a linked representation such that each record is pointed to by a bounded number of pointers.*

Proof. As mentioned above, our data structure is essentially just a link-cut tree representation of a leftist monotone spanning tree T and its graph-theoretic planar dual D. Moreover, we maintain D as a degree-3 tree, which implies that the underlying link-cut tree representation satisfies the bounded number of pointers condition (see [19, 42] for details). The tree T need not have bounded degree, however. Nevertheless, by using the implementation of edge-ordered dynamic trees given by Eppstein et al. [19], we represent T so as to satisfy the bounded-degree condition (see [19] for details). □

Thus we can create a persistent version of our point location data structure. But being able to search in the "past" must also be meaningful. We find this meaning in the following lemma.

LEMMA 5.2. *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be monotone subdivisions whose associated planar st-graphs are isomorphic. A dynamic point location data structure for $\mathcal{S}_1$ (as discussed in Theorem* 4.5*) can be used for dynamic point location in $\mathcal{S}_2$ after changing only the values of the vertex coordinates.*

Proof. Since the planar *st*-graphs associated with $\mathcal{S}_1$ and $\mathcal{S}_2$ are isomorphic, the leftist spanning tree for (the refinement of) $\mathcal{S}_1$ and the leftist spanning tree for (the refinement of) $\mathcal{S}_2$ are isomorphic (as are their respective graph-theoretic planar duals). Thus applying our construction to $\mathcal{S}_1$ yields a data structure that is topologically identical to that for $\mathcal{S}_2$. Moreover, at no place in our point-location method do we ever explicitly need the coordinates of the subdivision endpoints. We only needed to be able to perform a comparison-based binary search for a y-coordinate in a monotone chain, and then we must be able to determine to which side of a line L a query point lies. That is, we can use the actual x- and y-coordinates of a query point *implicitly* to resolve y-coordinate comparisons in a binary search of a monotone chain or the "side-of" comparisons with an oriented line L (i.e., we "plug" them in at the last moment). Thus by replacing the comparison tests of $\mathcal{S}_1$ with the isomorphic tests in $\mathcal{S}_2$, we obtain a dynamic point-location structure for $\mathcal{S}_2$. □

We reduce the 3-dimensional point-location problem to an application of persistence to a dynamic 2-dimensional point location where we "sweep" space by a plane $\pi(z)$ parallel to the x- and y-axes and at height z for $z = -\infty$ to $z = +\infty$. Let $\mathcal{C}(z)$ be the intersection of $\mathcal{C}$ with the plane $\pi(z)$. It is easy to verify that $\mathcal{C}(z)$ is a convex (and hence monotone) subdivision. We view the z-axis as a measure of "time" and consider the process of making plane $\pi(z)$ sweep the cell complex $\mathcal{C}$. The location of a query point $q = (x, y, z)$ in the cell complex $\mathcal{C}$ can be reduced to the location of point (x, y) in the monotone subdivision $\mathcal{C}(z)$. Hence spatial point location can be performed using a partially persistent planar point location data structure.

While the geometry of $\mathcal{C}(z)$ continuously evolves in time, its topology changes only when plane $\pi(z)$ goes through a vertex v of $\mathcal{C}$; i.e., for z', z'' such that $z_i < z' < z'' < z_{i+1}$ the planar *st*-graphs associated with $\mathcal{C}(z')$ and $\mathcal{C}(z'')$ are isomorphic. Hence the space-sweep process goes through $2n+1$ topologically different subdivisions. Also, when the plane $\pi(z)$ goes through a vertex v_i, the resulting modification of the subdivision $\mathcal{C}(z)$ can be performed by a sequence of f_i update operations, each an *Expand*or a *Contract*, where f_i is the number of facets whose top or bottom vertex is v_i (see Preparata and Tamassia [40] for more details). Note that $\sum_{i=1}^{n} f_i = O(N)$.

By Lemma 5.2, the same planar point-location data structure can be used for all query points whose z-coordinate is in the range (z_i, z_{i+1}), provided the x and y coordinates of the vertices are expressed as (linear) functions of z. Thus our data structure for spatial point location consists of a partially persistent version of the dynamic

planar point-location data structure of Theorem 4.5. By Lemma 5.1, such structure satisfies the hypothesis for applying the persistence-addition technique of [16].

It is important to observe that although our query algorithm modifies the ephemeral data structure (see section 4.2), such changes are only temporary and need not be remembered by the persistent data structure. Hence at the expense of increasing the storage space by a constant factor we can use duplicate copies for the pointers and data fields that are modified by a query operation. The duplicate fields are disregarded during the updates.

We summarize the performance of our spatial point-location data structure in the following theorem.

THEOREM 5.3. *Let $\mathcal{C}$ be a convex 3-dimensional cell complex with N facets. There exists a data structure for point location in $\mathcal{C}$ that uses $O(N \log N)$ space and supports queries in $O(\log^2 N)$ time worst case.*

6. On-line point location. Many applications involve constructing an object incrementally while requiring that all the properties of the structure be maintained *on line*. In the context of this paper, we desire a scheme to incrementally construct a planar subdivision while maintaining an efficient point-location data structure for it. This can also be viewed as an instance of dynamic point location when only insertions are allowed. In this section we show how to implement our centroid-decomposition approach to planar point location on line using $BB(\alpha)$ trees and some dynamic data structuring techniques of Overmars [34] to achieve $O(1)$ amortized time per update and $O(\log n \log\log n)$ time (worst case) for answering queries.

We support the following operations.

InsertVertex$(v, e, r_1, r_2; e_1, e_2)$. Insert a vertex v on edge e, which has regions r_1 and r_2 to its left and right, respectively, expanding e into e_1 and e_2.

InsertEdge$(e, r, v_1, v_2; r_1, r_2)$. Insert edge e between vertices v_1 and v_2 inside region r, which is decomposed into regions r_1 and r_2 to the left and right of e, respectively. v_1 and v_2.

InsertChain$(\gamma, r, v_1, v_2; r_1, r_2)$. Insert a monotone chain γ between vertices v_1 and v_2 inside region r, which is decomposed into regions r_1 and r_2 to the left and right of e, respectively.

6.1. A simple on-line point location structure. We explicitly maintain the monotone tree T and the dual tree D for the refined subdivision $\mathcal{R}$. We also explicitly maintain B, the balanced decomposition tree of D, in a $BB(\alpha)$ tree. Each node μ in B corresponds to a subtree D_μ of D, which in turn corresponds to a subpolygon P_μ of P. Thus each leaf in B corresponds to a node of D, which in turn corresponds to a region in $\mathcal{R}$. For each node μ in B we explicitly store the chains $L(\mu)$ and $R(\mu)$. In addition, in the spirit of *fractional cascading*[1] [8, 9], for each node μ we maintain auxiliary lists $AL(\mu)$ and $AR(\mu)$, which are defined recursively as follows:

$$AL(\mu) = \begin{cases} L(\mu) & \mu \text{ is a leaf,} \\ L(\mu) \cup AL(\lambda) \cup AL(\nu) & \text{otherwise,} \end{cases}$$

$$AR(\mu) = \begin{cases} R(\mu) & \mu \text{ is a leaf,} \\ R(\mu) \cup AR(\lambda) \cup AR(\nu) & \text{otherwise,} \end{cases}$$

where λ and ν are the children of μ should μ be an internal node. By keeping pointers from each element x in $AL(\mu)$ to its predecessors in $L(\mu)$, $AL(\lambda)$, and $AL(\nu)$ (and

[1] Specifically, we create auxiliary search lists as in the fractional cascading paradigm; we do not, however, need to implement fractional list propagation.

similar pointers for each x in $AR(\mu)$) we can answer queries in $O(\log n)$ time. This is because after an $O(\log n)$ time binary search in the AL and AR lists for the root of B, the search at every other node in B requires only $O(1)$ time, and there are $O(\log n)$ such other nodes. The update operations are easily implemented as follows.

InsertVertex$(v, e, r_1, r_2; e_1, e_2)$. We first make the obvious update to the planar graph representation of $\mathcal{R}$, inserting v on e and splitting e into e_1 and e_2. We then use the pointer to e to locate the records in $L(\mu)$ and $R(\mu')$ for the endpoint w of e that is nearer to the root of T. We then add a record for v next to w's record in $L(\mu)$ and $R(\mu')$, respectively. This can all easily be done in $O(1)$ time. We must also update the auxiliary lists, however, by adding a record for v to the AL list for each node from μ to the root of B and a record for v to the AR list for each node from μ' to the root. This can be implemented in $O(\log n)$ time by performing a query for v from the root to μ and μ', respectively, and adding v to each list we search in along the way.

InsertEdge$(e, r, v_1, v_2; r_1, r_2)$. We first make the obvious update to the planar graph representation of $\mathcal{R}$, inserting e into r and splitting r into r_1 and r_2. This also necessitates that we modify the node λ in D corresponding to r. This modification will be in the form of the division of λ into two nodes λ_1 and λ_2, with some of λ's adjacencies becoming λ_1's adjacencies and the other adjacencies of λ becoming λ_2's adjacencies. Of course, in performing this modification of D we must also update B to reflect this modification. We do this by visiting the leaf μ in B associated with λ, creating two new nodes μ_1 and μ_2, which are associated with λ_1 and λ_2, respectively, and making these nodes be the children of μ. This addition, in turn, requires that we update the balance information stored in B, and in some cases this requires that we perform node rotations in B to maintain the weight-balance requirements of this $BB(\alpha)$ tree. Performing a rotation at a node μ in B requires more than just updating balance information and changing some pointers at the nodes around μ—it also requires that we change the L and R lists (and their associated auxiliary lists) for μ and ν, the child of μ that is now becoming the parent of ν. Note, however, that we must change the pointer fields of the records in the auxiliary lists at μ's old parent ρ, but we do not need to add or delete any records from these lists. This is because the set of descendants of ρ do not change. Thus the time required to perform such a rotation is proportional to the size of the L, R, AL, and AR lists at μ and ν, which is proportional to n_μ, the number of descendants of μ.

InsertChain(γ, v_1, v_2). This operation can be implemented by combining the methods for *InsertVertex* and *InsertEdge*. We leave the details to the interested reader.

From the above descriptions it should be clear that queries can be answered in $O(\log n)$ worst-case time, as can *InsertVertex* operations. Also, the worst-case time for an *InsertEdge* operation is $O(n)$. Nevertheless, since we are represented B as a $BB(\alpha)$ tree, we can derive an efficient amortized running time for the *InsertEdge* operation. In particular, we observe that performing a rotation at μ requires $O(n_\mu)$ time, where n_μ is the number of leaf descendants of μ. This is because the size of $AL(\mu)$ and $AR(\mu)$ is bounded by the number of vertices in $P(\mu)$, the subpolygon associated with μ, which is $O(n_\mu)$. We can, therefore, take advantage of the following lemma.

LEMMA 6.1 (see [32]). *Let $\alpha \in (1/4, 1 - \sqrt{2}/2)$ and let f be a nondecreasing function such that the cost of performing a rotation in a $BB(\alpha)$ tree at a node μ is $f(n_\mu)$. Then the total cost of the rebalancing operations in a sequence of m insertions*

and deletions into an initially empty tree is

$$O\left(m \sum_{i=1}^{c \log m} (1-\alpha)^i f((1-\alpha)^{-i-1})\right),$$

where $c = 1/\log(1-\alpha)$.

This immediately implies that the cost of performing a sequence of n *InsertEdge* operations starting with an initially empty subdivision is $O(n \log n)$; hence the amortized cost of each *InsertEdge* operation is $O(\log n)$. This gives us the following theorem.

THEOREM 6.2. *One can maintain a monotone subdivision on-line with* $O(\log n)$ *query time for point locations and* $O(\log n)$ *amortized time for vertex and edge insertions (inserting a chain of* k *vertices requires* $O(k \log n)$ *amortized time). The space for this data structure is* $O(n \log n)$.

One can improve the space of the above method to $O(n)$ at the expense of making the query time an amortized bound, using the methods of Fries [20], and Fries, Melhorn, and Näher [21]. As we mentioned earlier, however, our interest is in performing updates in $O(1)$ amortized time ($O(k)$ time for chain insertion). In the next section we show how to modify our approach to achieve this goal. Our modification reduces the space to $O(n)$ and increases the query time by only a $\log \log n$ factor.

6.2. Improving the implementation. The main idea of our improvement is to apply a "bucketing" technique [34] at two different places in our structure. The first application is for the L and R lists at the nodes of B. For simplicity of expression, let us concentrate our attention on the L lists; the modifications for the R lists are similar. For each node μ we add a list $L'(\mu)$, which we maintain to be a subsequence of $L(\mu)$ so that between any two consecutive elements (e, f) in $L'(\mu)$ there are at most $2N$ elements of $L(\mu)$, where N is $\Theta(\log n)$. Moreover, for each pair of consecutive elements (e, f) in $L'(\mu)$ we store the elements of $L(\mu)$ that fall between e and f in a data structure that allows $O(1)$ insertion time, given an element's position, and $O(\log n_e)$ query time, where $n_e = O(N)$ is the number of elements between e and f [29]. The elements of $L(\mu)$ between e and f can intuitively be viewed as belonging to a "bucket" for the pair (e, f).

We modify our definition of the AL and AR lists to take advantage of the sublists L' and R'. In particular we now define $AL(\mu)$ and $AR(\mu)$ as follows:

$$AL(\mu) = \begin{cases} L'(\mu) & \mu \text{ is a leaf,} \\ L'(\mu) \cup AL(\lambda) \cup AL(\nu) & \text{otherwise,} \end{cases}$$

$$AR(\mu) = \begin{cases} R'(\mu) & \mu \text{ is a leaf,} \\ R'(\mu) \cup AR(\lambda) \cup AR(\nu) & \text{otherwise,} \end{cases}$$

where λ and ν are the children of μ should μ be an internal node. This immediately implies that the query time increases to $O(\log n \log N) = O(\log n \log \log n)$, since, given a query value x, determining the predecessor of x in $L(\mu)$ requires $O(\log N) = O(\log \log n)$ time given the position of x in $AL(\mu)$. Nevertheless, this modification has a worthwhile consequence—it reduces the time for *InsertVertex* operations to $O(1)$ (amortized). This bound follows from the fact that an *InsertVertex* requires more than $O(1)$ time only if that *InsertVertex* operation causes a bucket size to grow larger than $2N$. In such a case we simply split this bucket into two buckets with sizes N and $N+1$, respectively. Of course, splitting a bucket in, say, $L(\mu)$ requires that we add a

new element to $L'(\mu)$ and, hence, to the AL list at each node from μ to the root of B. Since all of these updates can be implemented in $O(\log n + N) = O(N)$ time, we can charge this cost to the N operations that previously inserted elements to this bucket (to make it grow to size $2N$). Thus the amortized cost of an *InsertVertex* is $O(1)$.

We can also reduce the space of this method to $O(n + n \log n/N) = O(n)$, since N is $\Theta(\log n)$, by insuring that any time two consecutive buckets have size smaller than $\lfloor N/2 \rfloor$ we concatenate these buckets into a single bucket.

These modifications do not reduce the cost for *InsertEdge* operations, however. To reduce their cost we apply the bucketing idea a second time, this time to the tree B itself. In particular, we modify our maintenance of B so that, instead of associating a single node of D (corresponding to a single region of $\mathcal{R}$) with each leaf of B, we associate a subtree of D with at most $2N$ nodes. Any time an *InsertEdge* operation causes a leaf subtree D_μ to grow to more than $2N$ nodes, we locate the centroid edge in D_μ and split D_μ into two trees D_{μ_1} and D_{μ_2}, creating two new nodes μ_1 and μ_2, which become the children of μ. This of course necessitates that we build new lists (L, R, L', R', AL, and AR) for μ_1 and μ_2 and update the AL and AR lists from μ to the root to reflect the addition of any new values (needed to maintain the recursive definitions of the AL and AR lists). Nevertheless, this can all be done in $O(\log n + N) = O(N)$ time, which can be charged to the N previous *InsertEdge* operations at μ (each were implemented in $O(1)$ time). Thus each *InsertEdge* will run in $O(1)$ amortized time.

The method for implementing *InsertChain* operations is basically a combination of the methods for *InsertVertex* and *InsertEdge*, the details of which we leave to the interested reader. Thus we have the following theorem.

THEOREM 6.3. *One can maintain a monotone subdivision on-line with the query time $O(\log n \log\log n)$ for point locations and $O(1)$ amortized time for vertex and edge insertions (inserting a chain of k vertices requires $O(k)$ amortized time). The space needed for this data structure is $O(n)$.*

We believe this theorem provides the first on-line point-location data structure with an $O(1)$ amortized update time and polylogarithmic query time.

7. Conclusion. We have given a new approach to planar point location and showed how it can be used to derive new, improved bounds for dynamic point location, spatial point location, and on-line point location. We leave as an open problem the existence of a fully dynamic method for point location in subdivisions that are at least as combinatorially rich as the monotone subdivisions that runs in $O(\log n)$ time per query and $O(\log n)$ amortized time per update. As mentioned in the introduction, Atallah, Goodrich, and Ramaiyer [2] achieve this result for the fairly restrictive class of staircase subdivisions.

Acknowledgments. We would like to thank Bernard Chazelle for suggesting the pursuit of an on-line point-location method with $O(1)$ amortized update time. We would also like to thank Siu-Wing Cheng, Ravi Janardan, and S. Rao Kosaraju for several stimulating conversations related to topics addressed in this paper.

REFERENCES

[1] M. J. ATALLAH, *Parallel techniques for computational geometry*, Proc. IEEE, 80 (1992), pp. 1435–1448.

[2] M. J. ATALLAH, M. T. GOODRICH, AND K. RAMAIYER, *Biased finger trees and three-dimensional layers of maxima*, in Proc. 10th Ann. ACM Sympos. Comput. Geom., 1994, pp. 150–159.

[3] H. Baumgarten, H. Jung, and K. Mehlhorn, *Dynamic point location in general subdivisions*, J. Algorithms, 17 (1994), pp. 342–380.
[4] J. L. Bentley and D. Wood, *An optimal worst case algorithm for reporting intersections of rectangles*, IEEE Trans. Comput., C-29 (1980), pp. 571–577.
[5] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North–Holland, New York, 1976.
[6] B. Chazelle, *A theorem on polygon cutting with applications*, in Proc. 23rd Ann. IEEE Sympos. Found. Comput. Sci., 1982, pp. 339–349.
[7] B. Chazelle, *Triangulating a simple polygon in linear time*, Discrete Comput. Geom., 6 (1991), pp. 485–524.
[8] B. Chazelle and L. J. Guibas, *Fractional cascading:* I. *A data structuring technique*, Algorithmica, 1 (1986), pp. 133–162.
[9] B. Chazelle and L. J. Guibas, *Fractional cascading:* II. *Applications*, Algorithmica, 1 (1986), pp. 163–191.
[10] B. Chazelle and L. J. Guibas, *Visibility and intersection problems in plane geometry*, Discrete Comput. Geom., 4 (1989), pp. 551–581.
[11] S. W. Cheng and R. Janardan, *New results on dynamic planar point location*, SIAM J. Comput., 21 (1992), pp. 972–999.
[12] Y.-J. Chiang, F. P. Preparata, and R. Tamassia, *A unified approach to dynamic point location, ray shooting, and shortest paths in planar maps*, in Proc. 4th ACM-SIAM Sympos. Discrete Algorithms, 1993, pp. 44–53.
[13] Y.-J. Chiang and R. Tamassia, *Dynamic algorithms in computational geometry*, Proc. IEEE, 80 (1992), pp. 1412–1434.
[14] Y.-J. Chiang and R. Tamassia, *Dynamization of the trapezoid method for planar point location in monotone subdivisions*, Internat. J. Comput. Geom. Appl., 2 (1992), pp. 311–333.
[15] R. Cole, *Searching and storing similar lists*, J. Algorithms, 7 (1986), pp. 202–220.
[16] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan, *Making data structures persistent*, J. Comput. System Sci., 38 (1989), pp. 86–124.
[17] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, EATCS Monogr. Theoret. Comput. Sci., 10 (1987), Springer-Verlag, Heidelberg.
[18] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, *Optimal point location in a monotone subdivision*, SIAM J. Comput., 15 (1986), pp. 317–340.
[19] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung, *Maintenance of a minimum spanning forest in a dynamic planar graph*, J. Algorithms, 13 (1992), pp. 33–54.
[20] O. Fries, *Zerlegung einer planaren Unterteilung der Ebene und ihre Anwendungen*, Master's thesis, Inst. Angew. Math. Inform., Univ. Saarlandes, Saarbrücken, 1985.
[21] O. Fries, K. Mehlhorn, and S. Näher, *Dynamization of geometric data structures*, in Proc. 1st Ann. ACM Sympos. Comput. Geom., 1985, pp. 168–176.
[22] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan, *Triangulating a simple polygon*, Inform. Process. Lett., 7 (1978), pp. 175–179.
[23] M. T. Goodrich and R. Tamassia, *Dynamic ray shooting and shortest paths via balanced geodesic triangulations*, in Proc. 9th Ann. ACM Sympos. Comput. Geom., 1993, pp. 318–327.
[24] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan, *Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons*, Algorithmica, 2 (1987), pp. 209–233.
[25] L. J. Guibas and J. Stolfi, *Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams*, ACM Trans. Graph., 4 (1985), pp. 74–123.
[26] D. G. Kirkpatrick, *Optimal search in planar subdivisions*, SIAM J. Comput., 12 (1983), pp. 28–35.
[27] D. T. Lee and F. P. Preparata, *Location of a point in a planar subdivision and its applications*, SIAM J. Comput., 6 (1977), pp. 594–606.
[28] D. T. Lee and F. P. Preparata, *Computational geometry: A survey*, IEEE Trans. Comput., C-33 (1984), pp. 1072–1101.
[29] C. Levcopoulos and M. H. Overmars, *A balanced search tree with $O(1)$ worst-case update time*, Acta Inform., 26 (1988), pp. 269–277.
[30] E. M. McCreight, *Priority search trees*, SIAM J. Comput., 14 (1985), pp. 257–276.
[31] N. Megiddo, *Linear-time algorithms for linear programming in R^3 and related problems*, SIAM J. Comput., 12 (1983), pp. 759–776.
[32] K. Mehlhorn, *Sorting and Searching*, Data Structures and Algorithms, vol. 1, Springer-Verlag, Heidelberg, 1984.

[33] J. O'Rourke, *Computational geometry*, Ann. Rev. Comput. Sci., 3 (1988), pp. 389–411.
[34] M. H. Overmars, *The design of dynamic data structures*, Lecture Notes in Computer Science 156, Springer-Verlag, New York, 1983.
[35] M. H. Overmars, *Range searching in a set of line segments*, in Proc. 1st Ann. ACM Sympos. Comput. Geom., 1985, pp. 177–185.
[36] F. P. Preparata, *A new approach to planar point location*, SIAM J. Comput., 10 (1981), pp. 473–482.
[37] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
[38] F. P. Preparata and R. Tamassia, *Fully dynamic point location in a monotone subdivision*, SIAM J. Comput., 18 (1989), pp. 811–830.
[39] F. P. Preparata and R. Tamassia, *Dynamic planar point location with optimal query time*, Theoret. Comput. Sci., 74 (1990), pp. 95–114.
[40] F. P. Preparata and R. Tamassia, *Efficient point location in a convex spatial cell-complex*, SIAM J. Comput., 21 (1992), pp. 267–280.
[41] N. Sarnak and R. E. Tarjan, *Planar point location using persistent search trees*, Comm. Assoc. Comput. Mach., 29 (1986), pp. 669–679.
[42] D. D. Sleator and R. E. Tarjan, *A data structure for dynamic trees*, J. Comput. System Sci., 26 (1983), pp. 362–381.
[43] D. E. Willard and G. S. Lueker, *Adding range restriction capability to dynamic data structures*, J. Assoc. Comput. Mach., 32 (1985), pp. 597–617.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 637–651

QUERY ORDER*

LANE A. HEMASPAANDRA†, HARALD HEMPEL‡, AND GERD WECHSUNG§

Abstract. We study the effect of query order on computational power and show that $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$—the languages computable via a polynomial-time machine given one query to the jth level of the boolean hierarchy followed by one query to the kth level of the boolean hierarchy—equals $\mathrm{R}^p_{j+2k-1\text{-tt}}(\mathrm{NP})$ if j is even and k is odd and equals $\mathrm{R}^p_{j+2k\text{-tt}}(\mathrm{NP})$ otherwise. Thus unless the polynomial hierarchy collapses it holds that, for each $1 \leq j \leq k$, $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]} = \mathrm{P}^{\mathrm{BH}_k[1]:\mathrm{BH}_j[1]} \iff (j = k) \vee (j \text{ is even} \wedge k = j+1)$. We extend our analysis to apply to more general query classes.

Key words. ordered computation, bounded queries, computational complexity theory

AMS subject classifications. 68Q15, 68Q10, 03D10, 03D15

PII. S0097539796297632

1. Introduction.

This paper studies the importance of query order. Everyone knows that it makes more sense to first look up in your on-line date book the date of the yearly computer science conference and then phone your travel agent to get tickets, as opposed to first phoning your travel agent (without knowing the date) and then consulting your on-line date book to find the date. In real life, order matters.

This paper seeks to determine—for the first time to the best of our knowledge—whether one's everyday-life intuition that order matters carries over to complexity theory.

In particular, for classes $\mathcal{C}_1$ and $\mathcal{C}_2$ from the boolean hierarchy [10, 11], we ask whether one question to $\mathcal{C}_1$ followed by one question to $\mathcal{C}_2$ is more powerful than one question to $\mathcal{C}_2$ followed by one question to $\mathcal{C}_1$. That is, we seek the relative powers of the classes $\mathrm{P}^{\mathcal{C}_1[1]:\mathcal{C}_2[1]}$ and $\mathrm{P}^{\mathcal{C}_2[1]:\mathcal{C}_1[1]}$.

As is standard [26], for any constant m we say A is m-truth-table reducible to B ($A \leq^p_{m\text{-tt}} B$) if there is a polynomial-time computable function that, on each input x, computes both (a) m strings $x_1, x_2, \ldots, x_m$ and (b) a boolean function, α, of m boolean variables, such that $x_1, x_2, \ldots, x_m$ and α satisfy

$$x \in A \iff \alpha(\chi_B(x_1), \chi_B(x_2), \ldots, \chi_B(x_m)) = 1,$$

where χ_B denotes the characteristic function of B. For any a and b for which $\leq^b_a$ is defined and for any class $\mathcal{C}$, let $\mathrm{R}^b_a(\mathcal{C}) = \{L \mid (\exists C \in \mathcal{C})[L \leq^b_a C]\}$.

We prove via the mind change technique that for $j, k \geq 1$,

$$\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]} = \begin{cases} \mathrm{R}^p_{j+2k-1\text{-tt}}(\mathrm{NP}) & \text{if } j \text{ is even and } k \text{ is odd,} \\ \mathrm{R}^p_{j+2k\text{-tt}}(\mathrm{NP}) & \text{otherwise.} \end{cases}$$

*Received by the editors January 24, 1996; accepted for publication (in revised form) September 26, 1996; published electronically August 4, 1998. This research was supported in part by grants NSF-INT-9513368/DAAD-315-PRO-fo-ab and NSF-CCR-9322513.

http://www.siam.org/journals/sicomp/28-2/29763.html

†Department of Computer Science, University of Rochester, Rochester, NY 14627 (lane@cs.rochester.edu). This work was done in part while this author was visiting Friedrich-Schiller-Universität Jena.

‡Institut für Informatik, Friedrich-Schiller-Universität Jena, 07740 Jena, Germany (hempel@informatik.uni-jena.de).

§Institut für Informatik, Friedrich-Schiller-Universität Jena, 07740 Jena, Germany (wechsung@informatik.uni-jena.de).

Informally, this says that the second query counts more toward the power of the class than the first query does. In particular, assuming that the polynomial hierarchy does not collapse, we have that if $1 \leq j \leq k$ then $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ and $\mathrm{P}^{\mathrm{BH}_k[1]:\mathrm{BH}_j[1]}$ differ unless $(j = k) \vee (j \text{ is even } \wedge\, k = j + 1)$.

The interesting case is the strength of $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ when j is even and k is odd. In some sense, $j + 2k$ NP questions underpin this class. However, by arguing that a certain underlying graph must contain an odd cycle, we show that one can always make do with $j + 2k - 1$ queries. We generalize our results to apply broadly to classes with tree-like query structure.

The work of this paper (especially Theorem 3.6), which first appeared in [21], should be compared with the independent work of Agrawal, Beigel, and Thierauf [1]. In particular, let $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]_+}$ denote the class of languages recognized by some polynomial-time machine making one query to a BH_j oracle followed by one query to a BH_k oracle and accepting if and only if the second query is answered "yes." Agrawal, Beigel, and Thierauf prove (using different notation)

$$\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]_+} = \begin{cases} \mathrm{BH}_{j+2k-1} & \text{if } j \not\equiv k \pmod 2, \\ \mathrm{BH}_{j+2k} & \text{otherwise.} \end{cases}$$

Note that this result is incomparable with the results of Theorem 3.6, as their result deals with a different and seemingly more restrictive acceptance mechanism. Some insight into the degree of restrictiveness of their acceptance mechanism, and its relationship to ours, is given by the following claim (Corollary 3.7), which follows immediately from Theorem 5.7 and Lemma 5.9 of [1] and Theorem 3.6 of the present paper:

$$\mathrm{R}^p_{1\text{-tt}}(\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]_+}) = \begin{cases} \mathrm{P}^{\mathrm{BH}_{j-1}[1]:\mathrm{BH}_k[1]} & \text{if } j \text{ is odd and } k \text{ is even,} \\ \mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]} & \text{otherwise.} \end{cases}$$

2. Preliminaries. For standard notions not defined here, we refer the reader to any computational complexity textbook, e.g., [6, 2, 27]. Let χ_A, m-truth-table reducibility ("$\leq^p_{m\text{-tt}}$"), $\mathrm{R}^b_a(\mathcal{C})$, and $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]_+}$ be as defined in section 1.

The boolean hierarchy [10, 11] is defined as follows, where $\mathcal{C}_1 \ominus \mathcal{C}_2 = \{L \mid (\exists A \in \mathcal{C}_1)(\exists B \in \mathcal{C}_2)[L = A - B]\}$:

$$\mathrm{BH}_1 = \mathrm{NP},$$

$$\mathrm{BH}_k = \mathrm{NP} \ominus \mathrm{BH}_{k-1}, \text{ for } k > 1,$$

$$\mathrm{coBH}_k = \{L \mid \overline{L} \in \mathrm{BH}_k\}, \text{ for } k > 0, \text{ and}$$

$$\mathrm{BH} = \bigcup_{i \geq 1} \mathrm{BH}_i.$$

The boolean hierarchy has been intensely investigated, and quite a bit has been learned about its structure (see, e.g., [10, 11, 9, 25, 24, 30, 14, 3, 5]). Recently, various results have also been developed regarding boolean hierarchies over classes other than NP [7, 13, 5, 22].

For any language classes $\mathcal{C}_1$ and $\mathcal{C}_2$, define $\mathrm{P}^{\mathcal{C}_1[1]:\mathcal{C}_2[1]}$ to be the class of languages accepted by polynomial-time machines making one query to a $\mathcal{C}_1$ oracle followed by one query to a $\mathcal{C}_2$ oracle. For any language classes $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$, define $\mathrm{P}^{\mathcal{C}_1[1]:\mathcal{C}_2[1],\mathcal{C}_3[1]}$ to be the class of languages accepted by polynomial-time machines making one query to a $\mathcal{C}_1$ oracle followed in the case of a "no" answer to this first query by one query to a $\mathcal{C}_2$ oracle, and in the case of a "yes" answer to the first query by one query to a $\mathcal{C}_3$ oracle.

3. The importance of query order. We ask whether the order of queries matters. We will study this in the setting of the boolean hierarchy. In particular, does $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ equal $\mathrm{P}^{\mathrm{BH}_k[1]:\mathrm{BH}_j[1]}$, or are they incomparable, or does one strictly contain the other?

For clarity of presentation, in this section we will handle classes only of the form $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$. We show that for no j, k, j', and k' are $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ and $\mathrm{P}^{\mathrm{BH}_{j'}[1]:\mathrm{BH}_{k'}[1]}$ incomparable. In section 4 we will handle the more general case of classes of the form $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1],\mathrm{BH}_l[1]}$, and even classes with a more complicated tree-like query structure.

Indeed, we show in this section that in almost all cases, $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ is so powerful that it can do anything that can be done with $j+2k$ truth-table queries to NP. Since, based on the answer to the first BH_j query, there are two possible BH_k queries that might follow, $j+2k$ is exactly the number of queries asked in a brute force truth-table simulation of $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$. Thus, our result shows that (in almost all cases) the power of the class is not reduced by the nonlinear structure of the $j+2k$ queries underlying $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$—that is, the power is not reduced by the fact that in any given run only $j+k$ underlying NP queries will be even implicitly asked (via the BH_j query and the one asked BH_k query). We say "in almost all cases" as if j is even and k is odd, we prove there is a power reduction of exactly one level.

All the results of the previous paragraph follow from a general characterization that we prove. For $j, k \geq 1$,

$$\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]} = \begin{cases} \mathrm{R}^p_{j+2k-1\text{-tt}}(\mathrm{NP}) & \text{if } j \text{ is even and } k \text{ is odd,} \\ \mathrm{R}^p_{j+2k\text{-tt}}(\mathrm{NP}) & \text{otherwise.} \end{cases}$$

Our proof employs the mind change technique, which predates complexity theory. In particular we show that $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ has at most $j+2k$ ($j+2k-1$ if j is even and k is odd) mind changes, and that BH_{j+2k} (BH_{j+2k-1} if j is even and k is odd) is contained in $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$.

The mind change technique or equivalent manipulation was applied to complexity theory in each of the early papers on the boolean hierarchy, including the work of Cai et al. ([10]; see also [32, 12]); Köbler, Schöning, and Wagner [25]; and Beigel [3]. These papers use mind changes for a number of purposes. Most crucially they use the maximum number of mind changes (what a mind change is will soon be made clear) of a class as an upper bound that can be used to prove that the class is contained in some other class. In the other direction, they also use the number of mind changes that certain classes—especially the classes of the boolean hierarchy due to their normal form as nested subtractions of telescoping sets [10]—possess to show that they can simulate other classes. Even for classes that have the same number of mind changes, relativized separations are obtained via showing that the mind changes are of different character (mind change sequences are of two types, depending on whether they start with acceptance or rejection). The technique has also proven useful in many other more recent papers, e.g., [14, 13, 5].

To make clear the basic nature of mind change arguments, in a simple form, we give an example. We informally argue that each set that is k-truth-table reducible to NP is in fact in $\mathrm{R}^p_{1\text{-tt}}(\mathrm{BH}_k)$.

LEMMA 3.1. *For every $k \geq 1$, $\mathrm{R}^p_{k\text{-}tt}(\mathrm{NP}) = \mathrm{R}^p_{1\text{-}tt}(\mathrm{BH}_k)$.*

Proof. This fact (stated slightly differently) is due to Köbler, Schöning, and Wagner [25], and the proof flavor presented here is most akin to the approach of Beigel [3]. Consider a k-truth-table reduction to an NP set F. Let L be the language accepted by the k-truth-table reduction to F. Consider some input x and without loss of generality assume k queries are generated. Let us suppose for the moment that the reduction rejects when all k queries receive the answer "no." Consider the k-dimensional hypercube such that one dimension is associated with each query (0 in that dimension means the query is answered no and 1 means it is answered yes). So the origin is associated with all queries getting the answer no, and the point (1,1,...,1) is associated with all queries getting the answer yes. Now, also label each vertex with either A (Accept) or R (Reject) based on what the truth-table would do given the answers represented by that vertex. So under our supposition the origin has the label R. Finally, label each vertex with an integer as follows. Label the origin with 0. Inductively label each remaining vertex with the *maximum* integer induced by the vertices that immediately precede it (i.e., those that are the same as it except one yes answer has been changed to a no answer). A preceding vertex v with integer label i induces in a successor v' the integer $i+1$ if v and v' have different A/R labels, and i if they have the same label. Note that vertices given even labels correspond to rejection and those given odd labels correspond to acceptance. Informally, a mind change is just changing one or more strings from no to yes in a way that moves us from a vertex labeled i to one labeled $i+1$. For $1 \leq i \leq k$, let B_i be the NP set that accepts x if (in the queries/labeling generated by the action of the truth-table on input x) for some vertex v labeled i all the queries v claims are yes are indeed in the NP set F. Note that $B_1 \supseteq B_2 \supseteq B_3 \supseteq \cdots$, as if a node labeled v is in B_j, $j \geq 2$, then certainly its predecessor node with label $j-1$ must be in B_{j-1}, as that predecessor represents a subset of the strings v represents. But now note that L is exactly $B_1 - (B_2 - (B_3 - (\cdots - (B_{k-1} - B_k)\cdots)))$. Why? Let the vertex w (say with integer label i_w) represent the true answers to the queries. Note that by construction $x \in B_q$ for all $q \leq i_w$ but $x \notin B_q$ for any $q > i_w$. As the B_i were alternating in terms of representing acceptance and rejection, and given the format $B_1-(B_2-(B_3-(\cdots-(B_{k-1}-B_k)\cdots)))$, the set $B_1-(B_2-(B_3-(\cdots-(B_{k-1}-B_k)\cdots)))$ will do exactly what B_{i_w} represents, namely, the action on the correct answers. Thus, we have just given a proof that a k-truth-table reduction that rejects whenever all answers are no can be simulated by a set in BH_k. Of course, one cannot validly assume that the reduction rejects whenever all answers are no. But it is not hard to see (analogously to the above) that the case of inputs where the reduction accepts when all answers are no can (analogously to the above) be handled via the complement of a BH_k set and that (since what the truth-table reduction does when all answers are no is itself polynomial-time computable) via a set in $\mathrm{R}^p_{1\text{-tt}}(\mathrm{BH}_k)$ we can accept an arbitrary set in $\mathrm{R}^p_{k\text{-tt}}(\mathrm{NP})$. Of course, it is clear by brute force simulation that $\mathrm{R}^p_{1\text{-tt}}(\mathrm{BH}_k) \subseteq \mathrm{R}^p_{k\text{-tt}}(\mathrm{NP})$, and so it holds that $\mathrm{R}^p_{1\text{-tt}}(\mathrm{BH}_k) = \mathrm{R}^p_{k\text{-tt}}(\mathrm{NP})$. □

What actually is being shown above is that $\mathrm{R}^p_{1\text{-tt}}(\mathrm{BH}_k)$ can handle k appropriately structured mind changes, starting either from reject or accept. In the following theorem, the crucial things we show are that (a) $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ can simulate, starting at either accept or reject, $j+2k$ (respectively, $j+2k-1$) mind changes if j is odd

or k is even (respectively, if j is even and k is odd), and (b) for j even and k odd, $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ can *never* have more than $j+2k-1$ mind changes. We achieve (b) by examining the possible mind change flow of a $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine, j even and k odd, and showing that either a mind change is flagrantly wasted, or a certain underlying graph has an odd length directed cycle (which thus is not two-colorable, and from this will lose one mind change).

Since our arguments in the proofs of this section use paths in hypercubes, we will find useful the concept of an ascending path in a hypercube. Let $K = \{0,1\}^d$ be the d-dimensional hypercube. Then every path p in K can be described as a linear combination of unit vectors $u_1, \ldots, u_d$, where u_i is the ith unit vector. We call p an ascending path in K leading from $(0,0,\ldots,0)$ to v if and only if it can be identified with a sum

$$u_{i_1} + u_{i_2} + \cdots + u_{i_n}$$

of distinct unit vectors u_ν such that the vertices of this path p are

$$v_0 = (0,\ldots,0), v_1 = u_{i_1}, v_2 = u_{i_1} + u_{i_2}, \ldots, v = u_{i_1} + u_{i_2} + \cdots + u_{i_n}.$$

We will call this sum the description of p. Note that the order of the u's matters, as a permutation of the u's results in another path. We call p an ascending path (without specifying starting point and endpoint) if p is an ascending path leading from $(0,0,\ldots,0)$ to $(1,1,\ldots,1)$.

Before turning to results, first we will study the structure of ascending paths in labeled hypercubes and give some necessary definitions. Building upon them, we will then prove Lemma 3.5, which states that $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ can handle exactly $j+2k$ ($j+2k-1$ if j is even and k is odd) mind changes.

Let M be a $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine with oracles $A \in \mathrm{BH}_j$ and $B \in \mathrm{BH}_k$ and let $x \in \Sigma^*$. On input x, M first makes a query $q_1(x)$ to A and then if the answer to the first query was "no" asks query $q_2(x)$ to B and if the answer to the first query was "yes" asks query $q_3(x)$ to B. Without loss of generality assume that on every input x exactly two queries are asked.

Every set $C \in \mathrm{BH}_l$ can be written as the nested difference of sets $C_1, C_2, \ldots, C_l \in \mathrm{NP}$

$$C = C_1 - (C_2 - (\cdots - (C_{l-1} - C_l)\cdots))$$

and following Cai et al. [10] we even can assume that

$$C_l \subseteq C_{l-1} \subseteq \cdots \subseteq C_2 \subseteq C_1.$$

Hence a query "$q \in C$?" can certainly be solved via l queries "$q \in C_1$?," "$q \in C_2$?," $\ldots$, "$q \in C_l$?"

In light of this comment, we let

$$A = A_1 - (A_2 - (\cdots - (A_{j-1} - A_j)\cdots)) \quad \text{where} \quad A_i \in \mathrm{NP} \quad \text{for} \quad i = 1,2,\ldots,j \quad \text{and} \quad A_j \subseteq \cdots \subseteq A_1, \text{ and}$$

$$B = B_1 - (B_2 - (\cdots - (B_{k-1} - B_k)\cdots)) \quad \text{where} \quad B_i \in \mathrm{NP} \quad \text{for} \quad i = 1,2,\ldots,k \quad \text{and} \quad B_k \subseteq \cdots \subseteq B_1.$$

For the sake of definiteness let us assume that the queries

$$q_1(x) \in A_1, \ldots, q_1(x) \in A_j, q_2(x) \in B_1, \ldots, q_2(x) \in B_k, q_3(x) \in B_1, \ldots, q_3(x) \in B_k$$

correspond in this order to the $j+2k$ dimensions of the $(j+2k)$-dimensional hypercube $H = \{0,1\}^{j+2k}$. More precisely, a vector $(a_1, \ldots, a_{j+2k}) \in H$ is understood to consist of the answers to the above-mentioned queries, where 0 means "no" and 1 means "yes."

Since a query "$q \in C$?" for some $C \in \mathrm{BH}_l$ and $C = C_1 - (C_2 - (\cdots (C_{l-1} - C_l) \cdots))$ can be solved by evaluating the answers to "$q \in C_1$?," "$q \in C_2$?," $\ldots$, "$q \in C_l$?" every node $v \in H$ gives us answers to "$q_1(x) \in A$?" (by evaluating the first j components of v), to "$q_2(x) \in B$?" (by evaluating the k components of v that immediately follow the first j components of v), and to "$q_3(x) \in B$?" (by evaluating the last k of v's components). This gives us a labeling of all vertices of H. We simply assign label A (Accept) to vertex $v \in H$ if $M^{A[1]:B[1]}(x)$ accepts if the answers to the two asked questions are as determined by v. If $M^{A[1]:B[1]}(x)$ rejects in this case we assign label R (Reject) to v.

So let $H_M(x)$ be the $(j+2k)$-dimensional hypercube labeled according to $M^{A[1]:B[1]}(x)$. The number of mind changes on an ascending path p of $H_M(x)$ leading from $(0,0,\ldots,0)$ to a vertex t is by definition the number of label changes when moving from $(0,0,\ldots,0)$ to t along p. The number of mind changes of an internal node v of $H_M(x)$ is the maximum number of mind changes on an ascending path leading from $(0,0,\ldots,0)$ to v. And, finally, the number of mind changes of a $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine M is by definition the maximum number (we take the maximum over all $x \in \Sigma^*$) of mind changes of the vertex $(1,1,\ldots,1)$ in $H_M(x)$; in other words, this number is the maximum number of label changes on an ascending path in $H_M(x)$ for some $x \in \Sigma^*$.

We say we lose a mind change (between two adjacent vertices v_i and v_{i+1}) along an ascending path if when moving from v_i to v_{i+1} the machine does not change its acceptance behavior.

One can easily verify the following fact.

FACT 3.2. *If M is a $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine such that on input x the acceptance behavior is independent of the answer to one or more of the two possible second queries (that is, if for at least one of the second queries both a "yes" and a "no" answer yield the same acceptance or rejection behavior), then we lose at least one mind change on every path in $H_M(x)$.*

So from now on, in light of Fact 3.2, let $M^{A[1]:B[1]}(x)$ be a $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine that has, on input x, one of the following four acceptance schemes (the scheme may depend on the input).

(1) M accepts if and only if exactly one of the two sequential queries is answered "yes."
(2) M accepts if and only if either both or neither of the two asked queries is answered "yes."
(3) M accepts if and only if the second query is answered "yes."
(4) M accepts if and only if the second query is answered "no."

FACT 3.3. *If p is an ascending path in $H_M(x)$ such that p contains adjacent vertices v and $v + u_d$ such that*

$$d \leq j \text{ and the } (d')\text{th component of } v \text{ is } 0 \text{ for some } d' < d,$$

then p loses a mind change.

Proof. Since $A \in \mathrm{BH}_j$ and thus $A = A_1 - (A_2 - (\cdots - (A_{j-1} - A_j) \cdots))$ and there is a 0 in the (d')th component of v and $v + u_d$, both vertices yield the same answer

to "$q_1(x) \in A$?" The 1 in the dth component of $v + u_d$ has no effect at all on the answer to "$q_1(x) \in A$?" and so on the outcome of $M^{A[1]:B[1]}(x)$. Hence both vertices have the same label and p loses a mind change. □

Similarly, one can prove that if p is an ascending path and p contains two adjacent vertices v and $v + u_d$ such that $j < d' < d \leq j + k$ and the (d')th component of v is 0 or $j + k < d' < d \leq j + 2k$ and the (d')th component of v is 0, then p also loses one mind change.

Furthermore, in light of Fact 3.3, let us focus only on paths p that change their first j, second k, and last k dimensions from the smallest to the highest dimension in each group. This allows us to simplify the description of paths as follows. Let e_1 be the following operator on H:

$$e_1((a_1, \ldots, a_{j+2k})) = \begin{cases} (a_1, \ldots, a_{i-1}, 1, \ldots, a_{j+2k}) & \text{if } i \leq j,\ a_i = 0 \wedge (\forall j : j < i)[a_j = 1], \\ (a_1, \ldots, a_{j+2k}) & \text{otherwise.} \end{cases}$$

The operators e_2 and e_3 act on the index groups $(j + 1, \ldots, j + k)$ and $(j + k + 1, \ldots, j + 2k)$, respectively, in the same manner: the zero component with smallest index among the zero components is incremented by 1. The only reasonable paths to consider are those emerging from repeated applications of e_1, e_2, and e_3 to $(0, \ldots, 0)$. We will use $(e_{i_1}, e_{i_2}, \ldots, e_{i_{j+2k}})$ to denote the path with vertices $v_0 = (0, \ldots, 0)$, $v_1 = e_{i_1}(v_0), v_2 = e_{i_2}(v_1), \ldots, v_{j+2k} = e_{i_{j+2k}}(v_{j+2k-1}) = (1, 1, \ldots, 1)$.

The next fact gives sufficient conditions for an ascending path to lose a mind change.

FACT 3.4. *On any ascending path p a mind change loss occurs if*

Case 1.1. *there is an e_2 after an odd number of e_1's in the description of p, or*

Case 1.2. *there is an e_3 after an even number of e_1's in the description of p, or*

Case 2. *the description of p contains a sequence of odd length at least 3 that starts and ends with e_1 and contains no other e_1's.*

Proof. We will call the occurrence of Case 1.1 (Case 1.2) in p an "e_2-loss" ("e_3-loss") and the occurrence of Case 2 an "odd episode." In general we call a subpath of p of length at least 3 that starts and ends with e_1 and contains no other e_1 an episode.

Intuitively p loses a mind change in the case of Case 1.1 (Case 1.2), since in the actual computation $M(x)$ does not really ask query $q_2(x)$ $(q_3(x))$ and so a change in the answers to the k underlying NP queries of $q_2(x)$ $(q_3(x))$ does not affect the outcome of the overall computation.

Intuitively in Case 2 the following argument holds. If the description of p contains an odd episode, say starting with $e_{i_l} = e_1$ and ending with $e_{i_{l'}} = e_1$, then $v_{l-1}, v_l, \ldots, v_{l'}$ form an even-length subpath p' of p. If the odd episode contains both e_2's and e_3's then note that Case 1 applies and we are done. In fact due to Case 1 we may henceforth assume the odd episode, between the starting and the ending e_1's, has only e_2's (respectively, e_3's) if we have an even (respectively, odd) number of e_1's up to and including the e_1 starting the odd episode. So in this case v_{l-1} and $v_{l'}$ have the same label Accept/Reject. The acceptance behavior of $M^{A[1]:B[1]}(x)$ due to v_{l-1} and $v_{l'}$ is the same, because after two e_1's the answer to "$q_1(x) \in A$?" is the same as it was before the two e_1's, and the e_2's (e_3's) have not influenced the answer to $q_3(x)$ $(q_2(x))$. Thus we have a subpath of even length, namely, $v_{l-1}, v_l, \ldots, v_{l'}$, whose starting point and endpoint have the same Accept/Reject label. To assign to each vertex of this path an Accept/Reject label in such a way that no mind changes are lost is equivalent to the impossible task of 2-coloring an odd cycle. Hence we lose at least one mind change for every occurrence of an odd episode. □

Before proving the main theorem of this section, we show Lemma 3.5, which tells how many mind changes $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ can handle. We say a complexity class $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ can handle exactly m mind changes if and only if (a) no $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine has more than m mind changes and (b) there is a specific $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine that has m mind changes. It is known (see, e.g., [10, 25, 3]) that $\mathrm{R}^p_{k\text{-tt}}(\mathrm{NP})$ can handle exactly k mind changes.

LEMMA 3.5. *The class* $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ *can handle exactly* m *mind changes, where*

$$m = \begin{cases} j + 2k - 1 & \text{if } j \text{ is even and } k \text{ is odd,} \\ j + 2k & \text{otherwise.} \end{cases}$$

Proof. We first consider the case in which j is even and k is odd.

We want to argue that for every $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine M and every $x \in \Sigma^*$, on every ascending path in the $j + 2k$-dimensional, appropriately labeled hypercube $H_M(x)$ there are at most $j+2k-1$ mind changes. Let $x \in \Sigma^*$ and M be a $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine with the oracles A and B. Due to Facts 3.3 and 3.2, it suffices to consider a $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine M with one of the four previously mentioned acceptance schemes on input x and to show that every path p having the introduced description loses at least one mind change. Let $M(x)$ be such a machine and p be such a path. There are two possibilities.

Case A. The description of p contains an e_2-loss or an e_3-loss.
According to Fact 3.4, p loses at least one mind change.

Case B. The description of p contains neither an e_2-loss nor an e_3-loss.
Hence the description of p consists of blocks of consecutive e_2's and e_3's separated by blocks of e_1's. Since the description of p contains k e_3's and k is odd, there is a block of e_3's of odd size in p. Since we have no e_3-loss and j is even, this block is surrounded by e_1's. Thus we have an odd episode in the description of p and, according to Fact 3.4, p loses a mind change.

So no $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine can realize more than $j + 2k - 1$ mind changes.

It remains to show that there is a deterministic $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine and an input $x \in \Sigma^*$ such that in the associated hypercube $H_M(x)$ there is a path having exactly $j + 2k - 1$ mind changes.

Let us consider the path p_0,

$$p_0 = (\underbrace{e_2, e_2, \ldots, e_2}_{k}, \underbrace{e_1, e_1, \ldots, e_1}_{j-1}, \underbrace{e_3, e_3, \ldots, e_3}_{k}, e_1).$$

Consider the deterministic oracle machine W that asks two sequential queries and accepts an input x if and only if the second query of $W(x)$ was answered "yes" (acceptance scheme (3)). We know as was just shown that all ascending paths of $H_W(x)$ have at most $j + 2k - 1$ mind changes. Note that for every $x \in \Sigma^*$ the path p_0 loses only one mind change and thus $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ can handle exactly $j + 2k - 1$ mind changes.

This completes the proof of the case "j is even and k is odd." We now turn to the "j is odd or k is even" case of the lemma being proven.

Since our hypercube has (in all cases) $j + 2k$ dimensions, certainly $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ can handle (in all cases) no more than $j + 2k$ mind changes.

If j is odd, we consider the path

$$p_1 = (\underbrace{e_2, e_2, \ldots, e_2}_{k}, \underbrace{e_1, e_1, \ldots, e_1}_{j}, \underbrace{e_3, e_3, \ldots, e_3}_{k})$$

and—using the acceptance scheme numbering introduced just after Fact 3.3—we consider the machine having for every input x acceptance scheme (3) or (1) for k odd or even, respectively. If j is even and k is even, we consider path p_0 and we consider the machine having acceptance scheme (1) for every input.

In each of these cases the considered machine changes its mind along the associated path exactly $j+2k$ times. Hence for j odd or k even the class $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ can handle exactly $j+2k$ mind changes. □

Now we are ready to prove our main theorem of this section.

THEOREM 3.6. *For $j,k \geq 1$,*

$$\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]} = \begin{cases} \mathrm{R}^p_{j+2k-1\text{-}tt}(\mathrm{NP}) & \text{if } j \text{ is even and } k \text{ is odd,} \\ \mathrm{R}^p_{j+2k\text{-}tt}(\mathrm{NP}) & \text{otherwise.} \end{cases}$$

Proof. In order to avoid unnecessary case distinctions we prove the fact for arbitrary j and k and simply denote the appropriate number of mind changes by m, namely (see Lemma 3.5), $j+2k-1$ if j is even and k is odd and $j+2k$ otherwise. First we would like to show that $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]} \subseteq \mathrm{R}^p_{m\text{-}\mathrm{tt}}(\mathrm{NP})$. We show this by explicitly giving the appropriate truth-table reduction.

Let $A \in \mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ and let m be the number of mind changes (according to Lemma 3.5) the class $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ can handle. Let M be a deterministic oracle machine, witnessing $A \in \mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$, via the sets $S_1 \in \mathrm{BH}_j$ and $S_2 \in \mathrm{BH}_k$. As noted by Beigel [3], the set $Q = \{\langle x,k\rangle \mid M(x) \text{ has at least } k \text{ mind changes}\}$ is an NP set. Note that if $M^{S_1[1]:S_2[1]}(x)$ on a particular input x rejects (respectively, accepts) if both queries have the answer "no," then $M^{S_1[1]:S_2[1]}(x)$ accepts if and only if the node (of the implicit hypercube) associated with the actual answers has an odd (respectively, even) number of mind changes.

Define the variables $o, y_1, y_2, \ldots, y_m$ and the m-ary boolean function α:

$o = 0$ if $M^{S_1[1]:S_2[1]}(x)$ rejects if both queries are answered "no,"

$o = 1$ if $M^{S_1[1]:S_2[1]}(x)$ accepts if both queries are answered "no,"

$$\begin{aligned} y_1 &= \langle x,1\rangle, \\ y_2 &= \langle x,2\rangle, \\ y_3 &= \langle x,3\rangle, \\ &\vdots \\ y_m &= \langle x,m\rangle, \quad \text{and} \end{aligned}$$

$$\alpha(z_1,z_2,\ldots,z_m) = 1 \iff (\max\{l \mid z_l = 1\} + o) \equiv 1 \pmod 2.$$

Clearly we can compute the just defined variables for a given x and also evaluate the function α at $(\chi_Q(y_1),\chi_Q(y_2),\ldots,\chi_Q(y_m))$ in polynomial time. And, finally, we have $x \in A \iff \alpha(\chi_Q(y_1),\chi_Q(y_2),\ldots,\chi_Q(y_m)) = 1$. Thus $A \in \mathrm{R}^p_{m\text{-}\mathrm{tt}}(\mathrm{NP})$.

It remains to show that $\mathrm{R}^p_{m\text{-}\mathrm{tt}}(\mathrm{NP}) \subseteq \mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$. Recall $\mathrm{R}^p_{k\text{-}\mathrm{tt}}(\mathrm{NP}) = \mathrm{R}^p_{1\text{-}\mathrm{tt}}(\mathrm{BH}_k)$ from Lemma 3.1. Since the class $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ is closed under $\leq^p_{1\text{-}\mathrm{tt}}$ reductions it suffices to prove $\mathrm{BH}_m \subseteq \mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$.

So let $B \in \mathrm{BH}_m$. Following Cai et al. [10] we may assume that the set B is of the form $B = B_1 - (B_2 - (B_3 - (\cdots - (B_{m-1} - B_m)\cdots)))$ with $B_1, B_2, \ldots, B_m \in \mathrm{NP}$ and $B_m \subseteq \cdots \subseteq B_2 \subseteq B_1$.

We show $B \in \mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ by using ideas of the second part of the proof of Lemma 3.5, namely by implementing the specific good path p_0, respectively p_1. B is accepted by a $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine M as follows:

Case 1. j is odd.

Define the two oracle sets O_1 and O_2:

$O_1 = B_{k+1} - (B_{k+2} - (\cdots - (B_{k+j-1} - B_{k+j})\cdots))$, and

$$O_2 = \{\langle y, 2\rangle \mid y \in B_1 - (B_2 - (\cdots - (B_{k-1} - B_k)\cdots))\}$$
$$\cup\{\langle y, 3\rangle \mid y \in B_{j+k+1} - (B_{j+k+2} - (\cdots - (B_{j+2k-1} - B_{j+2k})\cdots))\}.$$

Note that $O_1 \in \mathrm{BH}_j$ and $O_2 \in \mathrm{BH}_k$. On input x M first queries "$x \in O_1$." In case of a "no" answer $M(x)$ queries $\langle x, 2\rangle \in O_2$ and in case of a "yes"answer to the first query $M(x)$ asks $\langle x, 3\rangle \in O_2$.

Case 1.1. k is odd.

$M(x)$ accepts if and only if the second query is answered "yes."

Case 1.2. k is even.

$M(x)$ accepts if and only if exactly one of the two queries is answered "yes."

Case 2. j is even.

Define the two oracle sets O_1 and O_2:

$O_1 = B_{k+1} - (B_{k+2} - (\cdots - (B_{k+j-1} - B_m)\cdots))$, and

$$O_2 = \{\langle y, 2\rangle \mid y \in B_1 - (B_2 - (\cdots - (B_{k-1} - B_k)\cdots))\}$$
$$\cup\{\langle y, 3\rangle \mid y \in B_{j+k} - (B_{j+k+1} - (\cdots - (B_{m-2} - B_{m-1})\cdots))\}.$$

Note that $O_1 \in \mathrm{BH}_j$ and $O_2 \in \mathrm{BH}_k$. On input x M first queries "$x \in O_1$." In case of a "no" answer $M(x)$ queries $\langle x, 2\rangle \in O_2$ and in case of a "yes" answer to the first query $M(x)$ asks $\langle x, 3\rangle \in O_2$.

Case 2.1. k is odd.

$M(x)$ accepts if and only if the second query is answered "yes."

Case 2.2. k is even.

$M(x)$ accepts if and only if exactly one of the two queries is answered "yes." □

It is interesting to note which properties of NP are actually required in the above proof for the result to hold. The proof essentially rests on the fact that the key set Q (describing that, for given x and m, the $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ machine M on input x has at least m mind changes) is an NP set. So considering an arbitrary underlying class $\mathcal{C}$, for proving $Q \in \mathcal{C}$ it suffices to note that Q is in the class $\exists^b \cdot \mathrm{R}^p_{\text{c-btt}}(\mathcal{C})$,[1] and to assume that $\mathcal{C}$ be closed under $\exists^b$ and conjunctive bounded truth-table reductions. Indeed, the $\exists^b$ quantifier describes that there is a path in the boolean hypercube $H_M(x)$, and via the $\leq^p_{\text{c-btt}}$-reduction it can be checked that this path is an ascending path and all the answers the vertices on that path claim to be "yes" answers indeed correspond to query strings that belong to the class $\mathcal{C}$. Similar observations have been stated in earlier papers [3, 5]. In terms of the present paper, note in particular that the assertion of Theorem 3.6 holds true for all classes $\mathcal{C}$ closed under union, intersection, and polynomial-time many-one reductions. $\mathrm{C}_=\mathrm{P}$, R, and FewP all have these closure properties, to name just a few examples. If the underlying class $\mathcal{C}$ is closed under polynomially bounded $\exists$ quantification and unbounded conjunctive truth-table reductions, it is not hard to see that this analysis can even be done safely up to the case of logarithmically bounded query classes, as the number of paths in the hypercube is polynomial and thus generates a polynomial-sized disjunction.

Theorem 3.6 allows us to derive a relationship between classes of the form $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ and $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]_+}$. Classes of the latter form were studied in [1].

[1] Here, $\leq^p_{\text{c-btt}}$ denotes the conjunctive bounded truth-table reducibility, and for any class $\mathcal{K}$, $\exists^b \cdot \mathcal{K}$ is defined to be the class of languages A for which there exists a set $B \in \mathcal{K}$ and a constant bound m such that $x \in A$ if and only if there exists a string y of length at most m with $\langle x, y\rangle \in B$.

COROLLARY 3.7. *For every $j, k \geq 1$,*

$$\mathrm{R}^{p}_{1\text{-}tt}(\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]_+}) = \begin{cases} \mathrm{P}^{\mathrm{BH}_{j-1}[1]:\mathrm{BH}_k[1]} & \text{if } j \text{ is odd and } k \text{ is even,} \\ \mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]} & \text{otherwise.} \end{cases}$$

The proof is immediate by the results of Theorem 3.6 of this paper and Theorem 5.7 and Lemma 5.9 of [1].

From Theorem 3.6 we can immediately conclude that order matters for queries to the boolean hierarchy unless the boolean hierarchy itself collapses.

COROLLARY 3.8.

1. *If $(j = k) \vee (j$ is even and $k = j + 1)$, $1 \leq j \leq k$, then $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]} = \mathrm{P}^{\mathrm{BH}_k[1]:\mathrm{BH}_j[1]}$.*
2. *Unless the boolean hierarchy (and thus the polynomial hierarchy) collapses, for every $1 \leq j \leq k$, $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]} \neq \mathrm{P}^{\mathrm{BH}_k[1]:\mathrm{BH}_j[1]}$ unless $(j = k) \vee (j$ is even and $k = j + 1)$.*

The corollary holds, in light of the theorem, simply because the boolean hierarchy and the truth-table hierarchy are interleaved [25] in such a way that the boolean hierarchy levels are sandwiched between levels of the bounded-truth-table hierarchy, and thus if two different levels of the bounded-truth-table hierarchy are the same (say levels r and s, $r < s$), then some level (in particular, BH_{r+1}) of the boolean hierarchy is closed under complementation, and thus, by the downward separation property of the boolean hierarchy [10], the boolean hierarchy would collapse. Furthermore, Kadin [24] has shown that if the boolean hierarchy collapses then the polynomial hierarchy collapses, and Wagner [28, 29], Chang and Kadin [14], and Beigel, Chang, and Ogiwara [5] have improved the strength of this connection. The strongest known connection is: If $\mathrm{BH}_q = \mathrm{coBH}_q$, then $\mathrm{PH} = (\mathrm{P}^{\mathrm{NP}}_{(q-1)\text{-tt}})^{\mathrm{NP}}$ [5], where $(\mathrm{P}^{\mathrm{NP}}_{m\text{-tt}})^{\mathrm{NP}}$ denotes the class of languages accepted by P machines given m-truth-table access to an $\mathrm{NP}^{\mathrm{NP}}$ oracle and also given unlimited access to an NP oracle (note that $(\mathrm{P}^{\mathrm{NP}}_{1\text{-tt}})^{\mathrm{NP}}$ is equal to $\mathrm{P}^{\mathrm{NP}^{\mathrm{NP}}[1]:\mathrm{NP}}$ as leading NP queries can be absorbed into the $\mathrm{NP}^{\mathrm{NP}}$ query).

In light of this discussion, we can make more clear exactly what collapse is spoken of in the second part of the above corollary. In particular, the collapse of the polynomial hierarchy is (at least) to $(\mathrm{P}^{\mathrm{NP}}_{(k+2j)\text{-tt}})^{\mathrm{NP}}$.[2]

Note added in proof. A connection, stronger than that obtained in [5], between boolean and polynomial hierarchy collapses has very recently been obtained by Hemaspaandra, Hemaspaandra, and Hempel [33] and Reith and Wagner [34].

Of course, Theorem 3.6 applies far more generally. From it, for any j, k, j', and k', one can either immediately conclude equality, or can immediately conclude that the classes are not equal unless the polynomial hierarchy collapses to $(\mathrm{P}^{\mathrm{NP}}_{(\min(\alpha(j,k),\, \alpha(j',k')))\text{-tt}})^{\mathrm{NP}}$, where $\alpha(a, b)$ equals $a + 2b - 1$ if a is even and b is odd and $a + 2b$ otherwise.

The point of Theorem 3.6 is that from the even/odd structure of $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ classes one can immediately tell their number of mind changes, and thus their strength, without having to do a separate, detailed, mind change analysis for each j and k pair. However, note that one can, via a time-consuming but mechanical procedure, analyze

[2] Although one level is gained by the $q - 1$ in the [5] connection between the boolean hierarchy and the polynomial hierarchy, one level is lost in the collapse of the boolean hierarchy that follows from a given collapse in the truth-table hierarchy. We speculate that it might be possible for the $k + 2j$ claim to be strengthened by one level by applying the [5] technique directly to the truth-table hierarchy.

almost any class with a query tree structure (namely, by looking at the full tree of possible queries and answers, and for each of the huge number of possible ways its leaves can each be labeled accept-reject compute the number of mind changes that labeling creates, and then look at the maximum over all these numbers). For example, one can quickly see that one query to DP followed by 4-tt access to NP yields exactly the languages in $\mathrm{R}^p_{10\text{-}\mathrm{tt}}(\mathrm{NP})$.

4. General case. In the previous section, we studied classes of the form $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$. We completely characterized them in terms of reducibility hulls of NP and noted that in this setting the order of access to different oracles matters quite a bit. What can be said about, for example, the class $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]:\mathrm{BH}_l[1]}$? Is it equal to $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1],\mathrm{BH}_l[1]}$? (We'll see that the answer is "no" in certain cases.) Even more generally, what can be said about the classes of languages that are accepted by deterministic oracle machines with tree-like query structures and with each query being made to a (potentially) different oracle from a (potentially) different level of the boolean hierarchy? Is it possible that with a more complicated query structure we might lose even more than the one mind change lost in the case of $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1]}$ with j even and k odd? (From the results of the section, it will be clear that the answer to this question is "yes"; mind changes can, in certain specific circumstances, accumulate.)

First , we can immediately derive a characterization of the class $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1],\mathrm{BH}_l[1]}$ from the results of the previous section, namely, we have Theorem 4.1.

THEOREM 4.1. *For $j,k,l \geq 1$,*

$$\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1],\mathrm{BH}_l[1]} = \begin{cases} \mathrm{R}^p_{j+k+l-1\text{-}tt}(\mathrm{NP}) & \text{if } j \text{ is even and } l \text{ is odd,} \\ \mathrm{R}^p_{j+k+l\text{-}tt}(\mathrm{NP}) & \text{otherwise.} \end{cases}$$

Proof. Note that in Lemma 3.5 we handle the special case of $k = l$. However, notice that the mind change loss for j even and k odd is due only to the fact that the query made after the first query is answered "yes" is made to an oracle from an odd level, namely, k, of the boolean hierarchy. In particular the mind change loss is not tied to the query we ask in case the first query is answered "no." Thus we have the following claim.

Claim. The class $\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1],\mathrm{BH}_l[1]}$ can handle exactly m mind changes where

$$m = \begin{cases} j+k+l-1 & \text{if } j \text{ is even and } l \text{ is odd,} \\ j+k+l & \text{otherwise.} \end{cases}$$

Similar to the proof of Theorem 3.6 one can now show the equality we claim. □

Note that for every $j,k,l \geq 1$, we obviously have

$$\mathrm{P}^{\mathrm{BH}_j[1]:\mathrm{BH}_k[1],\mathrm{BH}_l[1]} = \mathrm{P}^{\mathrm{R}^p_{1\text{-}\mathrm{tt}}(\mathrm{BH}_j)[1]:\mathrm{R}^p_{1\text{-}\mathrm{tt}}(\mathrm{BH}_k)[1],\mathrm{R}^p_{1\text{-}\mathrm{tt}}(\mathrm{BH}_l)[1]},$$

and thus the following corollary holds.

COROLLARY 4.2. *For $j,k,l \geq 1$,*

$$\mathrm{P}^{\mathrm{R}^p_{1\text{-}tt}(\mathrm{BH}_j)[1]:\mathrm{R}^p_{1\text{-}tt}(\mathrm{BH}_k)[1],\mathrm{R}^p_{1\text{-}tt}(\mathrm{BH}_l)[1]} = \begin{cases} \mathrm{R}^p_{j+k+l-1\text{-}tt}(\mathrm{NP}) & \text{if } j \text{ is even and } l \text{ is odd,} \\ \mathrm{R}^p_{j+k+l\text{-}tt}(\mathrm{NP}) & \text{otherwise.} \end{cases}$$

The last corollary is the key tool to use in evaluating any class of languages that is accepted by deterministic oracle machines with tree-like query structures and with

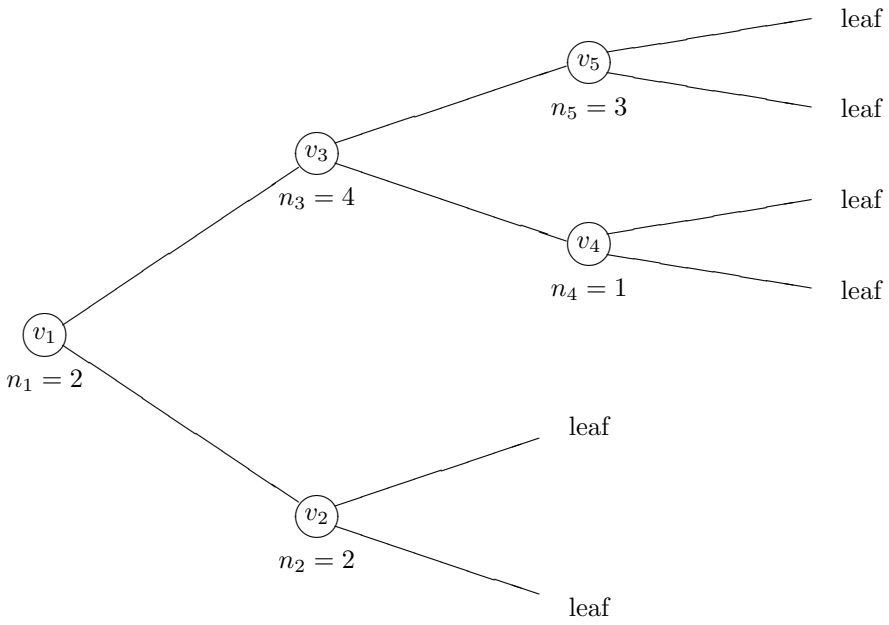


FIG. 1. *Tree* $\mathcal{T}$.

each query being made to a (potentially) different oracle from a (potentially) different level of the boolean hierarchy.

We formalize some notions to use in studying this. Let T be a binary tree, not necessarily complete, such that each internal node v_i (a) has exactly two children and (b) is labeled by a natural number n_i (whose purpose will be explained below). For such a tree T, define f_T by $f_T(v_i) = n_i$. Henceforward, we will write f for f_T in contexts in which T is clear. Let $root_T$ be the root of the tree (we will assign to this node the name v_1) and let LT_T and RT_T, respectively, be the left and right subtrees of the root. We will denote the class of sets that are accepted by a deterministic oracle machine with a T-like query structure by $\mathrm{P}^{(T)}$. Here the structure of the tree T gives the potential computation tree of every $\mathrm{P}^{(T)}$ machine in the sense that inductively if a query at node v is answered "no" ("yes") we keep on moving through the tree in the left (right) subtree of v. And at each internal node v_i of T the natural number n_i gives the level of the boolean hierarchy from which the oracle queried at that node is taken.

For example consider the tree $\mathcal{T}$ (see Figure 1), in which $f(v_1) = 2$, $f(v_2) = 2$, $f(v_3) = 4$, $f(v_4) = 1$, and $f(v_5) = 3$. A $\mathrm{P}^{(\mathcal{T})}$ machine works as follows. The first query is made to a DP oracle. If the answer to that first query is "no" a second query is made to the DP oracle associated with v_2, and if the answer to the first query is "yes" the second query is made to the BH_4 oracle associated with v_3. A third query is made only if the answer to the first query is "yes"; in this case, the oracle set of the third query is in NP if the answer to the second query is "no," and is in BH_3 if the answer to the second query is "yes." Note that for every input $x \in \Sigma^*$ every $\mathrm{P}^{(\mathcal{T})}$

machine $M(x)$ assigns a label A (Accept) or R (Reject) to each leaf of $\mathcal{T}$ with its own specific acceptance behavior (which, in particular, may depend on x).

If T is the complete tree of depth 1 (i.e., a root plus two leaves), then by definition $m(T) = f(root_T)$, and otherwise define

$$m(T) = \begin{cases} f(root_T) + m(LT_T) + m(RT_T) - 1 & \text{if } f(root_T) \equiv 0 \pmod 2 \text{ and} \\ & \quad m(RT_T) \equiv 1 \pmod 2, \\ f(root_T) + m(LT_T) + m(RT_T) & \text{otherwise.} \end{cases}$$

For our example tree $\mathcal{T}$ we have $m(\mathcal{T}) = 10$. The main theorem of this section will prove $m(T)$ determines the number of bounded truth-table accesses to NP that completely characterize the class $\mathrm{P}^{(T)}$. It follows from the main theorem that, for example, $\mathrm{P}^{(\mathcal{T})} = \mathrm{R}^p_{10\text{-tt}}(\mathrm{NP})$.

THEOREM 4.3. $\mathrm{P}^{(T)} = \mathrm{R}^p_{m(T)\text{-}tt}(\mathrm{NP})$.

Proof. The proof consists of an obvious induction over the depth d of the tree. Note that the correctness of the base case of the induction, $d = 2$, is given by Theorem 4.1. The proof of the inductive step follows immediately from the obvious fact that

$$\mathrm{P}^{(T)} = \mathrm{P}^{\mathrm{BH}_{f(root_T)}:\mathrm{P}^{(LT_T)},\mathrm{P}^{(RT_T)}},$$

combined with Lemma 3.1 ($\mathrm{R}^p_{k\text{-tt}}(\mathrm{NP}) = \mathrm{R}^p_{1\text{-tt}}(\mathrm{BH}_k)$) and Corollary 4.2. □

Finally, we mention that a study of query order in the polynomial hierarchy (as opposed to the boolean hierarchy) has very recently been initiated by E. Hemaspaandra, L. Hemaspaandra, and H. Hempel ([19]; see also [31, 4]) and this study has led to a somewhat surprising downward translation result: for $k > 2$, $\Sigma^p_k = \Pi^p_k \iff \mathrm{P}^{\Sigma^p_k[1]} = \mathrm{P}^{\Sigma^p_k[2]}$ ([16]; see also the extensions obtained in [8, 20]). Query order (see also the survey [17]) has also recently proven useful in studying the structure of complete sets [18] and in characterizing bottleneck-computation classes [23].

Acknowledgments. We thank Edith Hemaspaandra, Johannes Köbler, and Jörg Vogel for helpful conversations. We thank Johannes Köbler for providing an advance copy of [15]. We are extremely indebted to Jörg Rothe for his generous help. The insights of the paragraph after the proof of Theorem 3.6 are due to him and appear here with his kind permission. He also made countless invaluable suggestions throughout this project and proofread an earlier version of this paper. We also are deeply grateful to editor Ker-I Ko and two anonymous referees for their invaluable suggestions regarding the organization of the paper and a very nice proof simplification for section 4.

REFERENCES

[1] M. AGRAWAL, R. BEIGEL, AND T. THIERAUF, *Modulo Information from Nonadaptive Queries to NP*, Tech. Report 96-001, Electronic Colloquium on Computational Complexity, Trier, Germany, Jan. 1996.

[2] J. BALCÁZAR, J. DÍAZ, AND J. GABARRÓ, *Structural Complexity* I, 2nd ed., Springer-Verlag, Berlin, New York, 1995.

[3] R. BEIGEL, *Bounded queries to SAT and the boolean hierarchy*, Theoret. Comput. Sci., 84 (1991), pp. 199–223.

[4] R. BEIGEL AND R. CHANG, *Commutative queries*, in Proc. 5th Israeli Symposium on Theory of Computing and Systems, IEEE Computer Society Press, Piscataway, NJ, June 1997, pp. 159–165.

[5] R. BEIGEL, R. CHANG, AND M. OGIWARA, *A relationship between difference hierarchies and relativized polynomial hierarchies*, Math. Systems Theory, 26 (1993), pp. 293–310.

[6] D. Bovet and P. Crescenzi, *Introduction to the Theory of Complexity*, Prentice–Hall, Englewood Cliffs, NJ, 1993.
[7] D. Bruschi, D. Joseph, and P. Young, *Strong separations for the boolean hierarchy over RP*, Internat. J. Found. Comput. Sci., 1 (1990), pp. 201–218.
[8] H. Buhrman and L. Fortnow, *Two Queries*, Tech. Report 96-20, Department of Computer Science, University of Chicago, Chicago, IL, Sept. 1996.
[9] J. Cai, *Probability one separation of the boolean hierarchy*, Lecture Notes in Comput. Sci. 247, Springer-Verlag, New York, 1987, pp. 148–158.
[10] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung, *The boolean hierarchy* I*: Structural properties*, SIAM J. Comput., 17 (1988), pp. 1232–1252.
[11] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung, *The boolean hierarchy* II*: Applications*, SIAM J. Comput., 18 (1989), pp. 95–111.
[12] J. Cai and L. Hemachandra, *The boolean hierarchy: Hardware over NP*, Lecture Notes in Comput. Sci. 223, Springer-Verlag, New York, June 1986, pp. 105–124.
[13] R. Chang, *On the Structure of NP Computations under Boolean Operators*, Ph.D. thesis, Cornell University, Ithaca, NY, 1991.
[14] R. Chang and J. Kadin, *The boolean hierarchy and the polynomial hierarchy: A closer connection*, SIAM J. Comput., 25 (1996), pp. 340–354.
[15] F. Green, J. Köbler, K. Regan, T. Schwentick, and J. Torán, *The power of the middle bit of a #P function*, J. Comput. Systems Sci., 50 (1995), pp. 456–467.
[16] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel, *A downward collapse within the polynomial hierarchy*, SIAM J. Comput., to appear.
[17] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel, *An introduction to query order*, Bulletin of the EATCS., 63 (1997), pp. 93–107.
[18] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel, $R^{\mathcal{SN}}_{1\text{-}tt}$*(NP) distinguishes robust many-one and Turing completeness*, Theory Comput. Systems, to appear.
[19] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel, *Query order in the polynomial hierarchy*, Lecture Notes in Comput. Sci. 1279, Springer-Verlag, New York, Sept. 1997.
[20] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel, *Translating Equality Downwards*, Tech. Report TR-657, Department of Computer Science, University of Rochester, Rochester, NY, Apr. 1997. (See also the strengthening of this in TR-681.)
[21] L. Hemaspaandra, H. Hempel, and G. Wechsung, *Query Order and Self-Specifying Machines*, Tech. Report TR-596, Department of Computer Science, University of Rochester, Rochester, NY, Oct. 1995.
[22] L. Hemaspaandra and J. Rothe, *Unambiguous computation: Boolean hierarchies and sparse Turing-complete sets*, SIAM J. Comput., 26 (1997), pp. 634–653.
[23] U. Hertrampf, *Acceptance by transformation monoids (with an application to local self-reductions)*, in Proc. 12th Annual IEEE Conference on Computational Complexity, IEEE Computer Society Press, Piscataway, NJ, June 1997, pp. 213–224.
[24] J. Kadin, *The polynomial time hierarchy collapses if the boolean hierarchy collapses*, SIAM J. Comput., 17 (1988), pp. 1263–1282; Erratum SIAM J. Comput., 20 (1991), p. 404.
[25] J. Köbler, U. Schöning, and K. Wagner, *The difference and truth-table hierarchies for NP*, RAIRO Theoret. Inform. Appl., 21 (1987), pp. 419–435.
[26] R. Ladner, N. Lynch, and A. Selman, *A comparison of polynomial time reducibilities*, Theoret. Comput. Sci., 1 (1975), pp. 103–124.
[27] C. Papadimitriou, *Computational Complexity*, Addison–Wesley, Reading, MA, 1994.
[28] K. Wagner, *Number-of-Query Hierarchies*, Tech. Report 158, Universität Augsburg, Institut für Mathematik, Augsburg, Germany, Oct. 1987.
[29] K. Wagner, *Number-of-Query Hierarchies*, Tech. Report 4, Institut für Informatik, Universität Würzburg, Würzburg, Germany, Feb. 1989.
[30] K. Wagner, *Bounded query classes*, SIAM J. Comput., 19 (1990), pp. 833–846.
[31] K. Wagner, *A Note on Parallel Queries and the Difference Hierarchy*, Tech. Report 173, Institut für Informatik, Universität Würzburg, Würzburg, Germany, June 1997.
[32] G. Wechsung, *On the boolean closure of NP*, Lecture Notes in Comput. Sci. 199, Springer-Verlag, New York, 1985, pp. 485–493. (An unpublished precursor of this paper was coauthored by K. Wagner.)
[33] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel, *What's Up with Downward Collapse: Using the Easy-Hard Technique to Link Boolean and Polynomial Hierarchy Collapses*, Department of Computer Science Technical Report TR-98-682, University of Rochester, 1998.
[34] S. Reith and K. Wagner, *On Boolean Lowness and Boolean Highness*, manuscript, 1998.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 652–673

FINDING THE k SHORTEST PATHS*

DAVID EPPSTEIN†

Abstract. We give algorithms for finding the k shortest paths (not required to be simple) connecting a pair of vertices in a digraph. Our algorithms output an implicit representation of these paths in a digraph with n vertices and m edges, in time $O(m + n \log n + k)$. We can also find the k shortest paths from a given source s to each vertex in the graph, in total time $O(m+n \log n+kn)$. We describe applications to dynamic programming problems including the knapsack problem, sequence alignment, maximum inscribed polygons, and genealogical relationship discovery.

Key words. shortest paths, network programming, path enumeration, near-optimal solutions, dynamic programming, knapsack problem, sequence alignment, inscribed polygon, genealogy

AMS subject classifications. 05C12, 05C85, 94C15

PII. S0097539795290477

1. Introduction.

We consider a long-studied generalization of the shortest path problem, in which not one but several short paths must be produced. The *k-shortest-paths problem* is to list the k paths connecting a given source-destination pair in the digraph with minimum total length. Our techniques also apply to the problem of listing all paths shorter than some given threshold length. In the version of these problems studied here, cycles of repeated vertices are allowed. We first present a basic version of our algorithm, which is simple enough to be suitable for practical implementation while losing only a logarithmic factor in time complexity. We then show how to achieve optimal time (constant time per path once a shortest path tree has been computed) by applying Frederickson's [26] algorithm for finding the minimum k elements in a *heap-ordered tree*.

1.1. Applications.

The applications of shortest path computations are too numerous to cite in detail. They include situations in which an actual path is the desired output, such as robot motion planning, highway and power line engineering, and network connection routing. They include problems of scheduling such as critical path computation in PERT charts. Many optimization problems solved by dynamic programming or more complicated matrix searching techniques, such as the knapsack problem, sequence alignment in molecular biology, construction of optimal inscribed polygons, and length-limited Huffman coding, can be expressed as shortest path problems.

Methods for finding k shortest paths have been applied to many of these applications, for several reasons.

- *Additional constraints.* One may wish to find a path that satisfies certain constraints beyond having a small length, but those other constraints may be ill defined or hard to optimize. For instance, in power transmission route selection [18], a power line should connect its endpoints reasonably directly, but there may be more or less community support for one option or another. A typical solution is to compute several short paths and then choose among

*Received by the editors August 18, 1995; accepted for publication (in revised form) April 17, 1997; published electronically August 4, 1998. This research was supported in part by NSF grant CCR-9258355 and by matching funds from Xerox Corporation.
http://www.siam.org/journals/sicomp/28-2/29047.html
†Department of Information and Computer Science, University of California, Irvine, CA 92697-3425 (eppstein@ics.uci.edu, http://www.ics.uci.edu/~eppstein).

them by considering the other criteria. We recently implemented a similar technique as a heuristic for the NP-hard problem of, given a graph with colored edges, finding a shortest path using each color at most once [20]. This type of application is the main motivation cited by Dreyfus [17] and Lawler [39] for k-shortest-path computations.

- *Model evaluation.* Paths may be used to model problems that have known solutions, independent of the path formulation; for instance, in a k-shortest-path model of automatic translation between natural languages [30], a correct translation can be found by a human expert. By listing paths until this known solution appears, one can determine how well the model fits the problem, in terms of the number of incorrect paths seen before the correct path. This information can be used to tune the model as well as to determine the number of paths that need to be generated when applying additional constraints to search for the correct solution.
- *Sensitivity analysis.* By computing more than one shortest path, one can determine how sensitive the optimal solution is to variation of the problem's parameters. In biological sequence alignment, for example, one typically wishes to see several "good" alignments rather than one optimal alignment; by comparing these several alignments, biologists can determine which portions of an alignment are most essential [8, 64]. This problem can be reduced to finding several shortest paths in a grid graph.
- *Generation of alternatives.* It may be useful to examine not just the optimal solution to a problem but a larger class of solutions, to gain a better understanding of the problem. For example, the states of a complex system might be represented as a finite state machine, essentially just a graph, with different probabilities on each state transition edge. In such a model, one would likely want to know not just the chain of events most likely to lead to a failure state but rather all chains having a failure probability over some threshhold. Taking the logarithms of the transition probabilities transforms this problem into one of finding all paths shorter than a given length.

We later discuss in more detail some of the dynamic programming applications listed above and show how to find the k best solutions to these problems by using our shortest path algorithms. As well as improving previous solutions to the general k-shortest-paths problem, our results improve more specialized algorithms for finding length-bounded paths in the grid graphs arising in sequence alignment [8] and for finding the k best solutions to the knapsack problem [15].

1.2. New results. We prove the following results. In all cases we assume we are given a digraph in which each edge has a nonnegative length. We allow the digraph to contain self-loops and multiple edges. In each case the paths are output in an implicit representation from which simple properties such as the length are available in constant time per path. We may explicitly list the edges in any path in time proportional to the number of edges.

- We find the k shortest paths (allowing cycles) connecting a given pair of vertices in a digraph in time $O(m + n \log n + k)$.
- We find the k shortest paths from a given source in a digraph to each other vertex in time $O(m + n \log n + kn)$.

We can also solve the similar problem of finding all paths shorter than a given length, with the same time bounds. The same techniques apply to digraphs with negative edge lengths but no negative cycles, but the time bounds above should be

modified to include the time to compute a single source shortest path tree in such networks, $O(mn)$ [6, 23] or $O(mn^{1/2} \log N)$ where all edge lengths are integers and N is the absolute value of the most negative edge length [29]. For a directed acyclic graph (DAG), with or without negative edge lengths, shortest path trees can be constructed in linear time and the $O(n \log n)$ term above can be omitted. The related problem of finding the k longest paths in a DAG [4] can be transformed to a shortest path problem simply by negating all edge lengths; we can therefore also solve it in the same time bounds.

1.3. Related work. Many papers study algorithms for k shortest paths [3, 5, 7, 9, 13, 14, 17, 24, 31, 32, 34, 35, 37, 38, 39, 40, 41, 43, 44, 45, 47, 50, 51, 56, 57, 58, 59, 60, 63, 65, 66, 67, 68, 69]. Dreyfus [17] and Yen [69] cite several additional papers on the subject going back as far as 1957.

One must distinguish several common variations of the problem. In many of the papers cited above, the paths are restricted to be *simple*; i.e., no vertex can be repeated. This has advantages in some applications, but as our results show this restriction seems to make the problem significantly harder. Several papers [3, 13, 17, 24, 41, 42, 58, 59] consider the version of the k-shortest-paths problem in which repeated vertices are allowed, and it is this version that we also study. Of course, for the DAGs that arise in many of the applications described above, including scheduling and dynamic programming, no path can have a repeated vertex and the two versions of the problem become equivalent. Note also that in the application described earlier of listing the most likely failure paths of a system modelled by a finite state machine, it is the version studied here rather than the more common simple path version that one wants to solve.

One can also make a restriction that the paths found be edge disjoint or vertex disjoint [61] or include capacities on the edges [10, 11, 12, 49]; however, such changes turn the problem into one more closely related to network flow.

Fox [24] gives a method for the k-shortest-path problem based on Dijkstra's algorithm, which with more recent improvements in priority queue data structures [27] takes time $O(m + kn \log n)$; this seems to be the best previously known k-shortest-paths algorithm. Dreyfus [17] mentions the version of the problem in which we must find paths from one source to each other vertex in the graph, and describes a simple $O(kn^2)$ time dynamic programming solution to this problem. For the k-shortest-simple-paths problem, the best known bound is $O(k(m + n \log n))$ in undirected graphs [35] or $O(kn(m + n \log n))$ in directed graphs ([39], again including more recent improvements in Dijkstra's algorithm). Thus all previous algorithms took time $O(n \log n)$ or more per path. We improve this to constant time per path.

A similar problem to the one studied here is that of finding the k minimum weight spanning trees in a graph. Recent algorithms for this problem [22, 21, 25] reduce it to finding the k minimum weight nodes in a *heap-ordered tree*, defined using the *best swap* in a sequence of graphs. Heap-ordered tree selection has also been used to find the smallest interpoint distances or the nearest neighbors in geometric point sets [16]. We apply a similar tree selection technique to the k-shortest-path problem; however, the reduction of k shortest paths to heap-ordered trees is very different from the constructions in these other problems.

2. The basic algorithm. Finding the k shortest paths between two terminals s and t has been a difficult enough problem to warrant much research. In contrast, the similar problem of finding paths with only one terminal s, ending anywhere in the graph, is much easier: one can simply use breadth first search. Maintain a priority

queue of paths, initially containing the single zero-edge path from s to itself; then repeatedly remove the shortest path from the priority queue, add it to the list of output paths, and add all one-edge extensions of that path to the priority queue. If the graph has bounded degree d, a breadth first search from s until k paths are found takes time $O(dk + k \log k)$; note that this bound does not depend in any way on the overall size of the graph. If the paths need not be output in order by length, Frederickson's heap selection algorithm [26] can be used to speed this up to $O(dk)$.

The main idea of our k-shortest-paths algorithm, then, is to translate the problem from one with two terminals, s and t, to a problem with only one terminal. One can find paths from s to t simply by finding paths from s to any other vertex and concatenating a shortest path from that vertex to t. However, we cannot simply apply this idea directly, for several reasons: (1) There is no obvious relation between the ordering of the paths from s to other vertices and the ordering of the corresponding paths from s to t. (2) Each path from s to t may be represented in many ways as a path from s to some vertex followed by a shortest path from that vertex to t. (3) Our input graph may not have bounded degree.

In outline, we deal with problem (1) by using a potential function to modify the edge lengths in the graph so that the length of any shortest path to t is zero; therefore concatenating such paths to paths from s will preserve the ordering of the path lengths. We deal with problem (2) by considering only paths from s in which the last edge is not in a fixed shortest path tree to t; this leads to the implicit representation we use to represent each path in constant space. (Ideas similar to these appear also in [46].) However, this solution gives rise to a fourth problem: (4) We do not wish to spend much time searching edges of the shortest path tree, as this time can not be charged against newly found s-t paths.

The heart of our algorithm is the solution to problems (3) and (4). Our idea is to construct a binary heap for each vertex, listing the edges that are not part of the shortest path tree and that can be reached from that vertex by shortest-path-tree edges. In order to save time and space, we use persistence techniques to allow these heaps to share common structures with each other. In the basic version of the algorithm, this collection of heaps forms a bounded-degree graph having $O(m+n \log n)$ vertices. Later we show how to improve the time and space bounds of this part of the algorithm using tree decomposition techniques of Frederickson [25].

2.1. Preliminaries. We assume throughout that our input graph G has n vertices and m edges. We allow self-loops and multiple edges, so m may be larger than $\binom{n}{2}$. The *length* of an edge e is denoted $\ell(e)$. By extension we can define the length $\ell(p)$ for any path in G to be the sum of its edge lengths. The *distance* $d(s,t)$ for a given pair of vertices is the length of the shortest path starting at s and ending at t; with the assumption of no negative cycles, this is well defined. Note that $d(s,t)$ may be unequal to $d(t,s)$. The two endpoints of a directed edge e are denoted *tail*(e) and *head*(e); the edge is directed from *tail*(e) to *head*(e).

For our purposes, a *heap* is a binary tree in which vertices have weights, satisfying the restriction that the weight of any vertex is less than or equal to the minimum weight of its children. We will not always care whether the tree is balanced (and in some circumstances we will allow trees with infinite depth). More generally, a *D-heap* is a degree-D tree with the same weight-ordering property; thus the usual heaps above are 2-heaps. As is well known (e.g., see [62]), any set of values can be placed into a balanced heap by the *heapify* operation in linear time. In a balanced heap, any new element can be inserted in logarithmic time. We can list the elements of a heap in

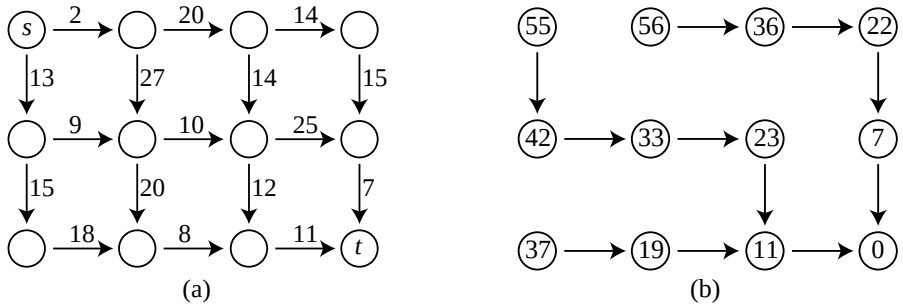


FIG. 1. (a) *Example digraph G with edge lengths and specified terminals;* (b) *shortest path tree T and distances to t in G.*

order by weight, taking logarithmic time to generate each element, simply by using breadth first search.

2.2. Implicit representation of paths. As discussed earlier, our algorithm does not output each path it finds explicitly as a sequence of edges; instead it uses an implicit representation, described in this section.

The ith shortest path in a digraph may have $\Omega(ni)$ edges, so the best time we could hope for in an explicit listing of shortest paths would be $O(k^2n)$. Our time bounds are faster than this, so we must use an implicit representation for the paths. However, our representation is not a serious obstacle to use of our algorithm: we can list the edges of any path we output in time proportional to the number of edges, and simple properties (such as the length) are available in constant time. Similar implicit representations have previously been used for related problems such as the k minimum weight spanning trees [22, 21, 25]. Further, previous papers on the k-shortest-path problem give time bounds omitting the $O(k^2n)$ term above, so these papers must tacitly or not be using an implicit representation.

Our representation is similar in spirit to those used for the k minimum weight spanning trees problem: for that problem, each successive tree differs from a previously listed tree by a *swap*, the insertion of one edge and removal of another edge. The implicit representation consists of a pointer to the previous tree and a description of the swap. For the shortest path problem, each successive path will turn out to differ from a previously listed path by the inclusion of a single edge not part of a shortest path tree and appropriate adjustments in the portion of the path that involves shortest path tree edges. Our implicit representation consists of a pointer to the previous path, and a description of the newly added edge.

Given s and t in a digraph G (Figure 1(a)), let T be a single-destination shortest path tree with t as destination (Figure 1(b); this is the same as a single source shortest path tree in the graph G^R formed by reversing each edge of G). We can compute T in time $O(m + n \log n)$ [27]. We denote by $next_T(v)$ the next vertex reached after v on the path from v to t in T.

Given an edge e in G, define

$$\delta(e) = \ell(e) + d(head(e), t) - d(tail(e), t).$$

Intuitively, $\delta(e)$ measures how much distance is lost by being "sidetracked" along e instead of taking a shortest path to t. The values of δ for our example graph are shown in Figure 2(a).

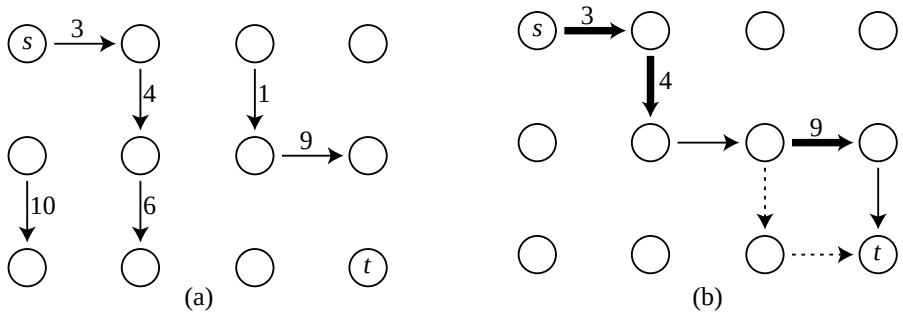


FIG. 2. (a) *Edges in $G-T$ labeled by $\delta(e)$ ($\delta(e)=0$ for edges in T);* (b) *path p, sidetracks(p) (the heavy edges, labeled* 3, 4, *and* 9*), and prefpath(p) (differing from p in the two dashed edges; sidetracks(prefpath(p)) consists of the two edges labeled* 3 *and* 4*).*

LEMMA 1. *For any $e \in G$, $\delta(e) \geq 0$. For any $e \in T$, $\delta(e) = 0$.*

For any path p in G, formed by a sequence of edges, some edges of p may be in T, and some others may be in $G-T$. Any path p from s to t is uniquely determined solely by the subsequence *sidetracks*(p) of its edges in $G-T$ (Figure 2(b)). For, given a pair of edges in the subsequence, there is a uniquely determined way of inserting edges from T so that the head of the first edge is connected to the tail of the second edge. As an example, the shortest path in T from s to t is represented by the empty sequence. A sequence of edges in $G-T$ may not correspond to any s-t path, if it includes a pair of edges that cannot be connected by a path in T. If $S = \mathit{sidetracks}(p)$, we define *path*($S$) to be the path p.

Our implicit representation will involve these sequences of edges in $G-T$. We next show how to recover $\ell(p)$ from information in *sidetracks*(p).

For any nonempty sequence S of edges in $G-T$, let *prefix*(S) be the sequence formed by the removal of the last edge in S. If $S = \mathit{sidetracks}(p)$, then we denote this last sidetrack edge by *lastsidetrack*(p); *prefix*(S) will define a path *prefpath*(p) = *path*(*prefix*(S)) (Figure 2(b)).

LEMMA 2. *For any path p from s to t,*

$$\ell(p) = d(s,t) + \sum_{e \in \mathit{sidetracks}(p)} \delta(e) = d(s,t) + \sum_{e \in p} \delta(e).$$

LEMMA 3. *For any path p from s to t in G, for which sidetracks(p) is nonempty, $\ell(p) \geq \ell(\mathit{prefpath}(p))$.*

Our representation of a path p in the list of paths produced by our algorithm will then consist of two components:

- the position in the list of *prefpath*(p).
- edge *lastsidetrack*(p).

Although the final version of our algorithm, which uses Frederickson's heap selection technique, does not necessarily output paths in sorted order, we will nevertheless be able to guarantee that *prefpath*(p) is output before p. One can easily recover p itself from our representation in time proportional to the number of edges in p. The length $\ell(p)$ for each path can easily be computed as $\delta(\mathit{lastsidetrack}(p)) + \ell(\mathit{prefpath}(p))$. We will see later that we can also compute many other simple properties of the paths in constant time per path.

2.3. Representing paths by a heap. The representation of s-t paths discussed in the previous section gives a natural tree of paths, in which the parent of any path p

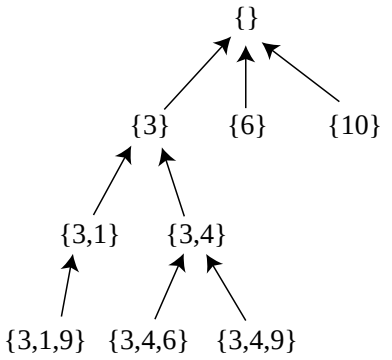


FIG. 3. *Tree of paths, labeled by sidetracks(p).*

is *prefpath*(*p*) (Figure 3). The degree of any node in this *path tree* is at most m, since there can be at most one child for each possible value of *lastsidetrack*(*p*). The possible values of *lastsidetrack*(*q*) for paths q that are children of p are exactly those edges in $G-T$ that have tails on the path from *head*(*lastsidetrack*(*p*)) to t in the shortest path tree T.

If G contains cycles, the path tree is infinite. By Lemma 3, the path tree is heap-ordered. However, since its degree is not necessarily constant, we cannot directly apply breadth first search (nor Frederickson's heap selection technique, described later in Lemma 8) to find its k minimum values. Instead we form a heap by replacing each node p of the path tree with an equivalent bounded-degree subtree (essentially, a heap of the edges with tails on the path from *head*(*lastsidetrack*(*p*)) to t, ordered by $\delta(e)$). We must also take care that we do this in such a way that the portion of the path tree explored by our algorithm can be easily constructed.

For each vertex v we wish to form a heap $H_G(v)$ for all edges with tails on the path from v to t, ordered by $\delta(e)$. We will later use this heap to modify the path tree by replacing each node p with a copy of $H_G(\mathit{head}(\mathit{lastsidetrack}(p)))$.

Let *out*(v) denote the edges in $G-T$ with tails at v (Figure 4(a)). We first build a heap $H_{out}(v)$ for each vertex v of the edges in *out*(v) (Figure 4(b)). The weights used for the heap are simply the values $\delta(e)$ defined earlier. $H_{out}(v)$ will be a 2-heap with the added restriction that the root of the heap only has one child. It can be built for each v in time $O(|\mathit{out}(v)|)$ by letting the root *outroot*(v) be the edge minimizing $\delta(e)$ in *out*(v) and letting its child be a heap formed by heapification of the rest of the edges in *out*(v). The total time for this process is $\sum O(|\mathit{out}(v)|) = O(m)$.

We next form the heap $H_G(v)$ by merging all heaps $H_{out}(w)$ for w on the path in T from v to t. More specifically, for each vertex v we merge $H_{out}(v)$ into $H_G(\mathit{next}_T(v))$ to form $H_G(v)$. We will continue to need $H_G(\mathit{next}_T(v))$, so this merger should be done in a persistent (nondestructive) fashion.

We guide this merger of heaps using a balanced heap $H_T(v)$ for each vertex v, containing only the roots *outroot*(w) of the heaps $H_{out}(w)$, for each w on the path from v to t. $H_T(v)$ is formed by inserting *outroot*(v) into $H_T(\mathit{next}_T(v))$ (Figure 5(a)). To perform this insertion persistently, we create new copies of the nodes on the path updated by the insertion (marked by asterisks in Figure 5(a)), with appropriate pointers to the other, unchanged members of $H_T(\mathit{next}_T(v))$. Thus we can store $H_T(v)$ without changing $H_T(\mathit{next}_T(v))$ by using an additional $O(\log n)$ words of memory to store only the nodes on that path.

We now form $H_G(v)$ by connecting each node *outroot*(w) in $H_T(v)$ to an additional

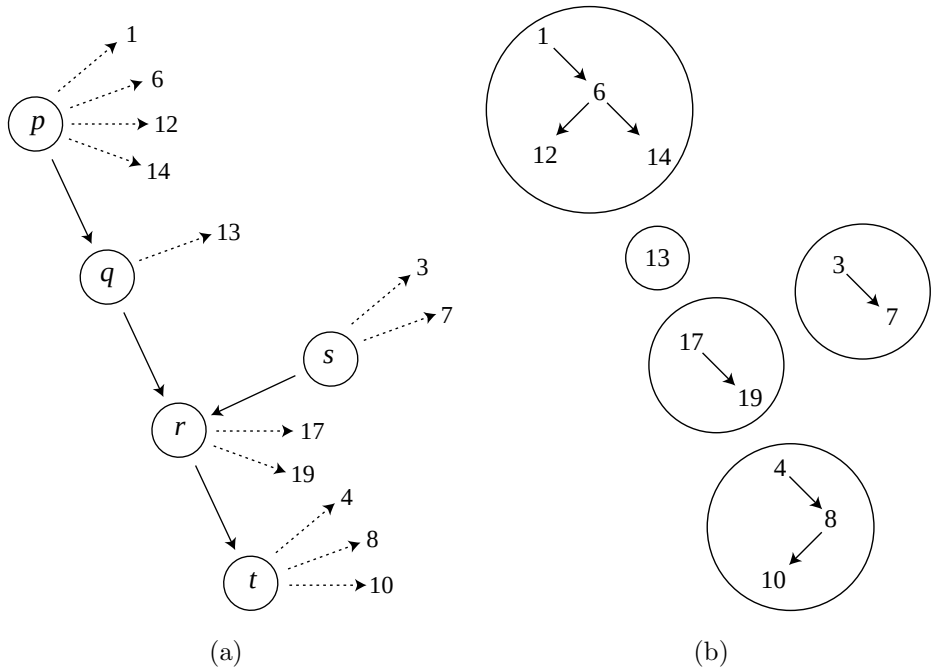


FIG. 4. (a) *Portion of a shortest path tree, showing out*(v) *and corresponding values of* δ*;* (b) $H_{out}(v)$.

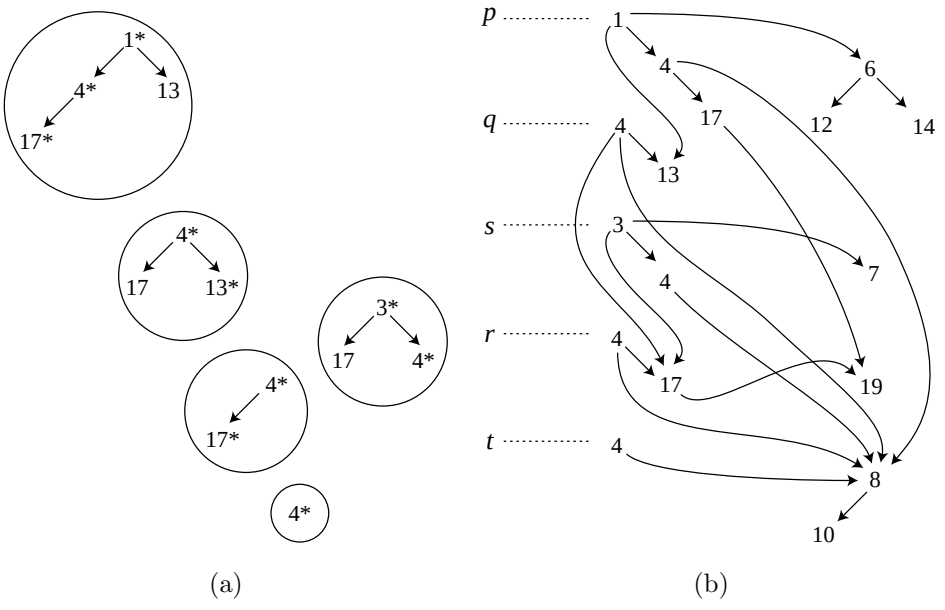


FIG. 5. (a) $H_T(v)$ *with asterisks marking path of nodes updated by insertion of outroot*(v) *into* $H_T(next_T(v))$*;* (b) $D(G)$ *has a node for each marked node in Figure* 5(a) *and each nonroot node in Figure* 4(b).

subtree beyond the two it points to in $H_T(v)$, namely, to the rest of heap $H_{out}(w)$.

$H_G(v)$ can be constructed at the same time as we construct $H_T(v)$, with a similar amount of work. $H_G(v)$ is thus a 3-heap, as each node includes at most three children, either two from $H_T(v)$ and one from $H_{out}(w)$ or none from $H_T(v)$ and two from $H_{out}(w)$.

We summarize the construction so far in a form that emphasizes the shared structure in the various heaps $H_G(v)$.

LEMMA 4. *In time $O(m + n \log n)$ we can construct a DAG $D(G)$ and a map from vertices $v \in G$ to $h(v) \in D(G)$, with the following properties:*

- *$D(G)$ has $O(m + n \log n)$ vertices.*
- *Each vertex in $D(G)$ corresponds to an edge in $G - T$.*
- *Each vertex in $D(G)$ has out-degree at most 3.*
- *The vertices reachable in $D(G)$ from $h(v)$ form a 3-heap $H_G(v)$ in which the vertices of the heap correspond to edges of $G - T$ with tails on the path in T from v to t, in heap order by the values of $\delta(e)$.*

Proof. The vertices in $D(G)$ come from two sources: heaps $H_{out}(v)$ and $H_T(v)$. Each node in $H_{out}(v)$ corresponds to a unique edge in $G - T$, so there are at most $m - n + 1$ nodes coming from heaps $H_{out}(v)$. Each vertex of G also contributes $\lfloor \log_2 i \rfloor$ nodes from heaps $H_T(v)$, where i is the length of the path from the vertex to t, $1 + \lfloor \log_2 i \rfloor$ measures the number of balanced binary heap nodes that need to be updated when inserting $outroot(v)$ into $H_T(next_T(v))$, and we subtract one because $outroot(v)$ itself was already included in our total for $H_{out}(v)$. In the worst case, T is a path and the total contribution is at most $\sum_i \lfloor \log_2 i \rfloor \le n \log_2 n - cn$, where c varies between roughly 1.91 and 2 depending on the ratio of n to the nearest power of two. Therefore the total number of nodes in $D(G)$ is at most $m + n \log_2 n - (c+1)n$. The degree bound follows from the construction, and it is straightforward to construct $D(G)$ as described above in constant time per node, after computing the shortest path tree T in time $O(m + n \log n)$ using Fibonacci heaps [27].

Map $h(v)$ simply takes v to the root of $H_G(v)$. For any vertex v in $D(G)$, let $\delta(v)$ be a shorthand for $\delta(e)$, where e is the edge in G corresponding to v. By construction, the nodes reachable from $h(v)$ are those in $H_T(v)$ together with, for each such node w, the rest of the nodes in $H_{out}(w)$; $H_T(v)$ was constructed to correspond exactly to the vertices on the path from v to t, and $H_{out}(w)$ represents the edges with tails at each vertex, so together these reachable nodes represent all edges with tails on the path. Each edge (u, v) in $D(G)$ corresponds to an edge either in some $H_T(w)$ or in some $H_{out}(w)$, and in either case the heap ordering for $D(G)$ is a consequence of the ordering in these smaller heaps. □

$D(G)$ is shown in Figure 5(b). The nodes reachable from s in $D(G)$ form a structure $H_G(s)$ representing the paths differing from the original shortest path by the addition of a single edge in $G - T$. We now describe how to augment $D(G)$ with additional edges to produce a graph which can represent all s-t paths, not just those paths with a single edge in $G - T$.

We define the *path graph* $P(G)$ as follows. The vertices of $P(G)$ are those of $D(G)$, with one additional vertex, the *root* $r = r(s)$. The vertices of $P(G)$ are unweighted, but the edges are given lengths. For each directed edge (u, v) in $D(G)$, we create the edge between the corresponding vertices in $P(G)$, with length $\delta(v) - \delta(u)$. We call such edges *heap edges*. For each vertex v in $P(G)$, corresponding to an edge in $G - T$ connecting some pair of vertices u and w, we create a new edge from v to $h(w)$ in $P(G)$, having as its length $\delta(h(w))$. We call such edges *cross edges*. We also create

an *initial edge* between r and $h(s)$, having as its length $\delta(h(s))$.

$P(G)$ has $O(m + n \log n)$ vertices, each with out-degree at most four. It can be constructed in time $O(m + n \log n)$.

LEMMA 5. *There is a one-to-one length-preserving correspondence between s-t paths in G, and paths starting from r in $P(G)$.*

Proof. Recall that an s-t path p in G is uniquely defined by *sidetracks*(p), the sequence of edges from p in $G - T$. We now show that for any such sequence, there corresponds a unique path from r in $P(G)$ ending at a node corresponding to *lastsidetrack*(p), and conversely any path from r in $P(G)$ corresponds to *sidetracks*(p) for some path p.

Given a path p in G, we construct a corresponding path p' in $P(G)$ as follows. If *sidetracks*(p) is empty (i.e., p is the shortest path), we let p' consist of the single node r. Otherwise, form a path q' in $P(G)$ corresponding to *prefpath*(p) by induction on the length of *sidetracks*(p). By induction, q' ends at a node of $P(G)$ corresponding to edge $(u, v) =$ *lastsidetrack*(*prefpath*(p)). When we formed $P(G)$ from $D(G)$, we added an edge from this node to $h(v)$. Since *lastsidetrack*(p) has its tail on the path in T from v to t, it corresponds to a unique node in $H_G(v)$, and we form p' by concatenating q' with the path from $h(v)$ to that node. The edge lengths on this concatenated path telescope to $\delta($*lastsidetrack*$(p))$, and $\ell(p) = \ell($*prefpath*$(p)) + \ell($*lastsidetrack*$(p))$ by Lemma 2, so by induction $\ell(p) = \ell(q') + \ell($*lastsidetrack*$(p)) = \ell(p')$.

Conversely, to construct an s-t path in G from a path p' in $P(G)$, we construct a sequence of edges in G, *pathseq*(p'). If p' is empty, *pathseq*(p') is also empty. Otherwise *pathseq*(p') is formed by taking in sequence the edges in G corresponding to tails of cross edges in p' and adding at the end of the sequence the edge in G corresponding to the final vertex of p'. Since the nodes of $P(G)$ reachable from the head of each cross edge (u, v) are exactly those in $H_G(v)$, each successive edge added to *pathseq*(p') is on the path in T from v to t, and *pathseq*(p') is of the form *sidetracks*(p) for some path p in G. □

LEMMA 6. *In $O(m + n \log n)$ time we can construct a graph $P(G)$ with a distinguished vertex r, having the following properties.*

- *$P(G)$ has $O(m + n \log n)$ vertices.*
- *Each vertex of $P(G)$ has out-degree at most four.*
- *Each edge of $P(G)$ has nonnegative weight.*
- *There is a one-to-one correspondence between s-t paths in G and paths starting from r in $P(G)$.*
- *The correspondence preserves lengths of paths in that length ℓ in $P(G)$ corresponds to length $d(s, t) + \ell$ in G.*

Proof. The bounds on size, time, and out-degree follow from Lemma 4, and the nonnegativity of edge weights follows from the heap ordering proven in that lemma. The correctness of the correspondence between paths in G and in $P(G)$ is shown above in Lemma 5. □

To complete our construction, we find from the path graph $P(G)$ a 4-heap $H(G)$, so that the nodes in $H(G)$ represent paths in G. $H(G)$ is constructed by forming a node for each path in $P(G)$ rooted at r. The parent of a node is the path with one fewer edge. Since $P(G)$ has out-degree four, each node has at most four children. Weights are heap-ordered, and the weight of a node is the length of the corresponding path.

LEMMA 7. *$H(G)$ is a 4-heap in which there is a one-to-one correspondence between nodes and s-t paths in G, and in which the length of a path in G is $d(s, t)$*

plus the weight of the corresponding node in $H(G)$.

We note that, if an algorithm explores a connected region of $O(k)$ nodes in $H(G)$, it can represent the nodes in constant space by assigning them numbers and indicating for each node its parent and the additional edge in the corresponding path of $P(G)$. The children of a node are easy to find simply by following appropriate out-edges in $P(G)$, and the weight of a node is easy to compute from the weight of its parent. It is also easy to maintain along with this representation the corresponding implicit representation of s-t paths in G.

2.4. Finding the k shortest paths.

THEOREM 1. *In time $O(m + n \log n)$ we can construct a data structure that will output the shortest paths from s to t in a graph in order by weight, taking time $O(\log i)$ to output the ith path.*

Proof. We apply breadth first search to $P(G)$, as described at the start of the section, and translate the search results to paths using the correspondence described above. □

We next describe how to compute paths from s to all n vertices of the graph. In fact our construction solves more easily the reverse problem of finding paths from each vertex to the destination t. The construction of $P(G)$ is as above, except that instead of adding a single root $r(s)$ connected to $h(s)$, we add a root $r(v)$ for each vertex $v \in G$. The modification to $P(G)$ takes $O(n)$ time. Using the modified $P(G)$, we can compute a heap $H_v(G)$ of paths from each v to t and compute the k smallest such paths in time $O(k)$.

THEOREM 2. *Given a source vertex s in a digraph G, we can find in time $O(m + n \log n + kn \log k)$ an implicit representation of the k shortest paths from s to each other vertex in G.*

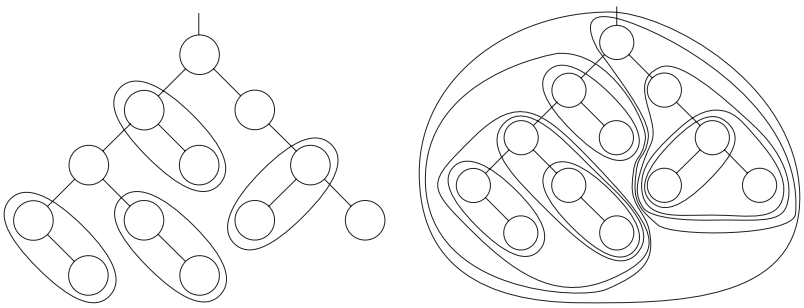
Proof. We apply the construction above to G^R, with s as destination. We form the modified path graph $P(G^R)$, find for each vertex v a heap $H_v(G^R)$ of paths in G^R from v to s, and apply breadth first search to this heap. Each resulting path corresponds to a path from s to v in G. □

3. Improved space and time. The basic algorithm described above takes time $O(m + n \log n + k \log k)$, even if a shortest path tree has been given. If the graph is sparse, the $n \log n$ term makes this bound nonlinear. This term comes from two parts of our method, Dijkstra's shortest path algorithm and the construction of $P(G)$ from the tree of shortest paths. But for certain graphs, or with certain assumptions about edge lengths, shortest paths can be computed more quickly than $O(m+n \log n)$ [2, 28, 33, 36], and in these cases we would like to speed up our construction of $P(G)$ to match these improvements. In other cases, k may be large and the $k \log k$ term may dominate the time bound; again we would like to improve this nonlinear term. In this section we show how to reduce the time for our algorithm to $O(m + n + k)$, assuming a shortest path tree is given in the input. As a consequence we can also improve the space used by our algorithm.

3.1. Faster heap selection. The following result is due to Frederickson [26].

LEMMA 8. *We can find the k smallest weight vertices in any heap in time $O(k)$.*

Frederickson's result applies directly to 2-heaps, but we can easily extend it to D-heaps for any constant D. One simple method of doing this involves forming a 2-heap from the given D-heap by making $D - 1$ copies of each vertex, connected in a binary tree with the D children as leaves, and breaking ties in such a way that the Dk smallest weight vertices in the 2-heap correspond exactly to the k smallest weights in



(a) (b)

FIG. 6. (a) *Restricted partition of order* 2*;* (b) *multilevel partition.*

the D-heap.

By using this algorithm in place of breadth first search, we can reduce the $O(k \log k)$ term in our time bounds to $O(k)$.

3.2. Faster path heap construction. Recall that the bottleneck of our algorithm is the construction of $H_T(v)$, a heap for each vertex v in G of those vertices on the path from v to t in the shortest path tree T. The vertices in $H_T(v)$ are in heap order by $\delta(\mathit{outroot}(u))$. In this section we consider the abstract problem, given a tree T with weighted nodes, of constructing a heap $H_T(v)$ for each vertex v of the other nodes on the path from v to the root of the tree. The construction of Lemma 4 solves this problem in time and space $O(n \log n)$; here we give a more efficient but also more complicated solution.

By introducing dummy nodes with large weights, we can assume without loss of generality that T is binary and that the root t of T has indegree one. We will also assume that all vertex weights in T are distinct; this can be achieved at no loss in asymptotic complexity by use of a suitable tie-breaking rule. We use the following technique of Frederickson [25].

DEFINITION 1. *A restricted partition of order z with respect to a rooted binary tree T is a partition of the vertices of V such that*

1. *each set in the partition contains at most z vertices;*
2. *each set in the partition induces a connected subtree of T;*
3. *for each set S in the partition, if S contains more than one vertex, then there are at most two tree edges having one endpoint in S;*
4. *no two sets can be combined and still satisfy the other conditions.*

In general such a partition can easily be found in linear time by merging sets until we get stuck. However for our application, z will always be 2 (Figure 6(a)), and by working from the bottom up we can find an optimal partition in linear time.

LEMMA 9 (Frederickson [25]). *In linear time we can find an order-*2 *partition of a binary tree T for which there are at most* $5n/6$ *sets in the partition.*

Contracting each set in a restricted partition gives again a binary tree. We form a *multilevel* partition [25] by recursively partitioning this contracted binary tree (Figure 6(b)). We define a sequence of trees T_i as follows. Let $T_0 = T$. For any $i > 0$, let T_i be formed from T_{i-1} by performing a restricted partition as above and contracting the resulting sets. Then $|T_i| = O((5/6)^i n)$.

For any set S of vertices in T_{i-1} contracted to form a vertex v in T_i, define $\mathit{nextlevel}(S)$ to be the set in the partition of T_i containing S. We say that S is an

interior set if it is contracted to a degree two vertex. Note that if t has indegree one, the same is true for the root of any T_i, so t is not part of any interior set, and each interior set has one incoming and one outgoing edge. Since T_i is a contraction of T, each edge in T_i corresponds to an edge in T. Let e be the outgoing edge from v in T_i; then we define *rootpath*(S) to be the path in T from *head*(e) to t. If S is an interior set, with a single incoming edge e', we let *inpath*(S) be the path in T from *head*(e') to *tail*(e).

Define an *m-partial heap* to be a pair (M, H), where H is a heap and M is a set of m elements each smaller than all nodes in H. If H is empty, M can have fewer than m elements and we will still call (M, H) an m-partial heap.

Let us outline the structures used in our algorithm, before describing the details of computing these structures. We first find a partial heap $(M_1(S), H_1(S))$ for the vertices of T in each path *inpath*(S). Although our algorithm performs an interleaved construction of all of these sets at once, it is easiest to define them from the top down by defining $M_1(S)$ for a set S in the partition of T_{i-1} in terms of similar sets in T_i and higher levels of the multilevel partition. Specifically, let $M_2(S)$ denote those elements in $M_1(S')$ for those S' containing S at higher levels of the multilevel partition, and let $k = \max(i+2, |M_2(S)|+1)$; then we define $M_1(S)$ to be the vertices in *inpath*(S) having the k smallest vertex weights. Our algorithm for computing $H_1(S)$ from the remaining vertices on *inpath*(S) involves an intermediate heap $H_2(S')$ formed by adding the vertices in $M_1(S') - M_1(S)$ to $H_1(S')$, where S' consists of one or both of the subsets of S contracted at the next lower level of the decomposition and containing vertices of *inpath*(S). After a bottom-up computation of M_1, H_1, and H_2, we then perform a top-down computation of a family of $(i+1)$-partial heaps, $(M_3(S), H_3(S))$; M_3 is formed by removing some elements from M_1 and H_3 is formed by adding those elements to H_1. Finally, the desired output $H_T(v)$ can be constructed from the 1-partial heap $(M_3(v), H_3(v))$ at level T_0 in the decomposition.

Before describing our algorithms, let us bound a quantity useful in their analysis. Let m_i denote the sum of $|M_1(S)|$ over sets S contracted in T_i.

LEMMA 10. *For each i, $m_i = O(i|T_i|)$.*

Proof. By the definition of $M_1(S)$ above,

$$m_i = \sum_S \max(i+2, |M_2(S)|+1) \le \sum_S |M_2(S)| + i + 2 \le (i+2)|T_i| + \sum_S |M_2(S)|.$$

All sets $M_2(S)$ appearing in this sum are disjoint, and all are included in m_{i+1}, so we can simplify this formula to

$$m_i \le (i+2)|T_i| + m_{i+1} \le \sum_{j \ge i} (j+2)|T_j| \le \sum_{j \ge i} (j+2) \left(\frac{5}{6}\right)^{j-i} |T_i| = O(i|T_i|). \qquad \square$$

We use the following data structure to compute the sets $M_1(S)$ (which, recall, are sets of low-weight vertices on *inpath*(S)). For each interior set S, we form a priority queue $Q(S)$, from which we can retrieve the smallest weight vertex on *inpath*(S) not yet in $M_1(S)$. This data structure is very simple: if only one of the two subsets forming S contains vertices on *inpath*(S), we simply copy the minimum-weight vertex on that subset's priority queue, and otherwise we compare the minimum-weight vertices in each subset's priority queue and select the smaller of the two weights. If one of the two subsets' priority queue values change, this structure can be updated simply by repeating this comparison.

We start by setting all the sets $M_1(S)$ to be empty, then progress from the top down through the multilevel decomposition, testing for each set S in each tree T_i (in decreasing order of i) whether we have already added enough members to $M_1(S)$. If not, we add elements one at a time until there are enough to satisfy the definition above of $|M_1(S)|$. Whenever we add an element to $M_1(S)$ we add the same element to $M_1(S')$ for each lower level subset S' to which it also belongs. An element is added by removing it from $Q(S)$ and from the priority queues of the sets at each lower level. We then update the queues bottom up, recomputing the head of each queue and inserting it in the queue at the next level.

LEMMA 11. *The amount of time to compute $M_1(S)$ for all sets S in the multilevel partition, as described above, is $O(n)$.*

Proof. By Lemma 10, the number of operations in priority queues for subsets of T_i is $O(i|T_i|)$. So the total time is $\sum O(i|T_i|) = O(n\sum i(5/6)^i) = O(n)$. □

We next describe how to compute the heaps $H_1(S)$ for the vertices on *inpath*(S) that have not been chosen as part of $M_1(S)$. For this stage we work from the bottom up. Recall that S corresponds to one or two vertices of T_i; each vertex corresponds to a set S' contracted at a previous level of the multilevel partition. For each such S' along the path in S we will have already formed the partial heap $(M_1(S'), H_1(S'))$. We let $H_2(S')$ be a heap formed by adding the vertices in $M_1(S') - M_1(S)$ to $H_1(S')$. Since $M_1(S') - M_1(S)$ consists of at least one vertex (because of the requirement that $|M_1(S')| \geq |M_1(S)|+1$), we can form $H_2(S')$ as a 2-heap in which the root has degree one.

If S consists of a single vertex we then let $H_1(S) = H_2(S')$; otherwise we form $H_1(S)$ by combining the two heaps $H_2(S')$ for its two children. The time is constant per set S or linear overall.

We next compute another collection of partial heaps $(M_3(S), H_3(S))$ of vertices in *rootpath*(S) for each set S contracted at some level of the tree. If S is a set contracted to a vertex in T_i, we let $(M_3(S), H_3(S))$ be an $(i+1)$-partial heap. In this phase of the algorithm, we work top down. For each set S, consisting of a collection of vertices in T_{i-1}, we use $(M_3(S), H_3(S))$ to compute for each vertex S' the partial heap $(M_3(S'), H_3(S'))$.

If S consists of a single set S', or if S' is the parent of the two vertices in S, we let $M_3(S')$ be formed by removing the minimum weight element from $M_3(S)$ and we let $H_3(S')$ be formed by adding that minimum weight element as a new root to $H_3(S)$.

In the remaining case, if S' and *parent*(S') are both in S, we form $M_3(S')$ by taking the $i+1$ minimum values in $M_1(\textit{parent}(S')) \cup M_3(\textit{parent}(S'))$. The remaining values in $M_1(\textit{parent}(S')) \cup M_3(\textit{parent}(S')) - M_3(S')$ must include at least one value v greater than everything in $H_1(\textit{parent}(S'))$. We form $H_3(S')$ by sorting those remaining values into a chain, together with the root of heap $H_3(\textit{parent}(S'))$, and connecting v to $H_1(\textit{parent}(S'))$.

To complete the process, we compute the heaps $H_T(v)$ for each vertex v. Each such vertex is in T_0, so the construction above has already produced a 1-partial heap $(M_3(v), H_3(v))$. We must add the value for v itself and produce a true heap, both of which are easy.

LEMMA 12. *Given a tree T with weighted nodes, we can construct for each vertex v a 2-heap $H_T(v)$ of all nodes on the path from v to the root of the tree, in total time and space $O(n)$.*

Proof. The time for constructing (M_1, H_1) has already been analyzed. The only

remaining part of the algorithm that does not take constant time per set is the time for sorting remaining values into a chain, in time $O(i \log i)$ for a set at level i of the construction. The total time at level i is thus $O(|T_i| i \log i)$ which, summed over all i, gives $O(n)$. □

Applying this technique in place of Lemma 4 gives the following result.

THEOREM 3. *Given a digraph G and a shortest path tree from a vertex s, we can find an implicit representation of the k shortest s-t paths in G, in time and space $O(m + n + k)$.*

4. Maintaining path properties. Our algorithm can maintain along with the other information in $H(G)$ various forms of simple information about the corresponding s-t paths in G.

We have already seen that $H(G)$ allows us to recover the lengths of paths. However, lengths are not as difficult as some other information might be to maintain, since they form an additive group. We used this group property in defining $\delta(e)$ to be a difference of path lengths, and in defining edges of $P(G)$ to have weights that were differences of quantities $\delta(e)$.

We now show that we can in fact keep track of any quantity formed by combining information from the edges of the path using any monoid. We assume that there is some given function taking each edge e to an element *value*(e) of a monoid, and that given two edges e and f we can compute the composite value *value*$(e) \cdot$ *value*(f) in constant time. By associativity of monoids, the value *value*(p) of a path p is well defined. Examples of such values include the path length and number of edges in a path (for which composition is real or integer addition) and the longest or shortest edge in a path (for which composition is minimization or maximization).

Recall that for each vertex we compute a heap $H_G(v)$ representing the sidetracks reachable along the shortest path from v to t. For each node x in $H_G(v)$ we maintain two values: *pathstart*(x) pointing to a vertex on the path from v to t, and *value*(x) representing the value of the path from *pathstart*(x) to the head of the sidetrack edge represented by x. We require that *pathstart* of the root of the tree is v itself, that *pathstart*(x) be a vertex between v and the head of the sidetrack edge representing x, and that all descendents of x have *pathstart* values on the path from *pathstart*(x) to t. For each edge in $H_G(v)$ connecting nodes x and y we store a further value, representing the value of the path from *pathstart*(x) to *pathstart*(y). We also store for each vertex in G the value of the shortest path from v to t.

Then as we compute paths from the root in the heap $H(G)$, representing s-t paths in G, we can keep track of the value of each path merely by composing the stored values of appropriate paths and nodes in the path in $H(G)$. Specifically, when we follow an edge in a heap $H_G(v)$ we include the value stored at that edge, and when we take a sidetrack edge e from a node x in $H_G(v)$ we include *value*(x) and *value*(e). Finally we include the value of the shortest path to t from the tail of the last sidetrack edge to t. The portion of the value except for the final shortest path can be updated in constant time from the same information for a shorter path in $H(G)$, and the remaining shortest path value can be included again in constant time, so this computation takes $O(1)$ time per path found.

The remaining difficulty is computing the values *value*(x), *pathstart*(x), and also the values of edges in $H_G(v)$.

In the construction of Lemma 4, we need only compute these values for the $O(\log n)$ nodes by which $H_G(v)$ differs from $H_G($*parent*$(v))$, and we can compute each such value as we update the heap in constant time per value. Thus the construction

here goes through with unchanged complexity.

In the construction of Lemma 12, each partial heap at each level of the construction corresponds to all sidetracks with heads taken from some path in the shortest path tree. As each partial heap is formed the corresponding path is formed by concatenating two shorter paths. We let *pathstart*(x) for each root of a heap be equal to the endpoint of this path farthest from t. We also store for each partial heap the near endpoint of the path, and the value of the path. Then these values can all be updated in constant time when we merge heaps.

THEOREM 4. *Given a digraph G and a shortest path tree from a vertex s, and given a monoid with values value(e) for each edge $e \in G$, we can compute value(p) for each of the k shortest s-t paths in G, in time and space $O(m+n+k)$.*

5. Dynamic programming applications. Many optimization problems solved by dynamic programming or more complicated matrix searching techniques can be expressed as shortest path problems. Since the graphs arising from dynamic programs are typically acyclic, we can use our algorithm to find longest as well as shortest paths. We demonstrate this approach by a few selected examples.

5.1. The knapsack problem. The *optimization* 0-1 *knapsack problem* (or *knapsack problem* for short) consists of placing "objects" into a "knapsack" that has room for only a subset of the objects, and maximizing the total value of the included objects. Formally, one is given integers L, c_i, and w_i ($0 \le i < n$), and one must find $x_i \in \{0,1\}$ satisfying $\sum x_i c_i \le L$ and maximizing $\sum x_i w_i$. Dynamic programming solves the problem in time $O(nL)$; Dai et al. [15] show how to find the k best solutions in time $O(knL)$. We now show how to improve this to $O(nL+k)$ using longest paths in a DAG.

Let directed acyclic graph G have $nL+L+2$ vertices: two terminals s and t and $(n+1)L$ other vertices with labels (i,j), $0 \le i \le n$ and $0 \le j \le L$. Draw an edge from s to each $(0,j)$ and from each (n,j) to t, each having length 0. From each (i,j) with $i < n$, draw two edges: one to $(i+1,j)$ with length 0, and one to $(i+1, j+c_i)$ with length w_i (omit this last edge if $j + c_i > L$).

There is a simple one-to-one correspondence between s-t paths and solutions to the knapsack problem: given a path, define x_i to be 1 if the path includes an edge from (i,j) to $(i+1, j+c_i)$; instead let x_i be 0 if the path includes an edge from (i,j) to $(i+1,j)$. The length of the path is equal to the corresponding value of $\sum x_i w_i$, so we can find the k best solutions simply by finding the k longest paths in the graph.

THEOREM 5. *We can find the k best solutions to the knapsack problem as defined above, in time $O(nL+k)$.*

5.2. Sequence alignment. The *sequence alignment* or *edit distance* problem is that of matching the characters in one sequence against those of another, obtaining a matching of minimum *cost* where the cost combines terms for mismatched and unmatched characters. This problem and many of its variations can be solved in time $O(xy)$ (where x and y denote the lengths of the two sequences) by a dynamic programming algorithm that takes the form of a shortest path computation in a grid graph.

Byers and Waterman [8, 64] describe a problem of finding all near-optimal solutions to sequence alignment and similar dynamic programming problems. Essentially their problem is that of finding all s-t paths with length less than a given bound L. They describe a simple depth first search algorithm for this problem, which is

especially suited for grid graphs although it will work in any graph and although the authors discuss it in terms of general DAGs. In a general digraph their algorithm would use time $O(k^2m)$ and space $O(km)$. In the acyclic case discussed in the paper, these bounds can be reduced to $O(km)$ and $O(m)$. In grid graphs its performance is even better: time $O(xy + k(x + y))$ and space $O(xy)$. Naor and Brutlag [46] discuss improvements to this technique that among other results include a similar time bound for k shortest paths in grid graphs.

We now discuss the performance of our algorithm for the same length-limited path problem. In general one could apply any k shortest paths algorithm together with a doubling search to find the value of k corresponding to the length limit, but in our case the problem can be solved more simply: simply replace the breadth first search in $H(G)$ with a length-limited depth first search.

THEOREM 6. *We can find the k s-t paths in a graph G that are shorter than a given length limit L, in time $O(m+n+k)$ once a shortest path tree in G is computed.*

Even for the grid graphs arising in sequence analysis, our $O(xy + k)$ bound improves by a factor of $O(x + y)$ the times of the algorithms of Byers and Waterman [8] and Naor and Brutlag [46].

5.3. Inscribed polygons. We next discuss the problem of, given an n-vertex convex polygon, finding the "best" approximation to it by an r-vertex polygon, $r < n$. This arises, e.g., in computer graphics, in which significant speedups are possible by simplifying the shapes of faraway objects. To our knowledge the "k best solution" version of the problem has not been studied before. We include it as an example in which the best-known algorithms for the single solution case do not appear to be of the form needed by our techniques; however, one can transform an inefficient algorithm for the original problem into a shortest path problem that with our techniques gives an efficient solution for large enough k.

We formalize the problem as that of finding the maximum area or perimeter convex r-gon inscribed in a convex n-gon. The best known solution takes time $O(n \log n + n\sqrt{r \log n})$ [1]. However, this algorithm does not appear to be in the form of a shortest path problem, as needed by our techniques.

Instead we describe a less efficient technique for solving the problem by using shortest paths. Number the n-gon vertices v_1, v_2, etc. Suppose we know that v_i is the lowest numbered vertex to be part of the optimal r-gon. We then form a DAG G_i with $O(rn)$ vertices and $O(rn^2)$ edges, in r levels. In each level we place a copy of each vertex v_j, connected to all vertices with lower numbers in the previous level. Each path from the copy of v_i in the first level of the graph to a vertex in the last level of the graph has r vertices with numbers in ascending order from v_i, and thus corresponds to an inscribed r-gon. We connect one such graph for each initial vertex v_i into one large graph, by adding two vertices s and t, edges from s to each copy of a vertex v_i at the first level of G_i, and edges from each vertex on level r of each G_i to t. Paths in the overall graph G thus correspond to inscribed r-gons with any starting vertex.

It remains to describe the edge lengths in this graph. Edges from s to each v_i will have length zero for either definition of the problem. Edges from a copy of v_i at one level to a copy of v_j at the next level will have length equal to the Euclidean distance from v_i to v_j, for the maximum perimeter version of the problem, and edges connecting a copy of v_j at the last level to t will have length equal to the distance between v_j and the initial vertex v_i. Thus the length of a path becomes exactly the perimeter of the corresponding polygon, and we can find the k best r-gons by finding the k longest paths.

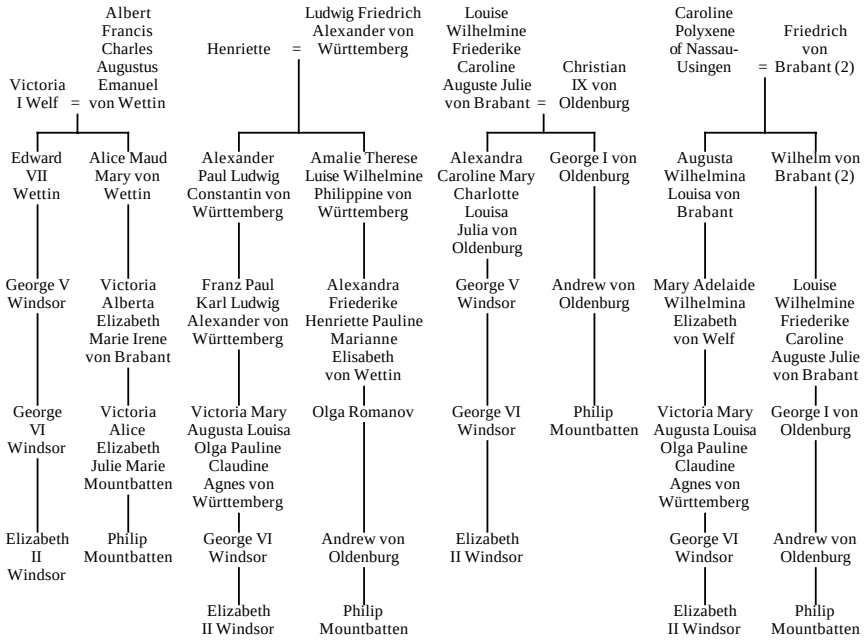


FIG. 7. *Some short relations in a complicated genealogical database.*

For the maximum area problem, we instead let the distance from v_i to v_j be measured by the area of the n-gon cut off by a line segment from v_i to v_j. Thus the total length of a path is equal to the total area outside the corresponding r-gon. Since we want to maximize the area inside the r-gon, we can find the k best r-gons by finding the k shortest paths.

THEOREM 7. *We can find the k maximum area or perimeter r-gons inscribed in an n-gon, in time $O(rn^3 + k)$.*

5.4. Genealogical relations. If one has a database of family relations, one may often wish to determine how some two individuals in the database are related to each other. Formalizing this, one may draw a DAG in which nodes represent people, and an arc connects a parent to each of his or her children. Then each different type of relationship (such as that of being a half-brother, great-aunt, or third cousin twice removed) can be represented as a pair of disjoint paths from a common ancestor (or couple forming a pair of common ancestors) to the two related individuals, with the specific type of relationship being a function of the numbers of edges in each path and of whether the paths begin at a couple or at a single common ancestor. In most families, the DAG one forms in this way has a tree-like structure, and relationships are easy to find. However, in more complicated families with large amounts of intermarriage, one can be quickly overwhelmed with many different relationships. For instance, in the British royal family, Queen Elizabeth and her husband Prince Philip are related in many ways, the closest few being second cousins once removed through King Christian IX of Denmark and his wife Louise, third cousins through Queen Victoria of England and her husband Albert, and fourth cousins through Duke Ludwig Friedrich Alexander of Württemberg and his wife Henriette (Figure 7). The single shortest relationship can be found as a shortest path in a graph formed by combining the DAG with its reversal, but longer paths in this graph

do not necessarily correspond to disjoint pairs of paths. A program my wife, Diana, and I wrote, Gene (`http://www.ics.uci.edu/~eppstein/gene/`), is capable of finding small numbers of relationships quickly using a backtracking search with heuristic pruning, but Gene starts to slow down when asked to produce larger numbers of relationships.

We now describe a technique for applying our k-shortest-path algorithm to this problem, based on a method of Perl and Shiloach [48] for finding shortest pairs of disjoint paths in DAGs. Given a DAG D, we construct a larger DAG D_1 as follows. We first find some topological ordering of D, and let $f(x)$ represent the position of vertex x in this ordering. We then construct one vertex of D_1 for each ordered pair of vertices (x, y) (not necessarily distinct) in D. We also add one additional vertex s in D_1. We connect (x, y) to (x, z) in D_1 if (y, z) is an arc of D and $f(z) > \max(f(x), f(y))$. Symmetrically, we connect (x, y) to (z, y) if (x, z) is an arc of D and $f(z) > \max(f(x), f(y))$. We connect s to all vertices in D_1 of the form (v, v).

LEMMA 13. *Let vertices u and v be given. Then the pairs of disjoint paths in D from a common ancestor a to u and v are in one-for-one correspondence with the paths in D_1 from s through (a, a) to (u, v).*

As a consequence, we can find shortest relationships between two vertices u and v by finding shortest paths in D_1 from s to (u, v).

THEOREM 8. *Given a DAG with n nodes and m edges, we can construct in $O(mn)$ time a data structure such that, given any two nodes u and v in a DAG, we can list (an implicit representation of) the k shortest pairs of vertex-disjoint paths from a common ancestor to u and v, in time $O(k)$. The same bound holds for listing all pairs with length less than a given bound (where k is the number of such paths). Alternately, the pairs of paths can be output in order by total length in time $O(\log i)$ to list the ith pair. As before, our representation allows constant-time computation of some simple functions of each path, and allows each path to be explicitly generated in time proportional to its length.*

For a proof of Lemma 13 and more details of this application, see [19].

6. Conclusions. We have described algorithms for the k-shortest-paths problem, improving by an order of magnitude previously known bounds. The asymptotic performance of the algorithm makes it an especially promising choice in situations when large numbers of paths are to be generated, and there already exist at least two implementations: one by Shibuya, Imai, and coworkers [52, 53, 54, 55] and one by Martins (`http://www.mat.uc.pt/~eqvm/eqvm.html`).

We list the following as open problems.

- The linear time construction when the shortest path tree is known is rather complicated. Is there a simpler method for achieving the same result? How quickly can we maintain heaps $H_T(v)$ if new leaves are added to the tree? (Lemma 4 solves this in $O(\log n)$ time per vertex, but it seems that at least $O(\log \log n)$ should be possible.)
- As described above, we can find the k best inscribed r-gons in an n-gon, in time $O(rn^3 + k)$. However, the best single-optimum solution has the much faster time bound $O(n \log n + n\sqrt{r \log n})$ [1]. Our algorithms for the k best r-gons are efficient (in the sense that we use constant time per r-gon) only when $k = \Omega(rn^3)$. The same phenomenon of overly large preprocessing times also occurs in our application to genealogical relationship finding: the shortest relationship can be found in linear time, but our k-shortest-relationship method takes time $O(mn + k)$. Can we improve these bounds?

- Are there properties of paths not described by monoids which we can nevertheless compute efficiently from our representation? In particular, how quickly can we test whether each path generated is simple?

Acknowledgments. I thank Greg Frederickson, Sandy Irani, and George Lueker for helpful comments on drafts of this paper.

REFERENCES

[1] A. Aggarwal, B. Schieber, and T. Tokuyama, *Finding a minimum weight K-link path in graphs with Monge property and applications*, in Proc. 9th Symp. Computational Geometry, Assoc. for Computing Machinery, 1993, pp. 189–197.

[2] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan, *Faster algorithms for the shortest path problem*, J. Assoc. Comput. Mach., 37 (1990), pp. 213–223.

[3] J. A. Azevedo, M. E. O. Santos Costa, J. J. E. R. Silvestre Madeira, and E. Q. V. Martins, *An algorithm for the ranking of shortest paths*, Eur. J. Operational Research, 69 (1993), pp. 97–106.

[4] A. Bako, *All paths in an activity network*, Mathematische Operationsforschung und Statistik, 7 (1976), pp. 851–858.

[5] A. Bako and P. Kas, *Determining the k-th shortest path by matrix method*, Szigma, 10 (1977), pp. 61–66 (in Hungarian).

[6] R. E. Bellman, *On a routing problem*, Quart. Appl. Math., 16 (1958), pp. 87–90.

[7] A. W. Brander and M. C. Sinclair, *A comparative study of k-shortest path algorithms*, in Proc. 11th UK Performance Engineering Workshop for Computer and Telecommunications Systems, 1995.

[8] T. H. Byers and M. S. Waterman, *Determining all optimal and near-optimal solutions when solving shortest path problems by dynamic programming*, Oper. Res., 32 (1984), pp. 1381–1384.

[9] P. Carraresi and C. Sodini, *A binary enumeration tree to find K shortest paths*, in Proc. 7th Symp. Operations Research, Methods Oper. Res. 45, Athenäum/Hain/Hanstein, 1983, pp. 177–188.

[10] G.-H. Chen and Y.-C. Hung, *Algorithms for the constrained quickest path problem and the enumeration of quickest paths*, Comput. Oper. Res., 21 (1994), pp. 113–118.

[11] Y. L. Chen, *An algorithm for finding the k quickest paths in a network*, Comput. Oper. Res., 20 (1993), pp. 59–65.

[12] Y. L. Chen, *Finding the k quickest simple paths in a network*, Inform. Process. Lett., 50 (1994), pp. 89–92.

[13] E. I. Chong, S. R. Maddila, and S. T. Morley, *On finding single-source single-destination k shortest paths*, in Proc. 7th Int. Conf. Computing and Information, Trent University, Canada, 1995.

[14] A. Consiglio and A. Pecorella, *Using simulated annealing to solve the K-shortest path problem*, in Proc. Conf. Italian Assoc. Operations Research, 1995.

[15] Y. Dai, H. Imai, K. Iwano, and N. Katoh, *How to treat delete requests in semi-online problems*, in Proc. 4th Int. Symp. Algorithms and Computation, Lecture Notes in Comput. Sci. 762, Springer-Verlag, New York, 1993, pp. 48–57.

[16] M. T. Dickerson and D. Eppstein, *Algorithms for proximity problems in higher dimensions*, Comput. Geom., 5 (1996), pp. 277–291.

[17] S. E. Dreyfus, *An appraisal of some shortest path algorithms*, Oper. Res., 17 (1969), pp. 395–412.

[18] El-Amin and Al-Ghamdi, *An expert system for transmission line route selection*, in Int. Power Engineering Conf, Vol. 2, Nanyang Technol. Univ., Singapore, 1993, pp. 697–702.

[19] D. Eppstein, *Finding common ancestors and disjoint paths in DAGs*, Tech. Report 95-52, Univ. of California, Dept. Information and Computer Science, Irvine, 1995.

[20] D. Eppstein, *Ten algorithms for Egyptian fractions*, Mathematica in Education and Research, 4 (1995), pp. 5–15.

[21] D. Eppstein, Z. Galil, and G. F. Italiano, *Improved sparsification*, Tech. Report 93-20, Univ. of California, Dept. Information and Computer Science, Irvine, 1993.

[22] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig, *Sparsification—A technique for speeding up dynamic graph algorithms*, in Proc. 33rd Symp. Foundations of Computer Science, IEEE, 1992, pp. 60–69.

[23] L. R. Ford, Jr. and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Prince-

ton, NJ, 1962.
[24] B. L. Fox, *k-th shortest paths and applications to the probabilistic networks*, in ORSA/TIMS Joint National Mtg., Vol. 23, 1975, p. B263.
[25] G. N. Frederickson, *Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees*, in Proc. 32nd Symp. Foundations of Computer Science, IEEE, 1991, pp. 632–641.
[26] G. N. Frederickson, *An optimal algorithm for selection in a min-heap*, Inform. and Comput., 104 (1993), pp. 197–214.
[27] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. Assoc. Comput. Mach., 34 (1987), pp. 596–615.
[28] M. L. Fredman and D. E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, in Proc. 31st Symp. Foundations of Computer Science, IEEE, 1990, pp. 719–725.
[29] A. V. Goldberg, *Scaling algorithms for the shortest paths problem*, SIAM J. Comput., 24 (1995), pp. 494–504.
[30] V. Hatzivassiloglou and K. Knight, *Unification-based glossing*, in Proc. 14th Int. Joint Conf. Artificial Intelligence, 1995, Morgan Kaufmann, San Francisco, pp. 1382–1389.
[31] G. J. Horne, *Finding the K least cost paths in an acyclic activity network*, J. Oper. Res. Soc., 31 (1980), pp. 443–448.
[32] L.-M. Jin and S.-P. Chan, *An electrical method for finding suboptimal routes*, in Int. Symp. Circuits and Systems, Vol. 2, IEEE, 1989, pp. 935–938.
[33] D. B. Johnson, *A priority queue in which initialization and queue operations take $O(\log \log D)$ time*, Math. Systems Theory, 15 (1982), pp. 295–309.
[34] N. Katoh, T. Ibaraki, and H. Mine, *An $O(Kn^2)$ algorithm for K shortest simple paths in an undirected graph with nonnegative arc length*, Trans. Inst. Electronics and Communication Engineers of Japan, E61 (1978), pp. 971–972.
[35] N. Katoh, T. Ibaraki, and H. Mine, *An efficient algorithm for K shortest simple paths*, Networks, 12 (1982), pp. 411–427.
[36] P. N. Klein, S. Rao, M. H. Rauch, and S. Subramanian, *Faster shortest-path algorithms for planar graphs*, in Proc. 26th Symp. Theory of Computing, Assoc. for Computing Machinery, 1994, pp. 27–37.
[37] N. Kumar and R. K. Ghosh, *Parallel algorithm for finding first K shortest paths*, Computer Science and Informatics, 24 (1994), pp. 21–28.
[38] A. G. Law and A. Rezazadeh, *Computing the K-shortest paths, under nonnegative weighting*, in Proc. 22nd Manitoba Conf. Numerical Mathematics and Computing, Congr. Numer., 92 (1993), pp. 277–280.
[39] E. L. Lawler, *A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem*, Management Science, 18 (1972), pp. 401–405.
[40] E. L. Lawler, *Comment on computing the k shortest paths in a graph*, Commun. Assoc. Comput. Mach., 20 (1977), pp. 603–604.
[41] E. Q. V. Martins, *An algorithm for ranking paths that may contain cycles*, Eur. J. Oper. Res., 18 (1984), pp. 123–130.
[42] S.-P. Miaou and S.-M. Chin, *Computing k-shortest path for nuclear spent fuel highway transportation*, Eur. J. Oper. Res., 53 (1991), pp. 64–80.
[43] E. Minieka, *On computing sets of shortest paths in a graph*, Commun. Assoc. Comput. Mach., 17 (1974), pp. 351–353.
[44] E. Minieka, *The K-th shortest path problem*, in ORSA/TIMS Joint National Mtg., Vol. 23, 1975, p. B/116.
[45] E. Minieka and D. R. Shier, *A note on an algebra for the k best routes in a network*, J. Inst. Mathematics and Its Applications, 11 (1973), pp. 145–149.
[46] D. Naor and D. Brutlag, *On near-optimal alignments of biological sequences*, J. Computational Biology, 1 (1994), pp. 349–366.
[47] A. Perko, *Implementation of algorithms for K shortest loopless paths*, Networks, 16 (1986), pp. 149–160.
[48] Y. Perl and Y. Shiloach, *Finding two disjoint paths between two pairs of vertices in a graph*, J. Assoc. Comput. Mach., 25 (1978), pp. 1–9.
[49] J. B. Rosen, S.-Z. Sun, and G.-L. Xue, *Algorithms for the quickest path problem and the enumeration of quickest paths*, Comput. Oper. Res., 18 (1991), pp. 579–584.
[50] E. Ruppert, *Finding the k shortest paths in parallel*, in Proc. 14th Symp. Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1200, Springer–Verlag, New York, 1997, pp. 475–486.

[51] T. Shibuya, *Finding the k shortest paths by AI search techniques*, Cooperative Research Reports in Modeling and Algorithms, 7 (1995), pp. 212–222.
[52] T. Shibuya, T. Ikeda, H. Imai, S. Nishimura, H. Shimoura, and K. Tenmoku, *Finding a realistic detour by AI search techniques*, in Proc. 2nd Intelligent Transportation Systems, 4 (1995), pp. 2037–2044.
[53] T. Shibuya and H. Imai, *Enumerating suboptimal alignments of multiple biological sequences efficiently*, in Proc. 2nd Pacific Symp. Biocomputing, World Scientific, Singapore, 1997, pp. 409–420.
[54] T. Shibuya and H. Imai, *New flexible approaches for multiple sequence alignment*, in Proc. 1st Int. Conf. Computational Molecular Biology, Assoc. for Computing Machinery, 1997, pp. 267–276.
[55] T. Shibuya, H. Imai, S. Nishimura, H. Shimoura, and K. Tenmoku, *Detour queries in geographical databases for navigation and related algorithm animations*, in Proc. Int. Symp. Cooperative Database Systems for Advanced Applications, 2 (1996), pp. 333–340.
[56] D. R. Shier, *Algorithms for finding the k shortest paths in a network*, in ORSA/TIMS Joint National Mtg., 1976, p. 115.
[57] D. R. Shier, *Iterative methods for determining the k shortest paths in a network*, Networks, 6 (1976), pp. 205–229.
[58] D. R. Shier, *On algorithms for finding the k shortest paths in a network*, Networks, 9 (1979), pp. 195–214.
[59] C. C. Skicism and B. L. Golden, *Solving k-shortest and constrained shortest path problems efficiently*, in Network Optimization and Applications, B. Shetty, ed., Annals Oper. Res. 20, Baltzer Science Publishers, Bussum, The Netherlands, 1989, pp. 249–282.
[60] K. Sugimoto and N. Katoh, *An algorithm for finding k shortest loopless paths in a directed network*, Trans. Information Processing Soc. Japan, 26 (1985), pp. 356–364 (in Japanese).
[61] J. W. Suurballe, *Disjoint paths in a network*, Networks, 4 (1974), pp. 125–145.
[62] R. E. Tarjan, *Data Structures and Network Algorithms*, CBMS-NSF Reg. Conf. Ser. Appl. Math. 44, SIAM, Philadelphia, 1983.
[63] R. Thumer, *A method for selecting the shortest path of a network*, Z. Oper. Res., Serie B (Praxis), 19 (1975), pp. B149–153 (in German).
[64] M. S. Waterman, *Sequence alignments in the neighborhood of the optimum*, Proc. Natl. Acad. Sci. USA, 80 (1983), pp. 3123–3124.
[65] M. M. Weigand, *A new algorithm for the solution of the k-th best route problem*, Computing, 16 (1976), pp. 139–151.
[66] A. Wongseelashote, *An algebra for determining all path-values in a network with application to k-shortest-paths problems*, Networks, 6 (1976), pp. 307–334.
[67] A. Wongseelashote, *Semirings and path spaces*, Discrete Math., 26 (1979), pp. 55–78.
[68] J. Y. Yen, *Finding the K shortest loopless paths in a network*, Management Science, 17 (1971), pp. 712–716.
[69] J. Y. Yen, *Another algorithm for finding the K shortest-loopless network paths*, in Proc. 41st Mtg. Operations Research Society of America, 20 (1972), p. B/185.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 674–699

EXACT LEARNING OF DISCRETIZED GEOMETRIC CONCEPTS*

NADER H. BSHOUTY†, PAUL W. GOLDBERG‡, SALLY A. GOLDMAN§, AND H. DAVID MATHIAS¶

Abstract. We first present an algorithm that uses membership and equivalence queries to exactly identify a discretized geometric concept defined by the union of m axis-parallel boxes in d-dimensional discretized Euclidean space where each coordinate can have n discrete values. This algorithm receives at most md counterexamples and uses time and membership queries polynomial in m and $\log n$ for any constant d. Furthermore, all equivalence queries can be formulated as the union of $O(md \log m)$ axis-parallel boxes.

Next, we show how to extend our algorithm to efficiently learn, from *only* equivalence queries, any discretized geometric concept generated from any number of halfspaces with any number of known (to the learner) slopes in a constant dimensional space. In particular, our algorithm exactly learns (from equivalence queries *only*) unions of discretized axis-parallel boxes in constant dimensional space in polynomial time. Furthermore, this equivalence query only algorithm can be modified to handle a polynomial number of lies in the counterexamples provided by the environment.

Finally, we introduce a new complexity measure that better captures the complexity of the union of m boxes than simply the number of boxes and the dimension. Our new measure, σ, is the number of segments in the target, where a segment is a maximum portion of one of the sides of the target that lies entirely inside or entirely outside each of the other halfspaces defining the target. We present a modification of our first algorithm that uses time and queries polynomial in σ and $\log n$. In fact, the time and queries (both membership and equivalence) used by this single algorithm are polynomial for *either* m or d constant.

Key words. computational learning, geometric concepts, exact learning, membership and equivalence queries

AMS subject classifications. 68Q25, 68T05

PII. S0097539794274246

1. Introduction.

Recently, learning geometric concepts in d-dimensional Euclidean space has been the subject of much research [6, 15, 17, 28, 30, 31, 32, 34]. We study the problem of learning geometric concepts under the model of learning with queries [1] in which the learner is required to output a final hypothesis that correctly classifies *every* point in the domain. To apply such a learning model to a geometric domain, it is necessary to look at a discretized (or digitalized) version of the domain. We use d to denote the number of dimensions and n to denote the number of discrete values that exist in each dimension. Thus a discretized geometric concept G is a set of

*Received by the editors September 6, 1994; accepted for publication (in revised form) January 12, 1997; published electronically August 4, 1998. Portions of this paper appear in preliminary form in [22] and [12].

http://www.siam.org/journals/sicomp/28-2/27424.html

†Department of Computer Science, University of Calgary, Calgary, AB, Canada T2N 1N4 and Department of Computer Science, Technion 32000, Haifa, Israel (bshouty@csa.technion.ac.il). The research of this author was supported in part by the NSERC of Canada.

‡Department of Computer Science, University of Warwick, Coventry CV47AL, UK (pwg@dcs.warwick.ac.uk). The research of this author was performed while visiting Washington University with support from NSF NYI grant CCR-9357707, and while at Sandia National Lab with support from U.S. Department of Energy contract DE-AC04-76AL85000.

§Department of Computer Science, Washington University, St. Louis, MO 63130 (sg@cs.wustl.edu). The research of this author was supported in part by NSF grant CCR-9110108 and NSF NYI grant CCR-9357707 with matching funds provided by Xerox PARC and WUTA.

¶Ohio State University, Columbus, OH 43210 (dmath@cis.ohio-state.edu). The research of this author was performed while visiting Washington University with support from NSF NYI grant CCR-9357707 with matching funds provided by Xerox PARC and WUTA.

integer points $G \subseteq \mathcal{N}_n^d$, where $\mathcal{N}_n = \{1, \ldots, n\}$. In this paper we consider discretized geometric concepts whose boundaries are defined by hyperplanes of known slope.

We begin by studying a special case: the well-studied class of unions of axis-parallel boxes. (By a "box," we mean an axis-aligned hypercuboid.) The algorithm for this special case is easily extended to learn discretized geometric concepts defined by axis-parallel hyperplanes. We use BOX_n^d to denote the class of axis-parallel boxes[1] over $\mathcal{N}_n^d$, and $\bigcup_{\leq m} \text{BOX}_n^d$ to denote the class of the union of at most m concepts from BOX_n^d. Let c be a concept in $\bigcup_{\leq m} \text{BOX}_n^d$. We say that c is defined by $s \leq m$ boxes from BOX_n^d if s is the minimum number of boxes whose union is equivalent to c. We note that it is easy to show that this class is a generalization of disjunctive normal form (DNF) formulas and a special case of the class of unions of intersections of halfspaces over $\mathcal{N}_n^d$.

We first present a query algorithm that exactly learns $\bigcup_{\leq m} \text{BOX}_n^d$ with at most $md + 1$ equivalence queries[2] and $O((4m)^d + md(\log n + d))$ membership queries and computation time. Thus our algorithm exactly learns the union of m discretized axis-parallel boxes over $\mathcal{N}_n^d$ in polynomial time for any constant d.

The hypothesis class used by this algorithm can be evaluated in time $O(d \log m \log n)$. Furthermore, in $O((2m)^{2d})$ time we can transform our hypothesis to the union of at most $O(md \log m)$ boxes. Thus we obtain the stronger result that our algorithm can exactly learn the union of m axis-parallel boxes over $\mathcal{N}_n^d$ while making at most $md+1$ equivalence queries, where each equivalence query is simply the union of $O(md \log m)$ concepts from BOX_n^d; making $O((4m)^d+md(\log n+d))$ membership queries, and using $O((md)^2 \log m \cdot (2m)^{2d} + md \log n)$ computation time. Thus for any constant d, this algorithm still uses time and queries polynomial in m and $\log n$. We also describe a variation of this basic algorithm that uses *only* equivalence queries and still has complexity polynomial in m for time and queries and $\log n$ for d constant. Our algorithm uses $O((8d^2 m \log n)^d)$ equivalence queries and computation time.

Next we study the problem of learning with only equivalence queries the class of discretized geometric concepts in which the hyperplanes defining the boundaries of the concept need *not* be axis parallel but rather can have any known slopes. That is, the geometric discretized concepts we study here are those whose boundaries lie on hyperplanes $\{x = (x_1, \ldots, x_d) \mid \sum_{j=1}^d a_{ij} x_j = b\}$ for $i = 1, \ldots, |S|$, where S is the set of slopes of the hyperplanes. The possible slopes of those hyperplanes, i.e., $a_i = (a_{i1}, \ldots, a_{id})$, are known to the learner, but the same slope with different shifts b can be used for many hyperplanes in the target concept g. Note that if we choose the slopes $S = \{e_i\}$, the standard basis, then we get the special case in which all hyperplanes are axis parallel.

Let $S \subset \mathcal{Z}^d$ where $\mathcal{Z}$ is the set of integers, and let $\|S\|$ denote the representation size of S (the sum of the logarithms of the absolute values of the integers in S). Let g be a geometric concept whose boundaries lie on m hyperplanes in $\mathcal{N}_n^d$ with slopes from S. A key result of this paper is that for any constant d, any such geometric concept is exactly learnable in $poly(l, m, \|S\|, \log n)$ time and equivalence queries even if the equivalence oracle lies on l counterexamples. So for example, if the space is the plane $\mathcal{N}_n^2$ and $S = \{(0,1), (1,0), (1,1), (1,-1), (1,2), (2,1), (1,-2), (2,-1)\}$, then our algorithm can efficiently learn the geometric concepts generated from lines that make angles $0, 90, 135, 45, 120,\ 150, 30$, and 60, respectively, with the x-axis, in polynomial

[1] Note that we include in BOX_n^d boxes with zero size in any dimension.

[2] The final equivalence query is the correct hypothesis, and thus at most md counterexamples are received.

time. (As this example illustrates, the representation that we use for a slope is that derived from the formula, given above, defining a hyperplane.) In higher constant dimensional space our algorithm can efficiently learn any geometric concept whose boundary slopes are known. Another generalization of this result is an algorithm to exactly learn polynomially sized decision trees over the basis "Is $x_i \succ c$," where $\succ \in \{>, <, \geq, \leq\}$ in constant dimensional space.

Finally, we reexamine our first algorithm for learning the union of m discretized boxes. We introduce a new complexity measure that better captures the complexity of a union of boxes than simply the number of boxes and dimensions. More specifically, our new measure, σ, is the number of segments in the target concept, where a segment is a maximum portion of one of the defining hyperplanes of the target that lies entirely inside or entirely outside each of the other defining hyperplanes. We show that $\sigma \leq (2m)^d$. We present an improvement of our first algorithm that uses time and queries polynomial in σ and $\log n$. The hypothesis class used by this modified algorithm is that of decision trees of height at most $2md$. Thus, observe that the hypothesis output (and the intermediate hypotheses) can be evaluated in polynomial time without any restrictions on m or d. We then use an alternate analysis of this algorithm to show that the time and queries used are polynomial in d and $\log n$ for any constant m, thus generalizing the exact learnability of DNF formulas with a constant number of terms. Combining these two methods of analysis, we get the interesting result that this single algorithm is efficient for *either* m or d constant.

The paper is organized as follows. In section 2 we describe the learning model that we use. Next, in section 3 we summarize the previous work on learning geometric concepts. Then in section 4 we give some preliminary definitions. Section 5 describes our results for learning unions of boxes with membership and equivalence queries. We present this algorithm, in part, because it introduces the approach used to obtain our other results and also because it uses very few equivalence queries, which is of interest if one's goal is to minimize the number of prediction errors made by the learner [13]. Next, in section 6 we describe a modification of this algorithm that efficiently learns the union of boxes in constant dimensional space with only equivalence queries. In section 7 we present our extensions to learning the class of geometric concepts defined by any hyperplanes of known slopes using only equivalence queries. In section 8 we describe how to modify this algorithm to handle the situation in which there are lies in the answers to the equivalence queries. In section 9 we present our new complexity measure and describe a modification of our first algorithm that runs in polynomial time with respect to this complexity measure. Finally, in section 10 we conclude with some open problems. This paper subsumes the results presented by Goldberg, Goldman, and Mathias [22] and includes several results given by Bshouty [11].

2. Learning model. The learning model we use in this paper is that of learning with queries developed by Angluin [1]. When applied to our class of discretized geometric concepts, the learner's goal is to learn *exactly* how an unknown target concept, g, drawn from the *concept class* $\mathcal{G} \subseteq 2^{\mathcal{N}_n^d}$, classifies as positive or negative all instances from the *instance space* $\mathcal{N}_n^d$. Thus each concept is the set of instances from $\mathcal{N}_n^d$ that it classifies as positive. We say that $y \in \mathcal{N}_n^d$ is a positive instance for target concept g if $y \in g$ (also denoted $g(y) = 1$) and say that y is a negative instance otherwise (also denoted $g(y) = 0$). It is often convenient to view the target concept g as the Boolean function $g : \mathcal{N}_n^d \to \{0, 1\}$. A *hypothesis* h is a polynomial-time algorithm that, given any $y \in \mathcal{N}_n^d$, outputs a prediction for $g(y)$. Throughout we use y to denote an instance (i.e., a point in $\mathcal{N}_n^d$) and $x = (x_1, \ldots, x_d)$ to denote the

variables associated with the d axes. For example, for $d = 2$, $2x_1 + 3x_2 = 7$ defines a two-dimensional hyperplane. For the point $y = (2, 1)$, y is on this hyperplane.

As mentioned above, the learning criterion in this paper is that of *exact identification*. In order to achieve exact identification, the learner's final hypothesis, h, must be such that $h(y) = g(y)$ for all instances $y \in \mathcal{N}_n^d$. To achieve this goal the learner is provided with two types of queries with which to learn about g. A *membership query*, $\mathrm{MQ}(y)$, returns "yes" if $g(y) = 1$ and returns "no" if $g(y) = 0$. We note that learning using only membership queries is quite difficult for many concept classes. For example, to learn a single positive point in an n^d discrete grid requires $O(n^d)$ membership queries. An *equivalence query*, $\mathrm{EQ}(h)$, returns "yes" if h is logically equivalent to g or returns a counterexample otherwise. A *positive counterexample* y is an instance such that $g(y) = 1$ and $h(y) = 0$. Similarly, a *negative counterexample* is such that $g(y) = 0$ and $h(y) = 1$. Equivalence queries are answered by a computationally unbounded adversary with knowledge of the target concept and the learning algorithm. Several of the algorithms we present use only equivalence queries. The others use *both* membership and equivalence queries. In some domains, presenting the learner with a single positive instance (as a counterexample to an equivalence query) makes learning with membership queries feasible (see the discussion of *geometric probing* in section 3).

An exact learning algorithm is said to run in polynomial time if the computation time and the number of queries are polynomial in both the size[3] of an example (whether it is for a membership query or a counterexample from an equivalence query) and the size of the target concept. Here an example is one of the n^d points in $\mathcal{N}_n^d$ and thus it can be represented with $d\lceil \lg n \rceil$ bits. Each box in the target concept can be encoded with $2d\lceil \lg n \rceil$ bits, and thus encoding the entire target concept uses $2dm\lceil \lg n \rceil$ bits. Thus for the problems studied here, a polynomial-time algorithm must use time and queries polynomial in $d, \log n$, and m.

An *l-liar* equivalence oracle is an oracle that is allowed to lie at most l times during the learning session when providing counterexamples to equivalence queries. The teacher is allowed at most l lies *total* even if multiple lies are for the same instance. This definition seeks to capture a notion of erroneous data and robust, error-tolerant algorithms. We seek a more efficient approach than testing each counterexample we receive using a membership query, especially in view of the fact that we are interested in *algorithms that may use only equivalence queries*.

Another important learning model is the PAC model introduced by Valiant [38]. In this model the learner is presented with labeled examples chosen at random according to an unknown, arbitrary distribution $\mathcal{D}$ over the instance space. Given values for parameters ϵ and δ, the learner's goal is to output a hypothesis that with high probability, at least $(1 - \delta)$, correctly classifies most of the instance space. That is, the weight, under $\mathcal{D}$, of misclassified instances must be at most ϵ. This is in contrast to the query learning model in which the learner is required to classify correctly every instance in the instance space. The learner is permitted time polynomial in $1/\epsilon$, $1/\delta$, the size of an example, and the size of the target concept to formulate a hypothesis. The relationship between the PAC model and the query model is well understood. Angluin [1] showed that any class that is learnable using only equivalence queries is also PAC learnable. The relationship is unchanged by the addition of membership queries to each model. Blum [8] showed that PAC learnability does not imply query learnability. The concept class studied here has also been considered in the PAC

[3] By size we mean the number of bits to encode the example.

model, as summarized in the next section.

3. Previous work. The problem of learning geometric concepts over a discrete domain was extensively studied by Maass and Turán [31, 32]. One of the geometric concepts that they studied was the class BOX_n^d. They showed that if the learner is restricted to make only equivalence queries in which each hypothesis was drawn from BOX_n^d then $\Omega(d \log n)$ queries are needed to achieve exact identification [28, 31]. Auer [6] improves this lower bound to $\Omega(\frac{d^2}{\log d} \log n)$.

If one always makes an equivalence query using the simple hypothesis that produces the smallest box consistent with the previously seen examples, then the resulting algorithm makes $O(dn)$ equivalence queries. An algorithm making $O(2^d \log n)$ equivalence queries was given by Maass and Turán [30]. The best result known for learning the class BOX_n^d was provided by Chen and Maass [17]. They gave an algorithm making $O(d^2 \log n)$ equivalence queries. They also provide an algorithm to learn the union of two axis-parallel rectangles in the discretized space $\{1, \ldots, n\} \times \{1, \ldots, m\}$ in time polynomial in $\log n$ and $\log m$, where one rectangle has a corner at $(0, m)$ and the other has a corner at $(n, 0)$. More recently, Chen [15] gave an algorithm that used equivalence queries to learn general unions of two boxes in the (discretized) plane. The algorithm uses $O(\log^2 n)$ equivalence queries and involves a detailed case analysis of the shapes formed by the two rectangles.

Homer and Chen [25] presented an algorithm to learn the union of m rectangles *in the plane* using $O(m^3 \log n)$ queries (both membership and equivalence) and $O(m^5 \log n)$ time. The hypothesis class of their algorithm is the union of $8m^2 - 2$ rectangles. In work independent of ours, Chen and Homer [16] have improved upon their earlier result by giving an algorithm that learns any concept from $\bigcup_{\leq m} \text{BOX}_n^d$ using $O(m^{2(d+1)} d^2 \log^{2d+1} n)$ equivalence queries by efficiently applying the bounded injury method from recursive function theory. This algorithm appears in [11] along with the equivalence-query algorithms presented here in sections 6 and 7. While the Chen and Homer result is very similar to our result of section 6, they use a very different technique to obtain the result. Also, our algorithm uses only $O((8d^2 m \log n)^d)$ equivalence queries.

Finally, in other independent work, Maass and Warmuth [34] have developed, as part of a more general result, an algorithm to learn any concept from $\bigcup_{\leq m} \text{BOX}_n^d$ using $O(md \log n)$ equivalence queries and $O\left((md \log n)^{O(d)}\right)$ computation time. In addition their technique enables them to efficiently learn a single box in a constant dimensional space that is not axis parallel but uses slopes from a known set of slopes. Their algorithm improves upon our result of section 6, in that the number of equivalence queries has polynomial dependence on m, d, and $\log n$. However, their work does *not* give results comparable to the other results presented in this paper. Most notably, in section 7 we give an algorithm that can learn the union of non-axis-parallel boxes (using slopes from a known set of slopes) versus just learning a single box.

Closely related to the problem of learning the union of discretized boxes, is the problem of learning the union of nondiscretized boxes in the PAC model [38]. Blumer et al. [10] present an algorithm to PAC-learn an m-fold union of boxes in E^d by drawing a sufficiently large sample of size $m' = \mathit{poly}\left(\frac{1}{\epsilon}, \lg \frac{1}{\delta}, m, d\right)$ and then performing a greedy covering over the at most $(\frac{em'}{2d})^{2d}$ boxes defined by the sample. Thus for d constant this algorithm runs in polynomial time. Long and Warmuth [29] present an algorithm to PAC-learn this same class by again drawing a sufficiently large sample and constructing a hypothesis that consists of at most $m(2d)^m$ boxes consistent with

the sample. Thus both the time and the sample complexity of their algorithm depend polynomially on $m, d^m, \frac{1}{\epsilon}$, and $\lg \frac{1}{\delta}$. For m constant this yields an efficient PAC algorithm.

We note that either of these PAC algorithms can be applied to the class $\bigcup_{\leq m} \text{BOX}_n^d$ giving efficient PAC algorithms for this class for either d constant or m constant. As discussed by Maass and Turán [31], the task of a concept learning algorithm is to provide a "smart" hypothesis based on the data available. In other words, the hypothesis must be carefully chosen so that as much information as possible is obtained from each counterexample. To illustrate this point consider the task of learning a concept of a half interval over $\{1, \ldots n\}$ (so an example $x \in \{1, \ldots, n\}$ is positive iff $x \leq r$ where $0 \leq r \leq n$ is not known). Within the PAC model a satisfactory hypothesis would be $r = h$, where h is the maximum positive example (0 if no positive examples have been seen). However, in the exact learning model when using *only equivalence queries* this hypothesis performs very poorly in that each counterexample could just be one to the right of the last one. Thus n counterexamples may be needed. However, if one uses the "smarter" hypothesis of $r = (h+g)/2$, where h is the maximum positive example seen and g is the minimum negative example seen ($n+1$ if no negative examples have been seen), then at most $\lceil \log n \rceil$ counterexamples are needed. More generally, the results from Blumer et al. [10] show that under the PAC model any concise hypothesis that is consistent with the data is satisfactory. In other words, the PAC model provides no suitable basis for distinction among different consistent hypotheses. On the other hand, a criterion for defining a "smart" hypothesis is implicitly contained within the query learning model. One must select hypotheses for the equivalence queries so that sufficient progress is made with each counterexample. This requirement of selecting a "smart" hypothesis makes the problem of obtaining an efficient algorithm to learn *exactly* the class $\bigcup_{\leq m} \text{BOX}_n^d$ significantly harder than obtaining the corresponding PAC result. Also Blum [8] has shown that if one-way functions exist, then there exist functions that are PAC-learnable but not exactly learnable.

Finally, under a variation of the PAC model in which membership queries can be made, Frazier et al. [21] have given an algorithm to PAC-learn the m-fold union of boxes in E^d for which each box is entirely contained within the positive quadrant *and* contains the origin. Furthermore, their algorithm learns this subclass of general unions of boxes in time polynomial in both m and d. Recall that since $\bigcup_{\leq m} \text{BOX}_n^d$ generalizes DNF, a polynomial-time algorithm for arbitrary d and m would solve the problem of learning DNF. Observe that the class considered by Frazier et al. is a generalization of the class of DNF formulas in which all variables only appear negated.

While there has been some work addressing the general issue of mislabeled training examples in the PAC model [3, 27, 37, 26], there has been little research on learning geometric concepts with noise. Auer [6] investigates exact learning of boxes where some of the counterexamples, given in response to equivalence queries, are noisy. Auer shows that BOX_n^d is learnable using hypotheses from BOX_n^d if and only if the fraction of noisy examples is less than $1/(d+1)$ and presents an efficient algorithm that handles a noise rate of $1/(2d+1)$. In the query learning model, Angluin and Kriķis [2] examine the case in which membership queries can be answered incorrectly under adversarial control.

There has also been some work on learning discretized geometric concepts defined by non–axis-parallel hyperplanes. Maass and Turán [33] study the problem of learning a single discretized halfspace using only equivalence queries. They give an efficient algorithm using $O(d^2(\log d + \log n))$ queries and give an information theoretic lower

bound of $\binom{d}{2}$ on the number of queries when all hypotheses are discretized halfspaces. There has also been work on learning non–axis-parallel discretized rectangles with only equivalence queries. Maass and Turán [32] show an $\Omega(n)$ information theoretic lower bound on the number of equivalence queries when the hypotheses must be drawn from the concept class. Contrasting this lower bound, Bultman and Maass [14] give an efficient algorithm that uses membership and equivalence queries to learn this class using $O(\log n)$ equivalence queries. Their algorithm returns a hypothesis consisting of a description of the vertices and edges of the polygon.

Computational geometry researchers have looked at the slightly related problem of *geometric probing* (for example, see [36]). Geometric probing studies how to identify, verify, or determine some property of an unknown geometrical object using a measuring device known as a probe. In one special case of geometric probing the aim is to construct (or learn) an unknown convex polygon given a point inside the polygon along with the ability to make a probe in which the algorithm can "shoot" a ray in a specified direction to find out the location where the ray hits the polygon. Note that using a binary search technique, such a probe can be simulated for this problem with a polynomial number of membership queries (for discretized domains). Thus such work can be thought of as learning a convex polygon from only membership queries along with a single positive example. However, when working with nonconvex objects, the probe used in such work is more powerful than a membership query.

Geometric testing (for example, see [35, 4]) is a subarea of geometric probing involved with solving verification problems. The work that has been done on this problem [5, 7, 23, 24] involves determining which of a finite set of models is being probed, and seems to be the most closely related to this paper among the geometric probing literature. The restriction to a finite set of models effectively discretizes the domain of geometric points, as is done here by considering points with bounded integer coordinates.

4. Preliminaries. Let $\mathcal{N}, \mathcal{Z}$, and $\mathcal{R}$ be the set of nonnegative integers, integers, and reals, respectively. Recall, as discussed in section 2, that we use $x_1, \ldots, x_d$ to denote the variables associated with the d axes. Let $\mathcal{N}_n = \{1, \ldots, n\}$ and $S = \{a_1, \ldots, a_s\} \subset \mathcal{Z}^d$ be a set of slopes. A d-dimensional *hyperplane* is $\{x = (x_1, \ldots, x_d) \mid \sum_{j=1}^{d} a_{ij} x_j = b\}$ for some $a_{ij} \in \mathcal{Z}, b \in \mathcal{R}$. A *halfspace* is $\{x = (x_1, \ldots, x_d) \mid \sum_{j=1}^{d} a_{ij} x_j \succ b\}$, where $\succ \in \{>, \geq, <, \leq\}$. A *discretized hyperplane* (respectively, halfspace) is $H \cap \mathcal{N}_n^d$ for some hyperplane (respectively, halfspace) H. A *geometric concept generated from hyperplanes* with slopes from $S \subset \mathcal{Z}^d$ is a set $\hat{g} \subset \mathcal{R}^d$ whose boundaries are defined by hyperplanes with slopes from S. A *discretized geometric concept generated from hyperplanes* with slopes from $S \subset \mathcal{Z}^d$ is $g = \hat{g} \cap \mathcal{N}_n^d$.

The *complexity* $C_S(\hat{g})$ of a geometric concept $\hat{g}$ is the minimum number of hyperplanes with slopes from S such that their union contains the boundary of $\hat{g}$. The complexity $C_S(g)$ of a discretized geometric concept g is the minimum $C_S(\hat{g})$ over all geometric concepts $\hat{g}$ that satisfy $g = \hat{g} \cap \mathcal{N}_n^d$. By a simple information theoretic argument, it follows that any exact learning algorithm for a nontrivial discretized geometric concept g cannot run in time less than the complexity $C_S(g)$. Also, the complexity of learning one box in $\mathcal{N}_n^d$ is at least $\Omega(d \log n)$. The input for our algorithms is S, together with n and d. We use $\|S\|$ to denote the sum of the logarithms of the absolute values of the integers in S and call this the *size* of S. Thus, a learning algorithm, for the class of discretized geometric concepts with complexity measure $C_S(g)$, is a polynomial-time algorithm if the time and query complexities are $poly(\|S\|, C_S(g), \log n, d)$.

Observe that if we choose the slopes $S = \{e_i\}$, the standard basis, then for any discretized geometric concept g defined by the union of m boxes in d dimensional space, $C_S(g) \leq 2md$ since at most $2d$ halfspaces are needed to define the boundaries of each box.

5. Learning unions of boxes with membership and equivalence queries. In this section we present an algorithm that exactly identifies any concept from $\bigcup_{\leq m} \text{BOX}_n^d$ (so $S = \{e_i\}$, the standard basis) while receiving at most md counterexamples and using time and membership queries that are polynomial in m and $\log n$ for any constant d. This section serves two purposes: (1) the other algorithms presented build upon this basic algorithm, and thus for ease of exposition we present it here, and (2) it uses very few equivalence queries, which is of interest if one's goal is to minimize the number of prediction errors made by the learner [13].

The following definition is used throughout this section.

DEFINITION 1. *For a discretized geometric concept g, we define a $+/^i-$ pair to be a positive point $y_+ = (y_1, \ldots, y_i, \ldots, y_d)$ paired with a negative point y_- where $y_- = (y_1, \ldots, y_i+1, \ldots y_d)$ or $y_- = (y_1, \ldots, y_i-1, \ldots, y_d)$, where we implicitly assume that any point outside $\mathcal{N}_n^d$ is a negative point.*

We define the *halfspace* associated with a given $+/-$ pair to be the unique, axis-aligned halfspace H that contains the positive but not the negative point. So for a $+/^i-$ pair where the positive point's ith coordinate is c, if the negative point's ith coordinate is $c+1$, then H is given by $x_i \leq c$. Similarly, if the negative point's ith coordinate is $c-1$, then H is given by $x_i \geq c$. We define the associated *hyperplane* to be the set of all points satisfying $x_i = c$.

For each of the d dimensions, we maintain a set $\mathcal{H}_i$ of hyperplanes discovered for that dimension that define the boundaries of g. We let $\mathcal{H} = \bigcup_{i=1}^d \mathcal{H}_i$. Observe that in dimension i we have decomposed $\mathcal{N}_n^d$ into up to $2|\mathcal{H}_i|+1$ regions: $|\mathcal{H}_i|$ corresponding to the hyperplanes themselves and $|\mathcal{H}_i|+1$ corresponding to the "strips" obtained when $\mathcal{N}_n^d$ is cut by each of the $|\mathcal{H}_i|$ hyperplanes. Recall that $|\mathcal{H}_i| \leq 2m$, and thus our final hypothesis divides $\mathcal{N}_n^d$ into a *grid* $G_{\mathcal{H}}$ of

$$\prod_{i=1}^{d}(2|\mathcal{H}_i|+1) \leq \prod_{i=1}^{d}(4m+1) = (4m+1)^d \tag{5.1}$$

regions (or connected components). We say that $G_{\mathcal{H}}$ is *consistent* if for any two points, with known classification, in any region of $G_{\mathcal{H}}$, the points have the same classification. Given a consistent $G_{\mathcal{H}}$, a hypothesis is obtained by simply classifying all points according to the unique classification of all known points in that region (with negative used as a default).

We now demonstrate that we can represent such a hypothesis so that it is very efficient to evaluate. For each dimension i we maintain a balanced binary search tree T_i where each internal node corresponds to one of the hyperplanes in $\mathcal{H}_i$ and each leaf node corresponds to one of the strips created. The ith coordinate of the points in a hyperplane in $\mathcal{H}_i$ is used as the key for the associated node in T_i. For each leaf node v of T_i we keep a pair $(\min_v^i, \max_v^i)$, where $\min_v^i$ (respectively, $\max_v^i$) holds the minimum (respectively, maximum) x such that x is the ith coordinate of some point in the region corresponding to leaf v. (For the internal nodes, the key itself serves the role of both $\min_v^i$ and $\max_v^i$.)

We define $R_i = \{[\min_v^i, \max_v^i] \mid v$ is a node of T_i and $\min_v^i$ and $\max_v^i$ are the minimum and maximum values of the ith coordinates of points in $v\}$. Let $\mathcal{T} =$

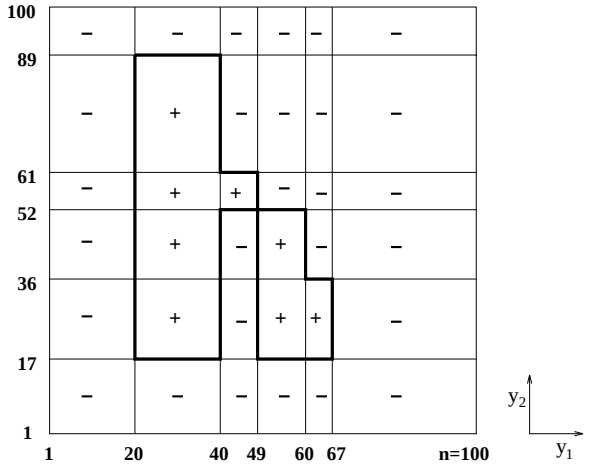


FIG. 1. *This shows the final set of regions corresponding to the target concept, formed by a union of $m = 4$ boxes, outlined in bold. The classification of each two-dimensional region is shown inside the region. (The classifications of the one-dimensional and zero-dimensional regions are not shown but are stored in the prediction matrix A.) The boundary of the target concept is segmented to illustrate a new complexity measure introduced in section 9.*

$\{T_1, \ldots, T_d\}$. Thus $G_{\mathcal{H}} = R_1 \times R_2 \times \cdots \times R_d$. For $r \in G_{\mathcal{H}}$, corresponding to nodes $v_i \in T_i$, we shall refer to the point $(\min^1_{v_1}, \ldots, \min^d_{v_d})$ as the *lower corner* of r and $(\max^1_{v_1}, \ldots, \max^d_{v_d})$ as the *upper corner* of r.

In addition to the trees $T_1, \ldots, T_d$, our hypothesis also maintains a prediction array A with $|G_{\mathcal{H}}|$ entries where, for $r \in G_{\mathcal{H}}$, $A[r]$ gives the classification for region r. For a consistent region (i.e., all known points have the same classification) A will contain either a 0 (for negative) or a 1 (for positive). However, for regions in which there is an inconsistency, there will be a pointer into a queue Q of inconsistent regions. In addition, for each region $r \in Q$ we store points y_+ and y_- giving a pair of inconsistent points in r. We use $h(G_{\mathcal{H}}, A)$ to denote the hypothesis defined by the regions in $G_{\mathcal{H}}$ with the classifications given in A. Note that the hypothesis is well-defined only if all regions are consistent. Figure 1 shows the set of regions defined by a target concept once all hyperplanes are discovered, and the classifications of all of the regions (as stored in A).

Given a hypothesis $h(G_{\mathcal{H}}, A)$ (we sometimes denote this simply as h) for which the queue Q of inconsistent regions is empty, and a point y, we can compute $h(y)$, the prediction made by hypothesis h on point y, as follows. For $1 \leq i \leq d$ we perform a search for y_i in tree T_i to find the node having y_i in its range. Combining the ranges of the d nodes found defines the region $r \in G_{\mathcal{H}}$ that contains y. Finally $h(y) = A[j_1, \ldots, j_d]$, where j_i is contained in the node found in T_i during the search for y and denotes the dimension i index for A corresponding to the region that contains y. Then combining the indices obtained from each of the trees provides an index into A. Since we know an upper bound on the number of nodes in each tree, we can initially allocate enough space for A.

5.1. The membership and equivalence query algorithm. Our algorithm works by repeatedly building a consistent hypothesis that incorporates all known halfspaces. The hypothesis is represented as a decision tree rather than as a union of boxes; subsequently we show how to express it as a union of (not too many) boxes.

Given an inconsistent hypothesis $h(G_{\mathcal{H}}, A)$ and the queue Q of inconsistent regions from $G_{\mathcal{H}}$, we refine $h(G_{\mathcal{H}}, A)$ so that it is consistent using the following procedure that uses membership queries to find *new* hyperplanes with which to modify the hypothesis. We never remove any hyperplane from $\mathcal{H}$ and search for a new hyperplane only in such a way that we are certain that an existing hyperplane will not be rediscovered in the process. We also maintain the invariant that Q always contains exactly one entry for each inconsistent region of $G_{\mathcal{H}}$. Note that we never make an equivalence query using an inconsistent hypothesis.

Our procedure to build a consistent hypothesis repeatedly does the following until Q is empty (and thus the hypothesis is consistent). Let r be the region at the front of the queue. Since r is not a consistent region, we know that there must be some unknown hyperplane of g that goes between y_+ (a known positive point in r) and y_- (a known negative point in r). Thus we can perform a binary search between y_+ and y_- (where the comparisons are replaced by membership queries) to find a $+/-$ pair contained within region r using only $d + \lceil \log n \rceil$ membership queries. We use $\lceil \log n \rceil$ queries to find a positive point and a negative point whose positions differ by at most one in each coordinate by performing a single binary search on all d dimensions. (Query points are chosen by bisecting the range in each dimension; noninteger coordinates may be rounded arbitrarily.) We then use d additional queries to determine the slope of the separating hyperplane. Furthermore, the hyperplane defined by this $+/-$ pair is guaranteed to be a hyperplane that has not yet been discovered (by the definition of a region).

The full details of procedure ADD-HYPERPLANE, which modifies $h(G_{\mathcal{H}}, A)$ to incorporate the new hyperplane found, is given in Figure 2. The learner begins by using a standard tree insertion procedure to insert the new hyperplane into the search tree for the appropriate dimension. Then the set of regions that have been split by the hyperplane are deleted from $\mathcal{H}$, and each is replaced by three new regions (one of them being a degenerate region corresponding to the hyperplane itself). For each new region of r we make a membership query on the lower and/or upper opposing corners if those queries have not already been made. As we shall discuss in section 6, this step can be replaced by just using 0 as the default value for $A[r]$. If the classifications of these two corners are the same, then the classification is entered in $A[r]$; otherwise the region is placed in Q with these corners used for y_+ and y_-.

Our algorithm LEARN-WITH-MQS works as follows. For ease of exposition we artificially extend the instance space from $\mathcal{N}_n^d$ to $\{0, 1, \ldots, n, n+1\}^d$, where it is known a priori that any example with a coordinate of 0 or $n+1$ in any dimension is a negative example. (The pseudocode does not explicitly make this check, but one could imagine replacing the calls to MQ with a procedure that first checks for such cases.) Initially, $G_{\mathcal{H}}$ just contains the single region corresponding to the entire instance space. Since the upper and lower corners of this region are negative, the initial hypothesis predicts 0 for all instances.

We then repeat the following process until a successful equivalence query is made. Let y be the counterexample received from an equivalence query made with a consistent hypothesis. Using membership queries (in the form of a binary search) we can find two new hyperplanes of the target concept. Without loss of generality, we assume that y is a positive counterexample in region r of $G_{\mathcal{H}}$. Since the hypothesis was consistent and y is a positive counterexample, we know that the upper and lower corners of r are classified as negative. Thus we can use these corners of r (with y) as the endpoints for binary searches to discover two new hyperplanes. We find a hyper-

```
ADD-HYPERPLANE(h(G_H, A), Q, i, c)
 Let v be the leaf of T_i for which min_v^i ≤ c ≤ max_v^i
 Using a standard balanced tree insertion procedure, update T_i so that
     v is an internal node with key c
     v has a left child with range [min_v^i, c − 1]
     v has a right child with range [c + 1, max_v^i]
 Let R_delete = R_1 × ··· × R_{i−1} × {[min_v^i, max_v^i]} × R_{i+1} × ··· × R_d
 Let R_add = R_1 × ··· × R_{i−1} × {[min_v^i, c − 1], [c, c], [c + 1, max_v^i]} × R_{i+1} × ··· × R_d

 For each r ∈ R_delete
     Let r_=, r_<, and r_> be the regions in R_add for which (r_= ∪ r_< ∪ r_>) = r
     If A[r] = b (for b = 0 or b = 1)
         Let A[r_=] = A[r_<] = A[r_>] = b
     Else (so A[r] is a pointer to element q of Q)
         Generate new queue nodes for r_=, r_<, and r_>
         Set the corresponding entries of A to point to these new nodes
         Copy y_− and y_+ in the appropriate queue entries for r_< and r_>
         Remove q from Q
     For each r' ∈ {r_=, r_<, r_>}
         Let y_u (respectively, y_ℓ) be the upper (respectively, lower) corner of r'
         Make a membership query to determine y_u and y_ℓ if not already known
         If g(y_u) = g(y_ℓ) then let A[r'] = g(y_u)
         Else (r' is inconsistent)
             Assign y_u and y_ℓ to y_− and y_+ as appropriate
             Q.ENQUEUE(r')
```

FIG. 2. *Our subroutine to update $h(G_{\mathcal{H}}, A)$ to incorporate the newly discovered hyperplane $x_i = c$. The new hyperplane is added to tree T_i. Then all regions in $G_{\mathcal{H}}$ that are split are removed from Q. Finally this procedure initializes the new entries of $G_{\mathcal{H}}$ in the prediction matrix A. Note that in the for loop that adds new regions, we could choose not to perform membership queries on the upper and lower corners of r, as is done in section 6. Instead, we could just use the known points in $r_=$, $r_<$, and $r_>$ to determine their labels (with 0 as a default if there are no points in the region). The advantage of doing this step is that the number of equivalence queries is reduced by a factor of 2 since two hyperplanes of the target are guaranteed to be found for each counterexample (Lemma 1). If we find that the opposing corners have different classifications then we place that newly created region on Q, which in turn will cause a binary search to be done to find a new hyperplane and split that region further.*

plane by doing a binary search until we find a $+/-$ pair (two points that are labeled differently and differ by at most one in each dimension). We know that a side of a target box must pass between these two points (or through one of the points). We then use d additional queries to discover the slope of this hyperplane. The hypothesis is updated using ADD-HYPERPLANE to incorporate these two hyperplanes. Finally, we call MAKE-CONSISTENT-HYPOTHESIS to refine any inconsistent regions. Figure 3 gives the complete algorithm.

5.2. Analysis. We now analyze the time and query complexity of LEARN-WITH-MQS. As part of this analysis we use the following lemma.

LEMMA 1. *Every counterexample can be used to discover at least two distinct new hyperplanes of the target concept.*

Proof. Let y be the counterexample and $r \in G_{\mathcal{H}}$ be the region containing y. Since $h(G_{\mathcal{H}}, A)$ is a consistent hypothesis, we know that the upper and lower corners of r

```
LEARN-WITH-MQS:
 Let Q ← ∅
 Let (0, ..., 0) and (n + 1, ..., n + 1) be negative corners of single region r
 For 1 ≤ i ≤ d
     Initialize T_i to be a single leaf covering the range 0 to n + 1
 For r the single region of G_H, let A[r] = 0

 While Equiv(h(G_H, A)) ≠ "yes"
     Let y be the counterexample where y is in region r of h(G_H, A)
     Let z_1 and z_2 be the lower and upper opposing corners of r
     For 1 ≤ ℓ ≤ 2
         Perform binary search between y and z_ℓ to find hyperplane x_i = c
         ADD-HYPERPLANE(h(G_H, A), Q, i, c)
     h(G_H, A) ←MAKE-CONSISTENT-HYPOTHESIS(h(G_H, A), Q)
 Return h(G_H, A)

MAKE-CONSISTENT-HYPOTHESIS(h(G_H, A), Q)
 While Q ≠ ∅
     r ← Q.DEQUEUE
     Perform binary search between y_− and y_+ from r
        to find the hyperplane x_i = c
     ADD-HYPERPLANE(h(G_H, A), Q, i, c)
```

FIG. 3. *Algorithm for learning unions of d-dimensional axis-parallel boxes using membership and equivalence queries. Note that i, in the calls to* ADD-HYPERPLANE*, is the dimension separated by the hyperplane found in the binary search.*

are classified opposite y and all points in r are classified opposite y by the hypothesis. Since a positive point and a negative point must be separated by some hyperplane of the target concept, searches between y and each of the upper and lower corners will find some $+/-$ pair. These will be distinct since the two searches move away from each other in all dimensions. □

We now prove that our first algorithm has the stated complexity.

THEOREM 1. *Given any* $g \in \bigcup_{\leq m} \text{BOX}_n^d$, LEARN-WITH-MQS *achieves exact identification of* g *making at most* $md + 1$ *equivalence queries, and using* $O((4m)^d + md(\log n + d))$ *time and membership queries.*

Proof. The correctness of LEARN-WITH-MQS is trivial. Since the algorithm only returns a hypothesis $h(G_{\mathcal{H}}, A)$ for which Equiv$(h(G_{\mathcal{H}}, A))$ returns "yes," the algorithm is correct upon returning a hypothesis.

We now analyze the query and time complexity of LEARN-WITH-MQS. Recall that since there are only m boxes in the target concept, there are at most $2md$ hyperplanes in the final hypothesis. Clearly, any box can be subdivided into a union of smaller boxes, unnecessarily increasing the complexity of the target. However, our algorithm adds hyperplanes only where there is evidence for the existence of a side of a target box (a $+/-$ pair) and, therefore, does not *unnecessarily* subdivide boxes (subdividing does occur due to extending hyperplanes through the entire domain, but this is already accounted for in our analysis). Furthermore, since no hyperplane is ever rediscovered and every binary search (which uses $O(\log n + d)$ membership queries) discovers a hyperplane, we know that $O(md(\log n + d))$ membership queries are used during all of the binary searches made by the algorithm. Also, as given in (5.1),

there are at most $(4m+1)^d$ regions in the final hypothesis, and thus the number of membership queries used for querying the upper and lower opposing corners is at most $2 \cdot (4m+1)^d = O((4m)^d)$. Since these are the only two places in which membership queries are performed, the total number of membership queries made by our algorithm is $O((4m)^d + md(\log n + d))$.

From Lemma 1 we know that each counterexample enables Learn-With-MQs to find at least two new, distinct hyperplanes of the target concept. Since there are at most $2md$ hyperplanes comprising the m boxes, at most md counterexamples can be received and thus at most $md+1$ equivalence queries are made.

The time needed to evaluate $h(G_{\mathcal{H}}, A)(x)$ for an unlabeled example y is $O(d \log m)$ since the key operation is performing d searches in balanced search trees of depth $O(\log m)$. Thus, the time complexity of this algorithm is found to be $O((4m)^d + md(\log n + d))$. □

Finally, it is easily seen that this algorithm extends to learning any discretized geometric concept generated by hyperplanes with slopes from $S = \{e_i\}$ (the standard basis) while receiving at most $C_S(g)/2$ counterexamples and using time and membership queries polynomial in $C_S(g)$ and $\log n$ for d any constant.

5.3. Using a hypothesis class of unions of boxes. We now describe how a consistent hypothesis can be converted to the union of $O(md \log m)$ boxes from BOX_n^d. Since all equivalence queries are made with consistent hypotheses, such a conversion enables our algorithm to learn the union of m boxes from BOX_n^d using as a hypothesis class the union of $O(md \log m)$ boxes from BOX_n^d. Note that it is NP-hard to find a minimum covering of a concept from $\bigcup_{\leq m} \text{BOX}_n^d$ by individual boxes [19].

Recall that a consistent hypothesis h essentially encodes the set of positive regions. Thus our goal is to find the union of as few boxes as possible that "cover" all of the positive regions. We now describe how to formulate this problem as a set covering problem for which we can then use the standard greedy set covering heuristic [18] to perform the conversion. The set X of objects to cover will simply contain all positive regions in h. Thus $|X| \leq (4m+1)^d$. Then the set $\mathcal{F}$ of subsets of X will be made as follows. Consider the set B of boxes where each box in B is formed by picking a minimum and maximum coordinate in each dimension from the hyperplanes represented in h for that dimension. For any $b \in B$, if b contains any negative region, then throw it out. Otherwise, place in $\mathcal{F}$ the set of regions contained within b. Thus $|\mathcal{F}| \leq (2m)^{2d}$ since there are at most $2m$ values in each dimension that can form the two sides of the box. Furthermore, $\mathcal{F}$ contains a subset of size m that covers all items in X. Finally, we can apply the greedy set covering heuristic to find a set of at most $m(\ln |X| + 1) = m(d \ln(4m+1) + 1) = O(md \log m)$ boxes that cover all positive regions. The time to perform the conversion is $O((2m)^{2d} md \log m)$. Thus, since at most $md+1$ equivalence queries are made, the total time spent in converting the internal hypotheses into hypotheses that are unions of boxes is at most $O((md)^2 \log m \cdot (2m)^{2d})$.

6. Learning unions of boxes using only equivalence queries. We now describe a simple method to remove the use of membership queries in Learn-With-MQs. First observe that the use of membership queries in this algorithm can easily be reduced to only their use within the binary searches. Instead of querying opposing corners of new regions created, we can instead use the classification of the single known point or otherwise a default of negative for the classification of the region. Then the counterexamples from the equivalence queries can be used to obtain a positive and negative point in the region that can be used for the binary search. (Of course, this

modification dramatically increases the number of equivalence queries used.)

Now to remove the use of membership queries in a binary search between y_+ and y_- we simply take the midpoint between y_+ and y_- (i.e., the first point on which a membership query would be made) and insert a hyperplane going through that point for each of the d dimensions. There are $2dm$ hyperplanes defining the target. We require at most $2\lceil \log n \rceil$ counterexamples (that cause $\mathcal{H}$ to be modified) to find each hyperplane. Thus, the total number of counterexamples required (that cause $\mathcal{H}$ to be modified) is at most $4dm\lceil \log n \rceil$. For each such counterexample (i.e., one that causes $\mathcal{H}$ to be modified) we insert d hyperplanes in the hypothesis. Thus the number of hyperplanes in the hypothesis is at most $4d^2m\lceil \log n \rceil$. By (5.1) there are at most $(8d^2m\lceil \log n \rceil + 1)^d$ regions in the final hypothesis. There is at most one equivalence query made for each of the hyperplanes found and at most one equivalence query made for each region. Thus we obtain the following result.

THEOREM 2. *Given any $g \in \bigcup_{\leq m} \text{BOX}_n^d$, there is an algorithm that achieves exact identification of g using $O((8d^2m \log n)^d)$ equivalence queries (and* no *membership queries). The time complexity is $O((8d^2m \log n)^d)$.*

It is easily seen that for d constant this algorithm exactly learns $\bigcup_{\leq m} \text{BOX}_n^d$ *using only equivalence queries* with both time and the number of equivalence queries polynomial in m and $\log n$.

7. Extending S to arbitrary known slopes. We now present a modification of the equivalence query algorithm described in section 6 that handles the situation in which S can be an arbitrary set of known slopes versus just being the standard basis. We let s denote the number of distinct slopes in S (i.e., $s = |S|$).

Let $S = \{a_1, \ldots, a_s\} \subset \mathcal{Z}^d$ be a set of slopes. Thus $a_i = (a_{i1}, a_{i2}, \ldots, a_{id})$ (for $1 \leq i \leq s$) defines a slope for a d-dimensional hyperplane, that is, a hyperplane of the form $\sum_{j=1}^{d} a_{ij}x_j = b$ (or equivalently $a_i x^T = b$) for any real constant b. Let $B = \{B_1, \ldots, B_s\}$, where $B_i \subset \mathcal{R}$ defines a set of $|B_i|$ possible values for the constant term "b" for those hyperplanes with slope given by a_i. Thus together S and B define a set of hyperplanes

$$\mathcal{H} = \{a_i x^T = b \mid x = (x_1, \ldots, x_d),\ i = 1, \ldots, s,\ b \in B_i\}.$$

The hypothesis $h = h(G_{\mathcal{H}}, A)$ is that which classifies each $x \in \mathcal{N}_n^d$ according to the unique classification of all known points (if any) in the region of $G_{\mathcal{H}}$ containing x. If there are no known points in the region containing x, then negative is used as the default.

As in our basic equivalence query algorithm, this algorithm begins with the entire region classified as negative. After the first two equivalence queries are made (the first with $h = \emptyset$ and the second with $h = \mathcal{N}_n^d$), the algorithm will have a positive counterexample y and a negative counterexample u. Thus the straight line between y and u must intersect one of the hyperplanes that define g. Let $v = (y + u)/2$ be the midpoint of the line between y and u. Without the ability to make membership queries it is not possible to find a $+/-$ pair. Furthermore, even if we could find a $+/-$ pair, we would not be able to determine the slope of the hyperplane that created that $+/-$ pair. As in the previous section, we address this problem by adding to our set of hyperplanes $\mathcal{H}$ a hyperplane passing through the midpoint v for *each* slope in S. We repeatedly use this process until an equivalence query made with $h(G_{\mathcal{H}}, A)$ is correct.

For ease of exposition, in this section we will not discuss the details of how to represent $\mathcal{H}$ so that $h(G_{\mathcal{H}}, A)$ can be efficiently evaluated (in terms of all parameters).

```
LEARN-GENERAL-SLOPES(S = {a_1, ..., a_s})
 H ← ∅
 Let r be the single region of G_H, let A[r] = 0
 While Equiv(h(G_H, A)) ≠ "yes"
     Let y be the counterexample where y is in region r of h(G_H, A)
     A[r] = 1 − h(G_H, A)(y)
     If there is a known point u in r classified opposite y
         v = (y + u)/2
         For i = 1 to s
             H ← H ⋃ (a_i x^T = b = (a_i v^T))
         Update A to incorporate the new regions created
 Return h(G_H, A)
```

FIG. 4. *Algorithm for exactly learning a discretized geometric concept defined by slopes in S using only equivalence queries.*

However, the technique of section 5 of using s balanced search trees, one for each element of S, generalizes in an obvious manner.

Our algorithm, at a high level, is shown in Figure 4. In this algorithm, LEARN-GENERAL-SLOPES, S is the set of the possible slopes of the hyperplanes. We initialize $\mathcal{H}$ to be the empty set and the classification of the single region to be 0. (And thus the initial hypothesis, $h(G_{\mathcal{H}}, A)$, is simply the always false hypothesis.) We ask the equivalence query $h(G_{\mathcal{H}}, A)$ and use the counterexample y to update A. If this counterexample is the first counterexample in its region then we just update A. Otherwise, if this counterexample is in some region for which we have already seen a point u, then y and u have different classifications in g and the line that passes through y and u must intersect a defining hyperplane of g. We then define $v = (y + u)/2$ and add all possible hyperplanes that pass through v to the set $\mathcal{H}$. We then repeat this process until $h(G_{\mathcal{H}}, A)$ is logically equivalent to g.

7.1. Analysis. We now prove the correctness of our algorithm and analyze its complexity. We use the following lemma to bound the maximum number of regions that will be contained in $G_{\mathcal{H}}$.

LEMMA 2. *Any t d-dimensional hyperplanes in a $(d+1)$-dimensional space divide the space into at most $t^{d+1} + 1$ regions.*

This result is well known. See Edelsbrunner [20] for a proof. We are now ready to analyze our algorithm LEARN-GENERAL-SLOPES.

THEOREM 3. *Let S be a set of slopes. Then* LEARN-GENERAL-SLOPES *exactly learns any target concept g from the class of discretized geometric concepts generated from hyperplanes with slopes from S using time and equivalence queries polynomial in $\|S\|, C_S(g)$, and $\log n$ for any constant d.*

Proof. The correctness follows trivially since the algorithm returns only a hypothesis $h(G_{\mathcal{H}}, A)$ for which Equiv$(h(G_{\mathcal{H}}, A))$ returns "yes," hence the algorithm is correct upon returning a hypothesis.

We now analyze the query and time complexity. Let $m = C_S(g)$. Recall that $\|S\|$ denotes the sum of the logarithms of the absolute values of the integers in S. Thus, by the definition of $\|S\|$, any $a_i \in S$ can be represented using at most $\|S\|$ bits. Thus there are at most $2^{\|S\|}$ possible values for a slope. Also, any example $x \in \mathcal{N}_n^d = \{1, \ldots, n\}^d$ and thus has n^d possible values. Let $H_1, \ldots, H_m$ be a minimum-size set of hyperplanes

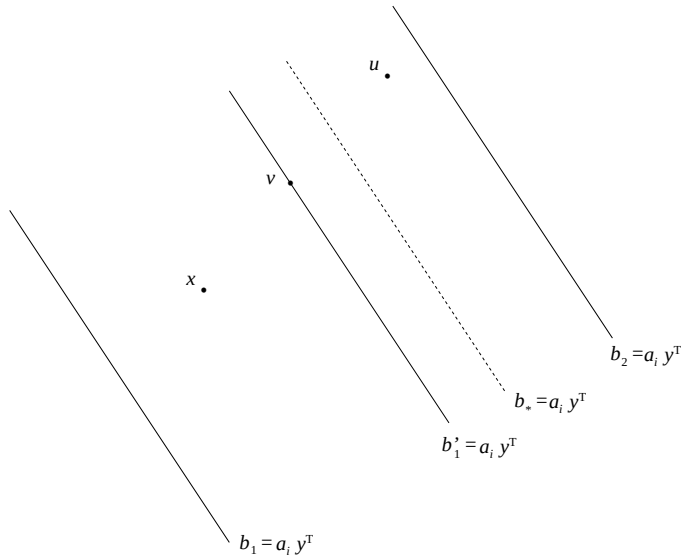


FIG. 5. *Here y is a counterexample to the current hypothesis and $b_1 = a_i x^T$ and $b_2 = a_i x^T$ are the nearest hyperplanes. The hyperplane $b'_1 = a_i x^T$ is added to $\mathcal{H}$ by* LEARN-GENERAL-SLOPES.

with slopes from S that generate the boundary of the geometric concept g. Since each hyperplane defining g is of the form $b = a_i x^T$, we have that the maximum number of values that any b can have is

$$\|b_i\| \leq \|a_i x^T\| \leq 2^{\|S\|} \cdot n^d = \gamma.$$

At any time during execution of our algorithm, for each of $H_1, \ldots, H_m$, there are two closest hyperplanes in $\mathcal{H}$ (one for each side) with the given slope. (At the beginning of execution, imagine two hyperplanes with each slope outside the domain and anchored at opposing corners of the domain.) We now show that for every counterexample that causes $\mathcal{H}$ to be modified there exists some H_i $(i = 1, \ldots, m)$ for which the distance between it and one of its closest hyperplanes is reduced by at least a factor of 2.

Suppose that y is a counterexample to the current hypothesis $h(G_{\mathcal{H}}, A)$ and that the region r of $G_{\mathcal{H}}$ that contains y already contains the point u (otherwise $\mathcal{H}$ is not modified). By the definition of $h(G_{\mathcal{H}}, A)$ it follows that $g(u) \neq g(y)$. Therefore the line segment between y and u must intersect some hyperplane—say, $H_* \equiv (a_i x^T = b_*)$—of g for $a_i \in S$ and $b_* \in \mathcal{Z}$. Let $b_1 = a_i x^T$ and $b_2 = a_i x^T$ be the two hyperplanes with slope a_i nearest H_* in $\mathcal{H}$. Let $v = (y + u)/2$ be the midpoint between y and u. Without loss of generality we assume that the hyperplane $b'_1 = a_i x^T$ that passes through v with slope a_i is between $b_1 = a_i x^T$ and $b_* = a_i x^T$ as illustrated in Figure 5. We denote the hyperplane that passes through y (respectively, u) with slope a_i by $b_y = a_i x^T$ (respectively, $b_u = a_i x^T$). Thus we have that $b_1 \leq b_y \leq b'_1 \leq b_* \leq b_u \leq b_2$.

Let $\Delta = b_* - b_1$. (Thus Δ is proportional to the distance between H_* and $b_1 = a_i x^T$.) Observe that LEARN-GENERAL-SLOPES will add the hyperplane $b'_1 = a_i x^T$ to $\mathcal{H}$, and this will now replace $b_1 = a_i x^T$ as a closest hyperplane of slope a_i to H_*. Let $\Delta' = b_* - b'_1$. (Thus Δ' is proportional to the new distance between H_* and its closest hyperplane in $\mathcal{H}$ in that direction.) We now show that $\Delta' \leq \Delta/2$. Observe that

$$\Delta' = b_* - b'_1$$

$$
\begin{aligned}
&= b_* - \left(\frac{(b_u + b_y)}{2}\right) \\
&\leq b_* - \left(\frac{(b_* + b_1)}{2}\right) \\
&= \frac{b_* - b_1}{2} = \frac{\Delta}{2}
\end{aligned}
$$

Thus the distance between H_* and one of its two nearest hyperplanes is reduced by a factor of two. Finally, when the distance between H_* and both of its two nearest hyperplanes is less than 1, the algorithm has determined the discretized hyperplane H_* and no other hyperplanes will be added for H_*.

Since the distance between each of $H_1, \ldots, H_m$ with both of its closest hyperplanes in $\mathcal{H}$ is at most γ it follows that the number of counterexamples needed to find one hyperplane is

$$2\lceil \log(\gamma) \rceil = 2\lceil \log(2^{\|S\|} \cdot n^d) \rceil = O(\|S\| + d \log n).$$

The number of hyperplanes is $m = C_S(g)$, and at each iteration we add $s = |S| \leq \|S\|$ hyperplanes to the hypothesis. Therefore the number of hyperplanes generated by our algorithm is $O(ms(\|S\| + d \log n))$. Thus by Lemma 2 the number of regions of the hypothesis is

$$O(ms(\|S\| + d \log n))^d.$$

The number of iterations that do not add any hyperplanes is bounded by the number of regions; thus the number of equivalence queries made by Learn-General-Slopes is

$$O(ms(\|S\| + d \log n))^d,$$

and clearly the time complexity is also polynomial in m, $\|S\|$, and $\log n$ for any constant d as desired. □

Finally, we note that since any slope in the discretized space $\mathcal{N}_n^d$ can be defined by two points from the domain, there are at most n^{2d} possible values for any one of the s slopes and thus $\|S\| \leq 2ds \log n$.

8. Handling lies in the counterexamples. In this section we consider the case in which the learner is provided with an l-liar teacher. Recall that an l-liar teacher is a teacher that can provide incorrect counterexamples as answers to at most l equivalence queries. This noise is not persistent. That is, if the teacher provides the same incorrect counterexample twice, then it counts as two lies (even if both were in response to the same query). Note that our learner in this model is not given access to a membership oracle. Thus, to learn the true classification of the points about which the environment has lied, it is necessary for the learner to isolate these instances. That is, the learner must create a region that consists of the single point that was the counterexample. This will force the teacher to give, eventually (after its l lies are exhausted), a correct classification for this instance if it is not already correctly classified by the hypothesis.

A degenerate region (a region consisting of fewer than d dimensions) is created whenever our algorithm adds to the hypothesis a hyperplane, passing through a given point, for each slope in S. We want to isolate the counterexample on which the lie occurred in a region of dimension 0. Notice, however, that given some set of

slopes, it is possible that the hyperplanes defined by the slopes, passing through the counterexample, do not create a 0-dimensional region.

Therefore, to ensure that the 0-dimensional region is created, we include in S the slopes of the elementary vectors $\{e_i\}$. Then the faulty points will be bounded by hyperplanes and at the end the oracle must give their real values. Thus, the main difference between our algorithm in this model and in the noise free model is the addition of hyperplanes through the counterexamples. Through the midpoints of $+/-$ pairs we still add only $|S|$ hyperplanes. It is only through the counterexamples that we add $|S| + d$ hyperplanes. This changes $C_S(g)$ to $C_S(g) + dl$, the number of slopes to $|S| + d$, and the complexity to

$$O((C_S(g) + dl)(|S| + d)(\|S\| + d + d \log n))^d.$$

9. A return to learning unions of boxes. Observe that by extending the hyperplane defined by a $+/-$ pair across the entire domain, our initial algorithm Learn-With-MQs may unnecessarily split a consistent region into a large number of smaller regions all of which make the same prediction. The algorithm we present here is motivated by the goal of reducing this unnecessary splitting by splitting only the region in which the counterexample is contained. We show that this algorithm runs in polynomial time for *either* m or d constant.

We begin by examining how one might measure the complexity of a concept from $\bigcup_{\leq m} \text{BOX}_n^d$. Observe that the minimum number m of boxes used to form the target concept is not a good measure of the complexity of the target concept. For example, consider the two examples shown in Figure 6. While both targets are composed of six boxes, the first is clearly more complex than the second. Thus the complexity of an algorithm should depend on some quantity other than just the number of boxes and dimension of the target concept. We now introduce such a new complexity measure, σ, to better capture the complexity of the target concept. We define a *segment* of the target concept g as a *maximal portion* of one of the sides of g that lies either entirely inside or entirely outside of each of the other halfspaces defining the target. (A halfspace defines the target if it labels some $+/-$ pair consistently with the labeling given by the target. See the definition of $+/-$ pair in Definition 1.) Note that two adjacent segments will intersect in a region of dimensionality less than the segments. We observe that there exist other possible measures of complexity besides σ that are potentially worthy of study. One example is the number of sides of a target polyhedron, where a side is a maximal connected set of coplanar boundary points of the polyhedron such that the interior of the polyhedron is on the same side of the set of boundary points. Sides (as opposed to segments) have the drawback of being of varying complexity (at least for polyhedra of more than two dimensions), as well as being apparently harder to work with. That is, we can have $0, 1, \ldots, d-1$ dimensional sides.

For example the target concept shown in Figure 1 has 14 sides. The same target concept has 22 segments. (The heavy black line defining the target concept is split into 20 clearly visible segments by the thin lines that correspond to the halfspaces that define the target concept. The final 2 segments come from the fact that the points $(20, 52)$ and $(49, 52)$ are segments due to the halfspaces $y_2 \geq 52$ and $y_2 \leq 52$.) The polygon in Figure 6(a) has 60 segments, since each of the long edges of the 6 boxes contributes 4 segments. By contrast the polygon in Figure 6(b) (defined by the same number of boxes) has 24 segments, since none of the sides of the individual boxes straddles any of the hyperplanes defining the boxes.

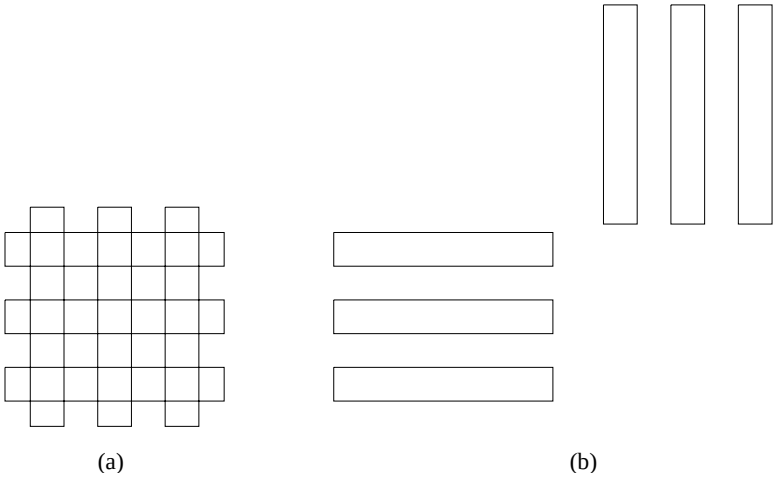


FIG. 6. *These concepts illustrate why "number of boxes" is an inadequate measure of complexity.* (a) *is more complex than* (b) *because of the large number of intersections of the sides of the boxes. For* (a) $\sigma = 60$. *For* (b) $\sigma = 24$.

For a concept $g \in \bigcup_{\leq m} \text{BOX}_n^d$ but not in $\bigcup_{\leq m-1} \text{BOX}_n^d$, we let σ denote the number of segments in the target concept corresponding to g. (In other words, suppose that m is the minimum number of boxes whose union is g.) First, observe that $m \leq \sigma$ since none of the boxes are contained within the union of the other $m-1$ boxes (which follows from the minimality of m). Furthermore, observe that $\sigma \leq (2m)^d$ since there are $2m$ halfspaces per dimension (one corresponding to each of the $2m$ sides per dimension) that can intersect to form at most $(2m)^d$ connected regions each of which would be part of at most one segment. Finally, it then follows that for any $g \in \bigcup_{\leq m} \text{BOX}_n^d$, the number of segments in $g \leq (2m)^d$ since the minimum number of boxes whose union is g is at most m.

The hypothesis class we use in this algorithm is a decision tree over the halfspaces defining the target concept. Namely, each hypothesis T is a rooted binary tree where each internal node is labeled with a halfspace and in which leaves are labeled from $\{0, 1\}$. We evaluate T recursively by starting at the root and evaluating the left subtree if the root's halfspace does not contain the point, and the right subtree otherwise. When a leaf is reached its label is output. Observe that each node of T corresponds to a subregion of the domain, with the root corresponding to the entire domain. The halfspace H associated with each internal node divides its region r into two subregions, with the left child being the subregion given by $\overline{H} \cap r$ and the right child being the subregion given by $H \cap r$. The leaves correspond to a set of nonoverlapping boxes that cover the entire region where the label for a given region is given by the label for the corresponding leaf. In Observation 1 we show that the height of the final decision tree will be at most $2md$. Thus the hypothesis can be evaluated in time polynomial in both m and d.

We now describe our algorithm. It has complexity polynomial in σ and $\log n$ as well as being an efficient algorithm for either m or d constant. We initialize T to be a single 0 leaf node. (Again we implicitly use the instance space $\{0, 1, \ldots, n, n+1\}^d$.) When a counterexample is received, we first search T to find the leaf v containing it. Let r be the subregion corresponding to v. Then as in LEARN-WITH-MQS we use a binary search to find a $+/-$ pair contained in r that defines a halfspace H. We

```
ALT-LEARN-WITH-MQS
 Initialize T to be the single 0-leaf
 While Equiv(T) ≠ "yes"
        Let y be the counterexample
        Search in T to find the leaf v corresponding to the region containing y
        T ←SPLIT-REGION(T, v, y)
 Return T

SPLIT-REGION(T, v, y)
▷T is the decision tree that defines the current hypothesis
▷y is a counterexample in a region r defined by T
▷v is the leaf of T that corresponds to r
 Perform binary search between y and a corner of region r
 Let H be the hyperplane found
 Let v_L and v_R correspond to the new regions created
 Make v an internal node labeled with H and having left child v_L and right child v_R
 Let r_L be the region H̄ ∩ r corresponding to v_L
 Let r_R be the region H ∩ r corresponding to v_R
 For each r' ∈ {r_L, r_R}
     Let v' be the leaf corresponding to region r'
     Perform a membership query on the upper and lower opposing corners of r'
     Call these corners upper and lower
     If both corners have classification b ∈ {0, 1} let v' be a b-leaf of T
         Let the predicted label of v' become b
     Else
         Let v' be a leaf of T with v' having the same classification as lower
         SPLIT-REGION(T, v', upper)
```

FIG. 7. *Alternate algorithm for learning unions of d-dimensional axis-parallel boxes that runs in polynomial time for* either *m or d constant. We note that the corner points of regions in the hypothesis, used throughout the algorithm, are easy to compute given the hypothesis.*

replace v with an internal node labeled with H, having left child leaf v_L corresponding to the region given by $\overline{H} \cap r$ and right child leaf v_R corresponding to the region given by $H \cap r$. At this point we call a procedure that recursively visits all newly created leaves in a depth-first manner and checks if the corresponding region is a consistent region. If the region r' associated with leaf v' is consistent (a region is consistent if its opposing corners have the same classification), then the classification field is filled; otherwise we use a binary search to obtain a halfspace H' for r' and replace v' with an internal node labeled with H'. We generate two new leaves: v'_L corresponding to the region $\overline{H'} \cap r'$ and v'_R corresponding to the region $H' \cap r'$. Then a recursive call is made to validate (if necessary) each of these new regions. The algorithm is shown in Figure 7. One possible final hypothesis that could be constructed by this algorithm, for the target concept shown in Figure 1, is shown in Figure 8. Figure 9 shows the decomposition of $\mathcal{N}_n^d$ that corresponds to the decision tree shown in Figure 8.

9.1. Analysis. We first show that the height of the final decision tree comprising our hypothesis is at most $2md$.

OBSERVATION 1. *The height of the final decision tree constructed by* ALT-LEARN-WITH-MQS (*shown in Figure* 7) *is at most* $2md$ *since each of the at most* $2md$ *halfspaces defining the target polyhedra can appear at most once on any path from*

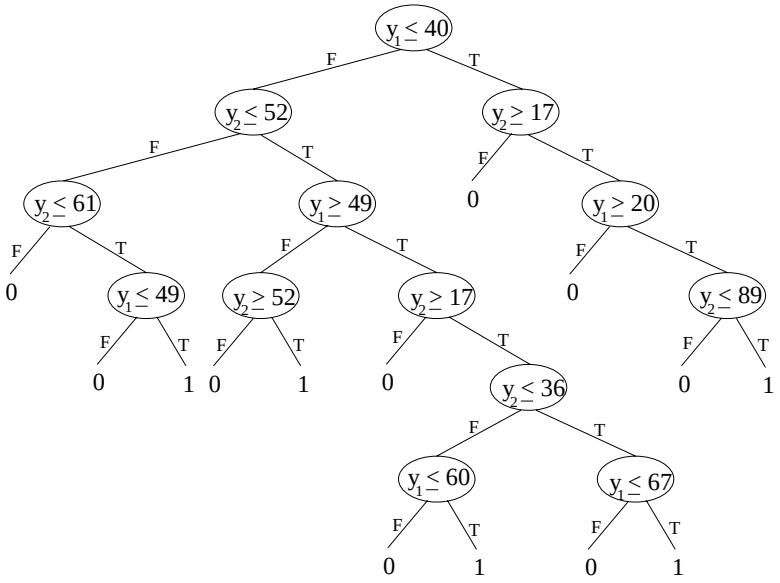


FIG. 8. *This shows the final decision tree that could be constructed by* ALT-LEARN-WITH-MQS *for the target geometric concept shown in Figure* 1.

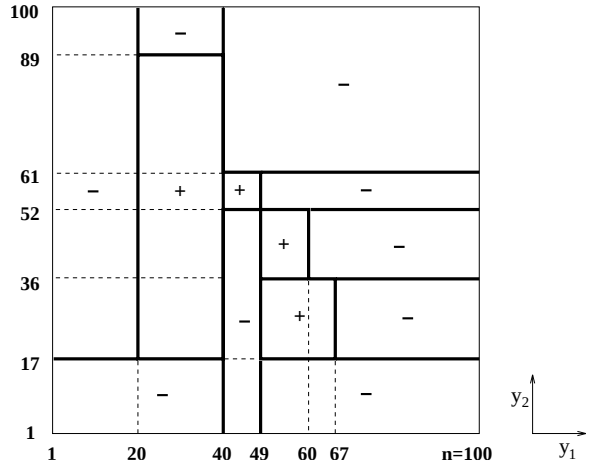


FIG. 9. *This shows the decomposition of* $\mathcal{N}_n^d$ *corresponding to the decision tree shown in Figure* 8.

the root to any leaf.

We now give two separate techniques for analyzing this algorithm. The first method of analysis gives that this algorithm uses queries and time polynomial in σ and $\log n$ (and thus polynomial in m and $\log n$ for d constant). The second method of analysis shows that our algorithm uses queries and time polynomial in d and $\log n$ for m any constant.

THEOREM 4. *Given any target concept* $g \in \bigcup_{\leq m} \text{BOX}_n^d$, *the algorithm* ALT-LEARN-WITH-MQS *achieves exact identification of* g *making at most* $(\sigma/2+1)$ *equivalence queries, and using* $O(\sigma(\sigma + \log n + d))$ *time and* $O(\sigma(\log n + d))$ *membership queries.*

Proof. Observe that each segment of g causes at most one region to be split. Thus the number of leaves in the decision tree created will be at most $\sigma + 1$.

By Lemma 1 we get that two segments are found as a result of the counterexample to each equivalence query (here the second halfspace is implicitly found by the call to SPLIT-REGION). Thus at most $\frac{\sigma}{2} + 1$ equivalence queries will be made.

Furthermore, since there are at most two membership queries made to query the upper and lower corners of each leaf, and $\lceil \log n \rceil + d$ membership queries used in the binary searches for the σ halfspaces, it follows that the number of membership queries made is at most $2\sigma + \sigma(\lceil \log n \rceil + d) = O(\sigma(\log n + d))$.

Observe that the depth of T is at most σ since any of the halfspaces defined by the σ segments in the target concept will appear at most once on any path from the root to a leaf. Thus the time to locate the region to split is $O(\sigma)$ and it immediately follows that the time complexity is $O(\sigma(\sigma + \log n + d))$. □

COROLLARY 5. *The algorithm* ALT-LEARN-WITH-MQS *achieves exact identification for any* $g \in \bigcup_{\leq m} \text{BOX}_n^d$ *using time and queries polynomial in* m *and* $\log n$ *for* d *constant.*

Proof. This follows immediately from Theorem 4 and the observation that $\sigma \leq (2m)^d$. □

Finally, observe that just as we described in section 5.3, our final hypothesis can be converted to the union of $O(md \log m)$ boxes from BOX_n^d. Recall that the time to perform the conversion is $O((2m)^{2d} md \log m)$ and thus, this will be efficient only if d is constant. Also, using the technique of section 6 we can refine this algorithm to use only equivalence queries.

We now use a different method of analysis to show that ALT-LEARN-WITH-MQS uses time and queries polynomial in d and $\log n$ for m constant.

We begin by examining the number of regions created by this algorithm. Each region is represented by a single leaf in the hypothesis decision tree. Thus, we can find the number of regions by finding the number of leaves in our hypothesis. We now derive a recurrence relation for the number of leaves in the final decision tree. Let $g \in \bigcup_{\leq m} \text{BOX}_n^d$ be the target concept and T be the final decision tree output by our algorithm. For each internal node of T there is an associated region of $\mathcal{N}_n^d$. For any node r in T, let h_r denote the height of the subtree of T rooted at r (we define the height of a leaf to be 0), and let m_r be the minimum number of boxes needed to cover the part of g contained in the region of $\mathcal{N}_n^d$ associated with r. Thus, for the region r corresponding to the root of T we have that $m_r \leq m$ and $h_r \leq 2md$ since, by Observation 1, the height of T is at most $2md$.

Let $L(m, h)$ denote the maximum number of leaves in a decision tree rooted at a node r with $m_r = m$ and $h_r = h$. Then we have that $L(m, h) = L(m, h-1) + L(m-1, h-1)$, where, for all $m \geq 0$, $L(m, 0) = 1$, and for all $h \geq 1$, $L(0, h) = 1$. To see this, observe that when we find, while building the hypothesis, a hyperplane that splits a node, the two subproblems that correspond to the left and right children both must have at most $h - 1$ hyperplanes left to find since in the worst case, all other hyperplanes are split by this one and thus appear on both sides. Finally, since the hyperplane just found must be the side of one of the m boxes, that box will not appear in one of the recursive calls. (In the worst case all other boxes will be split).

Consider the "largest possible" decision tree, according to the upper bound $L(m, h)$. (Note that a tree constructed from a problem instance will generally have fewer leaves.) Notice that in such a "largest" decision tree, there are no leaves in the decision tree on levels 0 (i.e., the root level) to $m - 1$. On each level from m to h there are leaves

caused by the base case $m = 0$ of the recurrence. The total number of these leaves is given by

$$\sum_{j=m}^{h} \binom{j-1}{m-1}$$

since the number of nodes, in the recursion tree, with $m = 0$ at level j is equal to the number of nodes with $m = 1$ at level $j - 1$. There are also leaf nodes caused by the other base case of the recurrence, $h = 0$. Note that this is the last level of the tree. The number of leaves here is given by

$$\sum_{j=0}^{m-1} \binom{h}{j}$$

since this gives the number of nodes for each nonzero value of m (the $m = 0$ nodes on this level were already counted in the previous expression). Thus, the total number of leaves is

$$\begin{aligned}
\sum_{j=m}^{h} \binom{j-1}{m-1} + \sum_{j=0}^{m-1} \binom{h}{j} &= \sum_{j=m-1}^{h-1} \binom{j}{m-1} + \sum_{j=0}^{m-1} \binom{h}{j} \\
&= \sum_{j=0}^{h-1} \binom{j}{m-1} + \sum_{j=0}^{m-1} \binom{h}{j} \\
&= \binom{h}{m} + \sum_{j=0}^{m-1} \binom{h}{j} \\
&= \sum_{j=0}^{m} \binom{h}{j}. \qquad (9.1)
\end{aligned}$$

Recall that $h \leq 2md$. In the following lemma we show that $(2md)^m$ is an upper bound on the summation in (9.1).

LEMMA 3. *The number of leaves in any hypothesis decision tree constructed by* ALT-LEARN-WITH-MQS *is bounded above by* $(2md)^m$ *for* $m > 1$ *and is* $2d + 1$ *for* $m = 1$.

Proof. Let $n = md$ and $k = m$ for $m > 1$. Then the expression we have derived for the number of leaves is $\sum_{j=0}^{k} \binom{2n}{j}$. It is easily shown by induction on k that $\sum_{j=0}^{k} \binom{2n}{j} \leq (2n)^k$ for $n \geq k > 1$, and thus the result follows for $m > 1$. Finally, for $m = 1$ the number of leaves in the hypothesis is $2d + 1$. □

We are now ready to prove the running time of our algorithm using this method of analysis. For ease of exposition we assume $m > 1$ in the remainder of this paper.

THEOREM 6. *Given any target concept* $g \in \bigcup_{\leq m} \text{BOX}_n^d$, *the algorithm* ALT-LEARN-WITH-MQS *achieves exact identification of* g, *making at most* $(2md)^m$ *equivalence queries, making* $O((2md)^m \cdot (\log n + d))$ *membership queries, and using* $O((2md)^{m+1} \cdot (\log n + d))$ *time.*

Proof. Observe that the number of counterexamples received by ALT-LEARN-WITH-MQS is at most the number of internal nodes in our final decision tree. Thus the number of equivalence queries made by ALT-LEARN-WITH-MQS is at most the number of leaves in the final decision tree.

Equation (9.1) shows that $L(m,h) = \sum_{j=0}^{m} \binom{h}{j}$, and Lemma 3 proves a bound for the number of leaves of T of at most $\sum_{j=0}^{m} \binom{2md}{j} \leq (2md)^m$. Thus it immediately follows that at most $(2md)^m$ equivalence queries are made. Since at most $\lceil \log n \rceil + d$ membership queries are used by the binary search procedure when splitting a node, it follows that the number of membership queries made by ALT-LEARN-WITH-MQS is $O((2md)^m \cdot (\log n + d))$. Finally, since it takes $O(md)$ time to find the node corresponding to the region containing the counterexample and at most $O(\log n + d)$ time for each binary search, it follows that the time complexity of ALT-LEARN-WITH-MQS is $O((2md)^{m+1} \cdot (\log n + d))$. □

Thus ALT-LEARN-WITH-MQS achieves exact identification for any $g \in \bigcup_{\leq m} \text{BOX}_n^d$ using time and queries polynomial in d and $\log n$ for m constant. We note that for $m \geq 6$ we can remove a factor of 2^m in the complexity by using the tighter upperbound that $\sum_{j=0}^{m} \binom{2md}{m} \leq (md)^m$.

10. Concluding remarks. We have given an efficient algorithm that uses membership and equivalence queries to exactly identify any concept from $\bigcup_{\leq m} \text{BOX}_n^d$ for d constant. Furthermore, this algorithm makes at most $md + 1$ equivalence queries, all of which can be formulated as the union of $O(md \log m)$ axis-parallel boxes.

We have also shown how to extend our basic algorithm to learn efficiently, using *only* equivalence queries, any discretized geometric concept generated from any number of halfspaces with any number of known (to the learner) slopes in a constant dimensional space. In particular, our algorithm exactly learns (from equivalence queries *only*) unions of discretized axis-parallel boxes in constant dimensional space in polynomial time. Further, this algorithm can be modified to handle a polynomial number of lies in the counterexamples provided by the environment.

Finally, we have introduced a new complexity measure, σ, that better captures the complexity of the union of m boxes than simply the number of boxes and the dimension. We presented an algorithm that uses time and queries polynomial in σ and $\log n$. In fact, the time and queries (both membership and equivalence) used by this single algorithm are polynomial for *either* m or d constant.

A number of important open questions that we have not answered concern the necessity of membership queries to exactly learn the class $\bigcup_{\leq m} \text{BOX}_n^d$ (or the more general class of a discretized geometric concept) in time polynomial in d and $\log n$ when the number of boxes (or defining hyperplanes) is constant.

While we have provided an algorithm to efficiently learn geometric concepts defined by hyperplanes that are *not* axis parallel, to achieve this goal it was necessary that the learner be given a priori knowledge as to the slopes of the hyperplanes. An interesting direction is to explore the learnability (even for fixed dimensions) of geometric concepts defined by hyperplanes whose exact slopes are not known to the learner.

Finally, it would be interesting to see if $\bigcup_{\leq m} \text{BOX}_n^d$ can be efficiently learned in time polynomial in m and $\log n$ for $d = O(\log m)$ or in time polynomial in d and $\log n$ for $m = O(\log d)$ (i.e., a generalization of the Blum and Rudich [9] result that $O(\log n)$-term DNF formulas are exactly learnable). Of course, since $\bigcup_{\leq m} \text{BOX}_n^d$ generalizes the class of DNF formulas, it seems very unlikely that one could develop an algorithm for the unrestricted case of $\bigcup_{\leq m} \text{BOX}_n^d$ that is polynomial in m, $\log n$, and d. It may be possible, however, to obtain a truly polynomial algorithm for some subclass of $\bigcup_{\leq m} \text{BOX}_n^d$.

Acknowledgments. We thank Peter Auer and the COLT committee for their comments. We thank Wolfgang Maass for suggesting we consider the material discussed in section 5.3. We also thank the anonymous referees whose comments improved this manuscript.

REFERENCES

[1] D. Angluin, *Queries and concept learning*, Machine Learning, 2 (1988), pp. 319–342.
[2] D. Angluin and M. Krikis, *Learning with malicious membership queries and exceptions*, in Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory, 1994, pp. 57–66.
[3] D. Angluin and P. Laird, *Learning from noisy examples*, Machine Learning, 2 (1988), pp. 343–370.
[4] E. Arkin, P. Belleville, J. Mitchell, D. Mount, K. Romanik, S. Salzberg, and D. Souvaine, *Testing simple polygons*, in Proceedings of the 5th Canadian Conference on Computational Geometry, 1993, pp. 387–392.
[5] E. Arkin, H. Meijer, J. Mitchell, D. Rappaport, and S. Skiena, *Decision trees for geometric models*, in Proceedings of the 9th Annual Symposium on Computational Geometry, 1993, pp. 369–378.
[6] P. Auer, *On-line learning of rectangles in noisy environments*, in Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, 1993, pp. 253–261.
[7] P. Belleville and T.C. Shermer, *Probing polygons mimimally is hard*, Comput. Geom., 2 (1993), pp. 255–265.
[8] A. Blum, *Separating distribution-free and mistake-bounded learning models over the Boolean domain*, SIAM J. Comput., 23 (1994), pp. 990–1000.
[9] A. Blum and S. Rudich, *Fast learning of k-term DNF formulas with queries*, J. Comput. System Sci., 51 (1995), pp. 367–373.
[10] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, *Learnability and the Vapnik-Chervonenkis dimension*, J. Assoc. Comput. Mach., 36 (1989), pp. 929–965.
[11] N. Bshouty, Personal communication, University of Calgary, 1994.
[12] N. Bshouty, Z. Chen, and S. Homer, *On learning discretized geometric concepts*, in 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 54–63.
[13] N. H. Bshouty, S. A. Goldman, T. R. Hancock, and S. Matar, *Asking questions to minimize errors*, J. Comput. System Sci., 52 (1996), pp. 268–286.
[14] W. J. Bultman and W. Maass, *Fast identification of geometric objects with membership queries*, in Proc. 4th Annual Workshop on Computational Learning Theory, San Mateo, CA, Morgan Kaufmann, 1991, pp. 337–353.
[15] Z. Chen, *Learning unions of two rectangles in the plane with equivalence queries*, in Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, ACM Press, 1993, pp. 243–252.
[16] Z. Chen and S. Homer, *The Bounded Injury Priority Method and the Learnability of Inions of Rectangles*, unpublished manuscript, 1994.
[17] Z. Chen and W. Maass, *On-line learning of rectangles*, Machine Learning, 17 (1994), pp. 23–50.
[18] V. Chvatal, *A greedy heuristic for the set covering problem*, Math. Oper. Res., 4 (1979), pp. 233–235.
[19] J. C. Culberson and R. A. Reckhow, *Covering polygons is hard*, in 29th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 601–611.
[20] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, New York, 1987.
[21] M. Frazier, S. Goldman, N. Mishra, and L. Pitt, *Learning from a consistently ignorant teacher*, J. Comput. System Sci., 52 (1996), pp. 472–492.
[22] P. W. Goldberg, S. A. Goldman, and H. D. Mathias, *Learning unions of boxes with membership and equivalence queries*, in Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory, 1994, pp. 198–207.
[23] Y.-B. Jia and M. Erdmann, *The complexity of sensing by point sampling*, in Proceedings of the First Workshop of the Algorithmic Foundations of Robotics, 1994.
[24] Y.-B. Jia and M. Erdmann, *Geometric sensing of known planar shapes*, International Journal of Robotics Research, 15 (1996), pp. 290–299.
[25] S. Homer and Z. Chen, *Fast Learning Unions of Rectangles with Queries*, unpublished

manuscript, 1993.
[26] M. Kearns and M. Li, *Learning in the presence of malicious errors*, in Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, 1988.
[27] P. D. Laird, *Learning from Good and Bad Data*, Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, Boston, 1988.
[28] N. Littlestone, *Learning when irrelevant attributes abound: A new linear-threshold algorithm*, Machine Learning, 2 (1988), pp. 285–318.
[29] P. M. Long and M. K. Warmuth, *Composite geometric concepts and polynomial predictability*, in Proceedings of the Third Annual Workshop on Computational Learning Theory, Morgan Kaufmann, 1990, pp. 273–287.
[30] W. Maass and G. Turán, *On the complexity of learning from counterexamples*, in 30th Annual Symposium on Foundations of Computer Science, 1989, pp. 262–267.
[31] W. Maass and G. Turán, *Lower bound methods and separation results for on-line learning models*, Machine Learning, 9 (1992), pp. 107–145.
[32] W. Maass and G. Turán, *Algorithms and lower bounds for on-line learning of geometrical concepts*, Machine Learning, 14 (1994), pp. 251–269.
[33] W. Maass and G. Turán, *How fast can a threshold gate learn?*, in Computational Learning Theory and Natural Learning Systems: Constraints and Prospects, G. Drastal, S. J. Hanson, and R. Rivest, eds., MIT Press, Cambridge, MA, 1994, pp. 318–414.
[34] W. Maass and M. Warmuth, *Efficient learning with virtual threshold gates*, in Proceedings of the 12th International Conference on Machine Learning, Morgan Kaufmann, 1995, pp. 378–386.
[35] K. Romanik and C. Smith, *Testing Geometric Objects*, Technical report UMIACS-TR-90-69, University of Maryland College Park, Department of Computer Science, 1990.
[36] S. Skiena, *Problems in geometric probing*, Algorithmica, 4 (1989), pp. 599–605.
[37] R. H. Sloan, *Four types of noise in data for PAC learning*, Information Processing Letters, 54 (1992), pp. 157–162.
[38] L. Valiant, *A theory of the learnable*, Communications of the ACM, 27 (1984), pp. 1134–1142.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 700–703

FAST EXPONENTIATION USING DATA COMPRESSION*

YACOV YACOBI†

Abstract. We present the first exponentiation algorithm that uses the entropy of the source of the exponent to improve on existing exponentiation algorithms when the entropy is smaller than $(1 + w(S)/l(S))^{-1}$, where $w(S)$ is the Hamming weight of the exponent, and $l(S)$ is its length. For entropy 1 it is comparable to the best-known general purpose exponentiation algorithms.

Key words. fast exponentiation, cryptography, discrete-log, compression

AMS subject classifications. 68P25, 68Q25

PII. S0097539792234974

1. Introduction. Exponentiation with huge numbers is used heavily in modern cryptography, and the topic is the focus of attention of many researchers who try to achieve efficient exponentiation algorithms ([1], [2], [3], [4], [12], and many others). This paper is the first to make use of the entropy of the source of the exponent. In cases of entropy smaller than $(1 + w(S)/l(S))^{-1}$, where $w(S)$ is the Hamming weight of the exponent and $l(S)$ is its length, this method becomes asymptotically faster than all known general methods. For entropy 1 it is asymptotically comparable to the best-known general-purpose algorithms. The method is applicable to every repeated group operation. A preliminary version appeared in [13].

We now present an outline of the paper. Section 2 gives the required elements of the LZ theory, section 3 applies this theory to fast exponentiation, section 4 is a brief overview of other exponentiation methods, and the final section gives a cryptographic viewpoint.

2. The LZ theory. For the sake of completeness we present here the definitions and theorems of [8], which are needed for the complexity analysis. (The actual compression algorithm is in [9], [10]. The earlier paper, [8], is written more along the lines of decompression.)

Let A^* denote the set of all finite-length sequences over a finite alphabet A. Let $l(S)$ denote the length of $S \in A^*$ and let $A^n = \{S \in A^* | l(S) = n\}$, $n \geq 0$. The null sequence χ is in A^*. We use $S(i, j)$ to denote a substring of S that starts at location i and ends at location j. Let $S\pi^i$ denote $S(1, l(S) - i)$, $\quad i = 0, 1, \ldots, l(S)$. The *vocabulary*, $v(S)$, of S is the subset of A^* formed by all the substrings (words) of S. When a sequence S is extended by concatenation with one of its words, say $W = S(i, j)$, the resulting sequence $R = SW$ can be viewed as being obtained from S through a copying procedure. The same recursive copying procedure could be applied to generate an extension $R = SQ$ of S which is much longer than warranted by any word in $v(S)$. The only provision is that Q be an element of $v(SQ\pi)$.

We use the denotation $S \Rightarrow R$ if $R = SQ$ can be obtained from S by application of the above copying procedure, where at the end of the copying process we use "one-symbol innovation" (any symbol from A, not subject to the copying procedure). This process is called *reproduction*, while a single-step copying without innovation is called

*Received by the editors July 27, 1992; accepted for publication (in revised form) February 20, 1997; published electronically August 4, 1998.
http://www.siam.org/journals/sicomp/28-2/23497.html
†Microsoft, One Microsoft Way, Redmond, WA 98052 (yacov@microsoft.com).

production and is denoted $S \to R$. If R cannot be obtained from S by production we write $S \not\to R$.

Consider an m-step production process of a sequence S and let $S(1, h_i), \quad i = 1, 2, \ldots, m, \quad h_1 = 1, \quad h_m = l(S)$, be the m states of the process. The parsing of S into $H(S) = S(1, h_1)S(h_1+1, h_2) \cdots S(h_{m-1}+1, h_m)$ is called the *production history* of S, and the m words $H_i(S) = S(h_{i-1}+1, h_i), \quad i = 1, 2, \ldots, m$, where $h_0 = 0$, are called the components of $H(S)$. A component $H_i(S)$ and the corresponding reproduction step $S(1, h_{i-1}) \Rightarrow S(1, h_i)$ are called *exhaustive* if $S(1, h_{i-1}) \not\to S(1, h_i)$; a history is called exhaustive if each of its components, with the possible exception of the last one, is such. Every nonnull sequence S has a unique exhaustive history (denoted $E(S)$).

Let $c_H(S)$ denote the number of components in a history $H(S)$ of S. The production complexity of S is defined as $c(S) = \min\{c_H(S)\}$, where minimization is over all histories of S. Let $c_E(S)$ denote the production complexity of the exhaustive history of S.

$$\forall\ S, \quad c(S) = c_E(S). \tag{1}$$

Let $\alpha = |A|$. All logarithms are to base α. Let $\epsilon_n = 2(1 + \log\log(\alpha n))/\log(n)$. Then

$$\forall\ S \in A^n, \quad c(S) < n/((1 - \epsilon_n)\log(n)). \tag{2}$$

Consider an ergodic α-symbol source with normalized entropy $h, \quad 0 \le h \le 1$. For this source we have asymptotically:

$$c(S) \le hn/\log(n). \tag{3}$$

The previous three results were proved in [8].

While [8] is written along the lines of decompression, the two later papers, [9] and [10], give an efficient compression algorithm that creates the exhaustive history of any given sequence.

3. Using the LZ compression method for fast exponentiation. The binary case is presented for concreteness. The natural many-to-one mapping from sequences to integers is used, and it is assumed that $S(1, 1)$ is the least significant bit. S is used to denote both integer and sequence.

Given the exponent S, the computation of any exponentiation x^S proceeds as follows.

Build a binary tree where each path from the root to any node corresponds to some segment of the exponent $S(i, j)$, and the node contains the result of $x^{S(i,j)}$. The root contains $1 = x^0$ (0 corresponds to the string χ). Proceed inductively as follows. Suppose that the component $S(i, j)$ was already processed; i.e., the tree already contains a path from the root to some leaf which corresponds to this component, and the leaf contains the result of $x^{S(i,j)}$. Traverse the partial tree from the root according to the new (so far unscanned) bits of $S(j + 1, \ldots)$ until you reach a leaf. Proceed with S having one more symbol. The new component contains now exactly one new untraversed symbol. Extend the tree according to the new symbol, and mark the new branch with this symbol. Compute the value of the new leaf. This simple construction (without the exponentiation) is the heart of the LZ algorithm. Ziv and Lempel proved that this construction creates the exhaustive history of S, $H(S) = S(1, h_1)S(h_1 + 1, h_2) \cdots S(h_{m-1} + 1, h_m)$, where each path from the root to any node corresponds to one of the components of $H(S)$, and hence the number of nodes in the tree is $c(S)$.

For depth i we need to do one squaring (to compute $X^{2^i} = (X^{2^{i-1}})^2$), and whenever the new bit is 1 we need to multiply that value by the value of the father. For random exponents we can expect that to happen in half the cases for a total cost of $c(S)/2$, and in general, for exponent of length $l(S)$ and expected Hamming weight $w(S)$, the expected number of cases where the new symbol is 1, is $c(S) \cdot w(S)/l(S)$.

To evaluate the cost of squarings we need to know more about the shape of the LZ trees. In [7] an analysis is given, showing that the height of the tree has normal distribution with mean value $\log_\alpha(c(S))/h$, where h is the entropy of the source, and the deviation is rather big: $O(\log_\alpha(c(S)))$ (big "oh"); i.e., for almost all cases the squaring component is $o(c(S))$ (little "oh").

We combine the partial results (stored in the leaves) using $l(S)$ squarings and $c(S)$ multiplications in the natural way. The total expected complexity is $\mathcal{C}(S) = l(S) + (1 + w(S)/l(S))c(S) + o(c(S))$. Plugging in (3) and defining $\sigma(S) = 1 + w(S)/l(S)$, $1 \le \sigma(S) \le 2$, we get the asymptotical upper bound

$$\mathcal{C}(S) = l(S) + \sigma(S) \cdot h \cdot l(S)/\log(l(S)) + o(l(S)/\log(l(S))). \tag{4}$$

When $h < \sigma(S)^{-1}$ the above method wins asymptotically in expected complexity over all existing general purpose methods.

Applying the algorithm in practice. We improve somewhat on a straightforward LZ compression, by taking advantage of leading zeros. Leading zeros are not accounted for in the binary tree, thus reducing the tree size. When strings are taken from a uniform distribution the expected length of the run of zeros is $\sum_{j=1}^{\infty} j \cdot 2^{-(j+1)} = 1$, thus truncating the expected tree height by 1.

To raise an integer x to power S, the bits of S are parsed using an LZ parsing from least to most significant. Each time a new phrase is started, the leading zeros are skipped, a root-to-leaf path is traversed (skipping over the corresponding bits), and a new leaf is formed for the next bit. Stored at this new leaf are its depth, i, a back pointer to the previous parsed phrase, the number of leading zeros, and a value which is set to that of its parent if the next bit is zero, or that of its parent times x^{2^i}, if the next bit is 1. After S has been parsed into substrings in this manner we know the value of x raised to each of the parsed substrings, and the result can then be combined by following back pointers and successively multiplying by the next term and then raising to the power of 2 to {the number of leading zeros plus the length of the next term}. The powers of 2 needed in the above description can be computed once by successive squaring for all future uses.

4. Concluding remarks: Other methods. The addition-chain method runs in time $n+n/\log(n)+o(n/\log(n))$ ([5], [12], [4]), once an optimal addition chain for the exponent is found; however finding an optimal addition chain is NP-hard. Suboptimal heuristics exist. The m-ary method [11] (the most popular among practitioners) runs in time upper bounded by $n+n/m+2^{m-1}+1$, where m is an optimization parameter. For example, for $n = 1024$, $m = 6$ is optimal. In the practical range this is comparable to the complexity of the new method, with $h = 1$ and $w(S)/l(S) = 1/2$. Other heuristics exist [1].

In some applications the exponent is fixed for many bases, in which case it is natural to partition the operations into two stages: exponent processing and exponentiation. The new method is comparable to optimal addition chains in the exponentiation phase even for $h = 1$, and wins when $h < 1$.

In [2] a method is proposed that optimizes exponentiation complexity when the base is fixed for many exponents.

Cryptographic viewpoint. For discrete-log–based cryptosystems, when the modulus is "good," we can go down with $w(S)$ quite far before the complexity of the discrete-log problem deteriorates, per the best-known algorithm [6]. Specifically, the complexity is $A^{1/2}$, where

$$A = \left(\begin{array}{c} l(S) \\ w(S) \end{array} \right).$$

For example, for $l(S) = 512$ bits, $w(S) = 22$ gives $A^{1/2} = 2^{64}$.

When the Hamming weight is very low, the entropy is also low, thus we gain using the new method. Specifically we have $h \leq l^{-1} \cdot \log_2 A$ affecting the second term of $\mathcal{C}(\mathcal{S})$ (in the above example $h < 1/4$, while $\sigma(S) \approx 1$).

The above numbers are not a recommendation. The discrete-log problem for low-entropy exponents is not well studied yet (however the method of [6] *does not seem to apply directly to low entropy).*

Acknowledgments. I am indebted to Arjen Lenstra for many insightful discussions. I would also like to thank Shimon Even, Rich Graveman, and Stuart Haber for many helpful comments. Finally I would like to thank two anonymous referees for numerous helpful comments, and in particular for pointing me to the paper by P. Jacquet and W. Szpankowski [7].

REFERENCES

[1] J. Bos and M. Coster, *Addition chain heuristics*, in Proc. Crypto'89, New York, Lecture Notes in Computer Sci. 435, Springer-Verlag, pp. 400–407, 1990.

[2] E. Brickell, D.M. Gordon, K.S. McCurley, and D. Wilson, *Fast exponentiation with precomputation*, in Proc. Eurocrypt'92, New York, Lecture Notes in Computer Sci. 658, Springer-Verlag, pp. 221–238.

[3] H. Cohen and A.K. Lenstra, *Implementation of a new primality test*, Math. Comput., 48 (1987), pp. 103–121.

[4] Downey, Leony, and Sethi, *Computing sequences with addition chains*, SIAM J. Comput., 3 (1981), pp. 638–696.

[5] P. Erdös, *Remarks on number theory* III, *on addition chains*, Acta Arith., VI (1960), pp. 77–81.

[6] R. Heiman, *A Note on Discrete Logarithms with Special Structure*, Bellcore TM-ARH-021448, 1992. Also in Proc. Eurocrypt'92, New York, Lecture Notes in Computer Sci. 658, Springer-Verlag, pp. 454–457.

[7] P. Jacquet and W. Szpankowski, *What can we learn about suffix trees from independent tries?*, in Proc. Algorithms and Data Structures, 2nd workshop, WADS'91, pp. 228–239.

[8] A. Lempel and J. Ziv, *On the complexity of finite sequences*, IEEE Trans. Inform. Theory, IT-22, (1976),

[9] J. Ziv and A. Lempel, *A universal algorithm for data compression*, IEEE Trans. Inform. Theory, IT-23, 1977.

[10] A. Lempel and J. Ziv, *Compression of individual sequences via variable rate coding*, IEEE Trans of Inform. Theory, IT-24 (1978), pp. 530–536.

[11] D. Knuth, *The Art of Computer Programming*, Vol. 2, *Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1980, pp. 441–462.

[12] Schönhage, *A Lower bound on the length of addition chains*, Theoret. Comput. Sci., 1 (1975), pp. 229–242.

[13] Y. Yacobi, *Exponentiating faster with addition chains*, in Proc. Eurocrypt' 90, Springer-Verlag, New York, Lecture Notes in Computer Sci. 473, pp. 222–229.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 704–708

A POLYNOMIAL TIME COMPLEXITY BOUND FOR COMPUTATIONS ON CURVES*

P. G. WALSH†

Abstract. In this paper, we use a recent quantitative version of Eisenstein's theorem on power series expansions of algebraic functions to compute a polynomial-time bit complexity bound for the computation of the genus of an algebraic curve over the rationals. The result is extendable to any field of characteristic zero which is finitely generated over the rationals.

Key words. algebraic curves, genus, Puiseux expansion, complexity

AMS subject classification. 14H05

PII. S0097539796300969

1. Introduction.

In [3] Duval describes a method based on computing Puiseux expansions for certain computations on curves. These include the computation of the ramification indices in the projection of a curve on any line, the genus of an algebraic curve, the construction of rational functions on a curve, and finding the irreducible components of a curve. An interesting aspect of this work is that it is shown that all of these computations can be done in polynomial time in the number of arithmetic operations. The shortcoming of this result is that there is no prior knowledge about the size of the integers on which these arithmetic operations are performed. In fact, due to a result of Coates (Lemma 3 of [2]) on the coefficients of algebraic power series, it is conceivable that Duval's algorithm would be completely impractical because of the exponential growth of the size of the integers involved. In this paper we use a recent result of Schmidt [11] on the coefficients of algebraic power series to prove that such exponential growth does not occur. As a result, the aforementioned computations on a curve can be computed in time which is polynomial in the size of the polynomial defining the curve.

Other work in this direction has been carried out by Chistov [1]. The main difference here is to compute explicit estimates on the number of bit operations required to complete the algorithms and show how this polynomial-time complexity depends entirely on the quantitative version of Eisenstein's theorem given in [11]. Kozen [6] has also described an algorithm for the resolution of plane curve singularities which has polynomial-time complexity in the number of bit operations. The method is based on passive factorization.

The main point of our paper is to show that the classical method, based on Puiseux expansion computation, does in fact have polynomial-time complexity in the number of bit operations, which is contrary to the wide belief that Coates' algorithm runs in exponential time. We make no claims as to the practicality of the classical method over the newer ones. In this regard, it is worth noting that Teitelbaum [12] also presents an efficient method for the resolution of singularities based on passive factorization.

*Received by the editors March 25, 1996; accepted for publication (in revised form) December 19, 1997; published electronically August 4, 1998.
http://www.siam.org/journals/sicomp/28-2/30096.html
†Department of Mathematics, University of Ottawa, 585 King Edward, Ottawa, ON K1N 6N5, Canada (gwalsh@jeanne.mathstat.uottawa.ca). This research was supported by an NSERC Postdoctoral Fellowship.

2. Puiseux expansions of algebraic functions. In this section we describe Puiseux expansions and state a result on the computation of Puiseux expansions of algebraic functions. The proof can be found in [15].

We recall Puiseux's theorem (for example, see [4]), which states that at any point $\alpha \in \mathbf{C} \cup \{\infty\}$ there is a factorization

$$F(x,y) = A_n(x) \prod_{i=1}^{n} (y - y_{i,\alpha}(x)),$$

where for $i = 1, \ldots, n$

$$y_{i,\alpha} = \sum_{k=f_{i,\alpha}}^{\infty} a_{k,i,\alpha} (z_\alpha^{1/e_{i,\alpha}})^k, \tag{1}$$

with $f_{i,\alpha} \in \mathbf{Z}$, $e_{i,\alpha} \in \mathbf{Z}^+$, $a_{k,i,\alpha} \in \mathbf{C}$, $a_{f_{i,\alpha},i,\alpha} \neq 0$, and $z_\alpha = (x - \alpha)$ if $\alpha \in \mathbf{C}$, $z_\alpha = 1/x$ if $\alpha = \infty$. The *singular part* of $y_{i,\alpha}$ is the partial sum

$$y_{i,\alpha,T} = \sum_{k=f_{i,\alpha}}^{T} a_{k,i,\alpha} (z_\alpha^{1/e_{i,\alpha}})^k,$$

where T is minimal such that $y_{i,\alpha,T}$ distinguishes $y_{i,\alpha}$ from the other $n-1$ expansions at α of y. The number $e_{i,\alpha}$ is the *ramification index* of $y_{i,\alpha}$ and can be obtained by computing the singular part $y_{i,\alpha,T}$ of $y_{i,\alpha}$.

The n Puiseux expansions are partitioned into *cycles*, where the cycle corresponding to the expansion in (1) is the set of $e_{i,\alpha}$ series

$$y_{i,\alpha}^{(j)} = \sum_{k=f_{i,\alpha}}^{\infty} a_{k,i,\alpha} (\zeta_{e_{i,\alpha}}^{(j)} z_\alpha^{1/e_{i,\alpha}})^k, \qquad j = 0, \ldots, e_{i,\alpha} - 1,$$

where $\zeta_{e_{i,\alpha}}$ denotes a primitive $e_{i,\alpha}$th root of unity.

THEOREM 2.1. *Let $F(x,y) \in Z[x,y]$ be a square-free polynomial of degree n in y, m in x, and let h denote the maximum of the absolute values of the coefficients of F. Then, for any $\epsilon > 0$, the singular part of a Puiseux expansion at $x = 0$ of the algebraic function y defined by $F(x,y) = 0$ can be computed in*

$$O(n^{32+\epsilon} m^{3+\epsilon} \log^{2+\epsilon}(h))$$

bit operations.

The proof of this result is based on Schmidt's recent quantitative version of Eisenstein's theorem on power series expansions of algebraic functions [11]. Schmidt exploits the fact that the algebraic function satisfies a linear differential equation and is able to use this to prove a result which is roughly an order of magnitude smaller than the corresponding result proved by Coates in Lemma 3 of [2].

3. Computations on curves. In this section we state the main results of the paper. Throughout this section we let $F(x,y)$ denote a polynomial with integer coefficients of degree n in y, m in x, and h denotes the maximum of the absolute values of the coefficients of F, referred to as the *height* of F and denoted $ht(F)$.

THEOREM 3.1. *If F is square-free in $\mathbf{Q}[x,y]$, then, for any $\epsilon > 0$, the ramification indices at all critical places of F and the genus of the algebraic curve defined by $F(x,y) = 0$ can be computed in*

$$O(n^{70} m^{40} \left(n^{2+\epsilon} + m^{2+\epsilon} \log^{2+\epsilon}(h)\right))$$

bit operations.

In this section we describe an algorithm to compute the genus of the curve $F(x, y) = 0$. This is accomplished by computing the Puiseux expansions at all places which correspond to singularities of the algebraic function y defined by $F(x, y) = 0$ and then applying the Hurwitz genus formula.

The Hurwitz genus formula is

$$g = 1 - n + (1/2)\sum(e - 1), \tag{2}$$

where the sum is the sum of all ramification indices over all places and all cycles at each place. It is well known (see [4]) that if α is a regular point, then $e_{i,\alpha} = 1$. As a result, for computing the genus of the curve, we need only compute the ramification index for each cycle at each critical place α. Finite critical places α are characterized by the property that they are zeros of the resultant polynomial $R(x) = res_t(F, F_y)$, where F_y is the derivative of F with respect to y. Note that since F is square-free, $R(x)$ is a nonzero polynomial.

4. Algorithm to compute the genus of the curve $F(x, y) = 0$.

1. Compute $R(x) = res_y(F, F_y)$.
2. Compute $S(x) = R(x)/gcd(R(x), R'(x))$.
3. Factor $S(x)$ into irreducibles $P_1, \ldots, P_k$ in $\mathbf{Q}(x)$.
4. For each $i = 1, \ldots, k$, compute $G_i(x, y) = res_t(P_i(t), F(x + t, y))$.
5. For each $i = 1, \ldots, k$, compute $H_i(x, y) = G_i(x, y)/gcd(G_i, (G_i)_y)$.
6. For each $i = 1, \ldots, k$, compute the singular part at $x = 0$ of the Puiseux expansions of the algebraic function y defined by $H_i(x, y) = 0$.
7. Let $F_\infty(x, y) = x^m F(1/x, y)$, and compute the singular part at $x = 0$ of the Puiseux expansions of the algebraic function y defined by $F_\infty(x, y) = 0$.
8. Use the ramification indices computed in steps 6 and 7 in (1) to compute the genus.

Before proceeding to the complexity of the algorithm, we first give a justification for its correctness. The purpose of the algorithm is to compute the singular part of all of the Puiseux expansions at each place α, where α is a critical place, that is, a root of $R(x)$ or the infinite place. For a fixed irreducible factor $P(x)$ of $R(x)$ we can simultaneously compute all of the Puiseux expansions of y at all places corresponding to the roots of $P(x)$ by computing the Puiseux expansions at $x = 0$ of the algebraic function defined by $G(x, y) = 0$, where $G(x, y) = \prod_i F(x + \alpha_i, y)$, and the product is over all roots α_i of $P(x)$. Moreover, this product is the norm of $F(x + \alpha, y)$ from $\mathbf{Q}(\alpha)$ to $\mathbf{Q}$, and hence has rational coefficients. By Theorem 1 of [10], this norm can be computed as described in step 4 above. By doing this for each of the irreducible factors of $R(x)$, we obtain the ramification indices corresponding to all cycles of every finite critical place of the algebraic function y defined by $F(x, y) = 0$. In step 7 we obtain the ramification indices of the cycles corresponding to the infinite place, and thus all of the required data are obtained to use (2).

4.1. Complexity analysis. Computationally, the dominant step in the algorithm is the computation of the singular parts, as described in step 6. To apply Theorem 2.1 we need to estimate the degree of $H_i(x, y)$ in x and in y and a bound for the size of its coefficients. We remind the reader that $n = deg_y F$, $m = deg_x F$, and h is the height of F.

$R(x)$ is the determinant of a $2n - 1 \times 2n - 1$ matrix whose coefficients are the coefficients of F and F_y, regarded as polynomials in y. Thus the determinant has $(2n - 1)!$ summands, where each summand is the product of n coefficients of F_y and

$n-1$ coefficients of F. Since the height of the polynomial $(1+\cdots+x^m)^{2n}$ is no larger than $(m+1)^{2n}$, we deduce that

$$ht(R) < (2n)^{2n}(nh)^n h^n (m+1)^{2n},$$

and $deg_x R(x) < (2n-1)m$. Fix i with $1 \le i \le k$. Since P_i is a factor of $R(x)$, it follows from Lemma 2 on p. 135 of [5] that $ht(P_i) < e^{(2n-1)m} ht(R)$; therefore,

$$ht(P_i) < 2^{2nm} n^n (2nh(m+1))^{2n}.$$

Now $G_i(x,y)$ is the determinant of the matrix with coefficients of $P_i(t)$ and $F(x+t,y)$, the latter regarded as a polynomial in t. Note that $deg_t P_i(t) < (2n-1)m$. It is easy to see that the height of $F(x+t,y)$, regarded as a polynomial in t, is no larger than $h(m+1)(n+1)2^m$, and so a similar argument as above shows that

$$ht(G_i) < (2n!m)^{2nm} (ht(P_i))^m (h(m+1)(n+1)2^m)^{2nm} ((n+1)(m+1))^{2nm}.$$

From the definition of G_i, we have that

$$deg_x G_i < 2nm^2, deg_y G_i < 2n^2 m.$$

By the aforementioned result of [5], $ht(H_i) < e^{2nm^2+2n^2m} ht(G_i)$, so we deduce finally that

$$\log(ht(H_i)) < c(nm^2 + n^2 m + nm^2 \log(nmh))$$

for some positive constant c. Moreover,

$$deg_x H_i < 2nm^2, \;\; deg_y H_i < 2n^2 m,$$

and so by Theorem 2.1, together with the fact that there are at most $2nm$ polynomials H_i, we deduce the result in Theorem 3.1.

REFERENCES

[1] A. L. Chistov, *Algorithms of Polynomial Complexity for Computational Problems in the Theory of Algebraic Curves,* Lomi preprint 176, Leningrad, 1989 (in Russian).

[2] J. Coates, *Construction of rational functions on a curve,* Proc. Camb. Phil. Soc., 68 (1970), pp. 105–123.

[3] D. Duval, *Computations on curves,* in Proc. Eurosam 84, Lecture Notes in Comput. Sci. 184, Springer-Verlag, New York, 1984, pp. 100–107.

[4] M. Eichler, *Introduction to the Theory of Algebraic Numbers and Functions,* Academic Press, London, 1966.

[5] A. O. Gel'fond, *Transcendental and Algebraic Numbers,* Dover, New York, 1960.

[6] D. Kozen, *Efficient resolution of singularities of plane curves,* in Proc. 14th Conf. Foundations of Software Technology and Theoret. Comput. Sci., 1994, to appear.

[7] A. Lenstra, H.W. Lenstra Jr., and L. Lovasz, *Factoring polynomials with rational coefficients,* Math. Ann., 261 (1982), pp. 515–534.

[8] A. Lenstra, *Factoring multivariate integral polynomials,* Theoret. Comput. Sci., 34 (1984), pp. 207–213.

[9] A. Lenstra, *Factoring polynomials over algebraic number fields,* in Proc. Eurocal. 1983, Lecture Notes in Comput. Sci. 162, Springer-Verlag, New York, 1983, pp. 245–254.

[10] R. Loos, *Computing in algebraic extensions,* in Computer Algebra, 2nd ed., B. Buchberger et al., eds., Springer-Verlag, Vienna, 1982, pp. 173–187.

[11] W. M. Schmidt, *Eisenstein's theorem on power series expansions of algebraic functions,* Acta Arith., 56 (1990), pp. 161–179.

[12] J. Teitelbaum, *The computational complexity of the resolution of plane curve singularities,* Math. Comp., 54 (1990), pp. 797–837.
[13] P. G. Walsh, *A quantitative version of Runge's theorem on Diophantine equations,* Acta Arith., 62 (1992), pp. 157–172.
[14] P. G. Walsh, *The Computation of Puiseux Expansions and a Quantitative Version of Runge's Theorem on Diophantine Equations,* Ph.D. Thesis, University of Waterloo, Waterloo, ON, Canada, 1994.
[15] P.G. Walsh, *A polynomial time complexity bound for the computation of a Puiseux expansion of an algebraic function,* Math. Comp., submitted.
[16] P.G. Walsh, *Irreducibility testing over local fields,* Math. Comp., submitted.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 709–719

STOCHASTIC CONTENTION RESOLUTION WITH SHORT DELAYS*

PRABHAKAR RAGHAVAN† AND ELI UPFAL‡

Abstract. We study contention resolution protocols under a stochastic model of continuous request generation from a set of contenders. The performance of such a protocol is characterized by two parameters: the maximum arrival rate for which the protocol is stable and the expected delay of a request from arrival to service.

Known solutions are either unstable for any constant injection rate or have at least polynomial (in the number of contenders) expected delay. Our main contribution is a protocol that is stable for a constant injection rate, while achieving logarithmic expected delay. We extend our results to the case of multiple servers, with each request being targeted for a specific server. This is related to the *optically connected parallel computer* (or *OCPC*) model. Finally, we prove a lower bound showing that long delays are inevitable in a class of protocols including backoff-style protocols, if the arrival rate is large enough (but still smaller than 1).

Key words. contention resolution, randomized algorithms, stochastic analysis

AMS subject classifications. 68Q25, 68Q75, 68M20, 60K30

PII. S0097539795285333

1. Introduction.

The subject of this paper is the stochastic analysis of protocols for contention resolution. The most concrete setting of this problem is that of multiple access channels, and so the remainder of the paper will use the terminology of this application. Naturally, our analyses are not specific to this setting and apply whenever we have several contenders requesting service from shared resources.

There are n *senders* and m *receivers*. At each of a series of time steps, one or more senders may generate a *packet*. A packet when generated has a *destination*: a unique receiver to which it must be delivered. Any sender may attempt to send a packet to any receiver at any step, but a receiver may only receive one packet in a step. If a receiver is sent more than one packet in a step (a *collision*), all packets sent to that receiver are lost and the senders are notified of the loss. The senders must then try to send these packets again at a future step. There is no explicit communication between the senders for coordinating the transmissions the only information that senders have is the packet(s) they have waiting for transmission and the history of losses. A packet can only be transmitted directly from its sender to its receiver; intermediate hops are disallowed.

The case $m = 1$ is a classical instance of sharing a common resource such as a bus or an Ethernet channel (the shared bus is modeled by the single "receiver"). The binary exponential backoff Ethernet protocol [9] is the solution used most commonly in practice here. The case $m = n$ has received much attention recently under the name *optically connected parallel computer*, or OCPC [8]. However, work on the OCPC model has been restricted to studies in which each sender begins with at most h

*Received by the editors April 24, 1995; accepted for publication (in revised form) March 28, 1997; published electronically August 4, 1998.
http://www.siam.org/journals/sicomp/28-2/28533.html

†IBM Almaden Research Center, 1650 Harry Rd., San Jose, CA 95120 (pragh@almaden.ibm.com). A portion of this work was done at the IBM T.J. Watson Research Center, Yorktown Heights, NY 10598.

‡The Weizmann Institute of Science, Israel (eli@wisdom.weizmann.ac.il), and IBM Almaden Research Center, 1650 Harry Rd., San Jose, CA 95120. Work at the Weizmann Institute of Science was supported in part by the Norman D. Cohen Professorial Chair of Computer Science, a MINERVA grant, and a grant from the Israeli Academy of Science.

packets for some positive integer h, with h or fewer packets bound for each receiver. In our work, we are interested in the more realistic setting in which packets are generated continuously. To this end, we adopt a stochastic model of packet generation.

1.1. The model. We assume that time is partitioned into intervals of equal length called *steps*. The injection distribution is characterized by an $n \times m$ matrix $\Lambda = (\lambda_{ij})$, in which $\sum_j \lambda_{ij} \leq \lambda$ for all i and $\sum_i \lambda_{ij} \leq \lambda$ for all j. At each step sender i generates a packet bound for receiver j with probability λ_{ij} independently of other senders and time steps. Since a sender can send no more than one packet per step, and a receiver can receive no more than one packet per step, we require $\lambda \leq 1$. For the special case $m = 1$, we use simplified notation: we assume that at each step sender i generates a packet with probability λ_i, independently of other steps and senders. Note that a sender generates at most one packet at a step, but several senders may generate a packet for a receiver. The reader is referred to the paper by Håstad, Leighton, and Rogoff [6] for a detailed account of the relation between these assumptions and reality.

Each sender uses a *protocol* to decide when to send a packet and what to do in the event of a collision. Informally, a protocol is an automaton that uses its state to remember the past. At each step, based on its state, the packets pending transmission, and any new packet it generates, it must decide which (if any) packet to transmit. For example, the binary exponential backoff protocol used in the Ethernet is the following: store arriving packets in a queue. When a packet reaches the head of the queue for the first time, transmit it. If there is a collision, retransmit the packet after T steps where T is selected randomly from $\{1, 2, 3, \ldots, 2^{\min\{10,b\}}\}$, where b is the number of times the packet has been involved in a collision.

Of primary interest is whether a protocol can sustain packet arrivals at some rate without *instability:* a protocol is unstable for a given arrival rate if the number of packets pending transmission grows unboundedly with time.[1] For stable protocols, two quantities are of interest: (1) the *maximum arrival rate* λ that can be sustained stably, and (2) the *delay*, defined to be the maximum over all senders of the expected number of steps from the generation of a packet to its delivery, in the steady state. Delay is of particular importance in high-speed communications applications such as video and ATM networks. (We actually prove a stronger result, a bound on the expected delay of any packet arriving after a polynomial number of steps from the start of the process.)

In our upper bounds, we do not assume that the sender knows the injection rates of other senders (i.e., the matrix Λ); we only assume that senders have some upper bound on the total number of active senders in the system. This bound does not need to be very accurate, as only the logarithm of this bound figures in the delay of our protocols. Since the total number of senders in the system is dictated by the hardware, it is reasonable to assume that this number cannot change very fast and that each sender has a good estimate of it. Our analysis holds even if the matrix Λ changes at every time step, subject to the constraints on row and column sums.

Throughout this paper, $\mathbf{Pr}[\mathcal{E}]$ denotes the probability of an event $\mathcal{E}$.

1.2. Related work. Most previous analysis focused on backoff protocols. The binary exponential backoff protocol used in the Ethernet was proposed by Metcalfe

[1] Formally let B_t denote the number of packets generated before time t that were not delivered until time t. Let F_t denote the distribution of B_t. A protocol is stable if the sequence $\{F_t \mid t \geq 1\}$ converges in distribution to a limit distribution F that is independent of t.

and Boggs [9]. Aldous [1] showed that for any positive constant λ, binary exponential backoff is unstable if the number of senders is infinite. Kelly [7] showed that any polynomial backoff protocol is unstable for infinitely many senders. Håstad, Leighton, and Rogoff [6] studied systems with a finite number of senders. They showed that binary exponential backoff is unstable for λ slightly larger than 0.567 even for a system with a finite number of senders, and that polynomial backoff protocol is stable for any $\lambda < 1$ and for any finite number of senders. Goldberg and MacKenzie [5] analyzed the backoff protocol for the multiple servers setting, showing that any superlinear polynomial backoff protocol is stable for any $\lambda < 1$. Our current work is motivated by the lower bound in Håstad, Leighton, and Rogoff [6] showing that long delays are inevitable in backoff-style protocols: they showed that the delay of any stable exponential or polynomial backoff protocol is at least polynomial in the total number of contenders. Following our work, Paterson and Srinivasan [11] recently gave a protocol with $O(1)$ expected delay assuming that the initial clock times of the senders are within a known bound of each other.

1.3. Our results. We follow the lead of [1, 6] here and make no unproven assumptions about the independence of the state of the system from one time step to the next, or between senders (many analyses in the queuing-theory literature do make such assumptions). We assume only that the generation of packets is independent between time steps and senders. Even for this case, complex dependencies arise between the transmissions at different senders and time steps. We focus on protocols with short packet delay. We first consider the case $m = 1$. Our main result (Theorem 1) is a protocol that ensures delay logarithmic in n, provided the arrival rate is no more than a fixed constant λ'. To our knowledge, this is the first protocol with sublinear delay (under an exact analysis) that is stable for a constant injection rate. Turning to the case $m > 1$, we present a protocol that achieves logarithmic delay provided that arrival rate at each sender and for each receiver is at most a fixed constant λ' (Theorem 2). Finally, we show (Theorem 3) that if every sender uses a class of protocols including backoff protocols, there is a fixed constant $\lambda_0 < 1$ such that if $\lambda > \lambda_0$, the delay must be $\Omega(n)$. Thus, in this class of protocols one cannot achieve full throughput and small delay simultaneously (whereas full throughput alone can be achieved by the polynomial backoff protocol that belongs to that class [6]).

2. Multiple access to one channel. In this section we consider the case $m = 1$. We show that there exists a positive constant λ_0 and a contention resolution protocol that is stable for any $0 < \lambda \leq \lambda_0$, with delay $O(\log n)$.

2.1. The protocol. Each sender in our protocol (Fig. 1) has a *transmission buffer* of size $O(\log n)$ and a *queue*. Packets awaiting transmission are stored either in the buffer or in the queue. Throughout the execution of the protocol a sender is in one of two states: a *normal state* or a *reset state*. Note that in the protocol, a transmission attempt may fail due to collision. The constants α and μ used in the protocol are fixed in the proof of Theorem 1 below.

Relation to practical protocols. The use of the queue and especially the reset state may appear to be somewhat artificial. We use these devices to cater to catastrophic events (e.g., every sender generates a packet in every one of n^{10} consecutive steps) that occur with extremely low but positive probabilities. In practice, such catastrophic events are handled by *dropping packets*: some packets are permanently erased from the system during such rare events. Rather than drop packets we invoke the emergency mechanism involving the queues and the reset state, while proving that the chance of resorting to these measures is extremely small.

$Count\_attempts(s)$ keeps a count of the number of times s tried to transmit a packet from its buffer in the $4\mu n \log n$ most recent steps.

$Failure\_counts(s)$ stores the failure rates in transmission attempts of packets from the buffer of s in the most recent $\mu \log n$ attempts.

$Random\_number()$ is a function that returns a random number uniformly chosen in the range $[0, 1]$, independent of the outcomes of previous calls to the function.

While in the normal state repeat:

1. Place new packets in the buffer.
2. Let X denote the number of packets in the buffer.
   If $Random\_number() \leq X/8\alpha \log n$ **then**
   (a) Try to transmit a random packet from the buffer.
   (b) Update $Count\_attempts(s)$ and $Failure\_counts(s)$.
   Else
   If $Random\_number() \geq 1 - 1/n^2$ **then** transmit the packet at the head of the queue.
3. **If** ($Count\_attempts(s) \geq \mu \log n$ **and** $Failure\_counts(s) > 5/8$), **or If** ($X > 2\alpha \log n$) **then**
   (a) Move all packets in the buffer to the end of the queue.
   (b) Switch to the reset state for $4\mu n^2 \log n + \gamma \log n$ steps.

While in the reset state repeat:

1. Append any new packets to the queue.
2. **If** $Random\_number() \leq 1/n^2$ **then** transmit the packet at the head of the queue.

FIG. 1. *Communication protocol for sender s.*

2.2. Analysis of the protocol. The performance of the protocol is summarized in the following theorem.

THEOREM 1. *There is a fixed constant $\lambda_0 > 0$, such that for any $\lambda \leq \lambda_0$ the above protocol is stable and the expected delay of each packet is $O(\log n)$.*

Proof. To simplify the analysis we assume, without loss of generality, that at each step each sender tries to transmit a packet from the queue with probability $1/n^2$ even if its queue is empty (in which case it tries to transmit an empty message). This assumption makes the process of transmitting from the buffers completely independent of the sizes of the queues at the beginning of that step.

Consider the n-vector of nonnegative integer whose ith component is the number of packets in the buffer of the ith sender. This vector defines a finite positive recurrent aperiodic Markov chain, which thus has a stationary distribution. Let X_t be a random variable counting the total number of packets at time t in all the buffers. By the condition tested in step 3 of the protocol, $X_t \leq 2\alpha n \log n + n$ for all t. The crux of the proof is to show that most of the time $X_t \leq 2\alpha \log n$, which guarantees short delays.

Throughout the analysis we use the following versions of the Chernoff bound [10]: let Z be the number of successes in k independent Bernoulli trials with probability p for success in each trial, then for $0 \leq \delta \leq 1$ $\mathbf{Pr}\{Z \leq (1-\delta)pk\} \leq e^{-\delta^2 pk/2}$, and $\mathbf{Pr}\{Z \geq (1+\delta)pk\} \leq e^{-\delta^2 pk/3}$. For $\delta > 1$ $\mathbf{Pr}\{Z \geq (1+\delta)pk\} \leq e^{-\delta \ln(1+\delta)pk}$. ∎

LEMMA 1. *Let $T = 4\mu n^2 \log n + \gamma \log n$ (the constant γ is determined in the proof of Claim 3). For any $x \le 2\alpha n \log n + n$ assumed by X_{t-T},*

$$\mathbf{Pr}[X_t > 2\alpha \log n \mid X_{t-T} = x] \le 1/n^{10}.$$

Proof. To prove the lemma, we begin with the following claim.

CLAIM 1. *If $X_{t-T} > 6\alpha \log n$, then with probability $1 - n^{-11}$ there is a step $\tau_1 \in [t - T, t - \gamma \log n]$ such that $X_{\tau_1} \le 6\alpha \log n$.*

Proof. We show that as long as the total number of packets in buffers exceeds $6\alpha \log n$, then in each interval of $4\mu n \log n$ steps at least one sender is very likely to switch to the reset state with high probability (and stays in that state for $4\mu n^2 \log n$ steps).

Consider the interval $[\tau_0, \tau_0 + 4\mu n \log n - 1]$ consisting of $4\mu n \log n$ steps. Let z_τ, $\tau \in [\tau_0, \tau_0 + 4\mu n \log n - 1]$ be a random variable defined as follows. If $X_\tau > 6\alpha \log n$, then z_τ equals the number of transmission attempts at this step; else $z_\tau = n$. Clearly $[z_\tau \mid z_{\tau_0}, z_{\tau_0+1}, \ldots, z_{\tau-1}]$ is stochastically lower-bounded[2] by a binomial distribution with expectation $(6\alpha \log n)/(8\alpha \log n)$, and $\sum_{\tau=\tau_0}^{\tau_0+4\mu n \log n-1} z_\tau$ is stochastically (lower-) bounded by a binomial distribution with expectation

$$\frac{(4\mu n \log n)(6\alpha \log n)}{8\alpha \log n} = 3\mu n \log n. \tag{1}$$

Thus, by the Chernoff bound, with probability at least

$$1 - e^{-\frac{2}{3}\mu n \log n}, \tag{2}$$

either $X_\tau \le 6\alpha \log n$ for some τ in this interval or there were at least $\mu n \log n$ attempts to transmit packets from buffers and at least one sender was involved in $\mu \log n$ or more attempts.

If there were at least $6\alpha \log n$ packets in buffers in a given step, then the success probability of an attempt at that step is at most

$$\left(1 - \frac{2\alpha \log n}{8\alpha \log n}\right)^2 \le 9/16. \tag{3}$$

(Since no sender has more than $2\alpha \log n$ packets in buffers, the "best" probability is when $6\alpha \log n$ packets are distributed equally among three senders.)

For a given sender s and given steps τ and τ' such that $\tau \ge \tau' \ge \tau_0$, let $y_\tau^{\tau',s}$ be a random variable defined as follows: $y_\tau^{\tau',s} = 1$ if sender s successfully transmitted a packet at time τ, $X_\tau > 6\alpha \log n$, and s had less than $\mu \log n$ transition attempts in the interval $[\tau', \tau]$; else $y_\tau^{\tau',s} = 0$. Let $\mathcal{H}_\tau$ describe the state of the system at all times before step τ. Clearly

$$\mathbf{Pr}[Y_\tau^{\tau',s} \mid \mathcal{H}_\tau] \le \frac{9}{16},$$

and $\mathbf{Pr}[\sum_{\tau=\tau'}^{\tau_0+4\mu n \log n-1} y_\tau^{\tau',s} \ge \frac{5}{8}\mu \log n]$ is upper-bounded by the probability that a binomial random variable with parameters $B(\mu \log n, \frac{9}{16})$ is at least $\frac{5}{8}\mu \log n$. Thus, using the Chernoff bound, with probability

$$1 - n(4\mu n \log n)e^{-\frac{1}{3}\frac{9}{16}(\frac{1}{9})^2 \mu \log n}, \tag{4}$$

[2]We say that distribution F is stochastically lower- (upper-) bounded by distribution G if for any x, $F(x) \ge G(x)$ (respectively, $F(x) \le G(x)$).

either $X_\tau \le 6\alpha \log n$ for some τ in that interval or no sender had a success rate of at least $\frac{5}{8}$ in a sequence of $\mu \log n$ attempts in that interval.

Combining the bounds in (2) and (4) we get that for each interval of $4\mu n \log n$ steps, with probability at least

$$1 - e^{-\mu \log n} - n(4\mu n \log n) e^{-\frac{1}{3}\frac{9}{16}(\frac{1}{9})^2 \mu \log n} \ge 1 - n^{-12}$$

(for sufficiently large constant μ), either $X_\tau \le 6\alpha \log n$ at some step of the interval or at least one sender switches to the reset state and moves all its packets from its buffer to its queue. Thus, with probability $1 - n^{-11}$, at some step $\tau_1 < t + 4\mu n^2 \log n$, $X_{\tau_1} \le 6\alpha \log n$. □

CLAIM 2. *Suppose that there is a positive integer L such that if $X_\tau \in [L, L + 2\alpha \log n]$ then the probability that a packet is delivered from a buffer at step τ is at least p for a constant $p > \lambda$. If $X_{\tau_1} \le L$, then*

$$\mathbf{Pr}[X_\tau \ge L + \alpha \log n \text{ for some step } \tau \in [\tau_1, \tau_1 + W]] \le W n^{-13}$$

for a sufficiently large constant α.

Proof. Since at most one message can be delivered from all buffers in each step, X_τ can decrease by at most one in each step. Thus, it is sufficient to prove that there is no time interval of length $\alpha \log n$ during which X_τ remains above L.

Consider an interval of $\alpha \log n$ steps, starting at step $\tau_2 \in [\tau_1, \tau_1 + W]$. Assume that $X_{\tau_2 - 1} \le L$. Let $\epsilon = (p - \lambda)/3\lambda$. The probability that more than $(1+\epsilon)\lambda\alpha \log n < 2\alpha \log n$ packets are placed in buffers in the interval is at most $n^{-13}/2$.

Let z_τ be a 0–1 random variable defined as follows: $z_\tau = 1$ if $X_\tau \le L$ or a packet is delivered from a buffer in step τ; else $z_\tau = 0$. Clearly, as long as $X_\tau \le L + 2\alpha \log n$, $\mathbf{Pr}[z_\tau = 1 \mid z_1, \ldots, z_{\tau-1}] \ge p$. Let $\delta = (p - \lambda)/3p$.

$$\mathbf{Pr}\left[\sum_{k=\tau_2}^{\tau_2 + \alpha \log n} z_k < (1 - \delta) p \alpha \log n\right] \le e^{-\frac{1}{3}\delta^2 p \alpha \log n} \le n^{-13}/2.$$

Since $(1 + \epsilon)\lambda\alpha \log n < (1 - \delta)p\alpha \log n$, the probability that $X_\tau > L$ for a given $\tau \in [\tau_2, \tau_2 + \alpha \log n]$ is at most n^{-13}, for any $p > \lambda$ and a sufficiently large constant α. □

CLAIM 3. *If $X_{\tau_1} \le 6\alpha \log n$ for some $\tau_1 \in [t - T, t - \gamma \log n]$, then with probability at least $1 - n^{-11}$, there is a step $\tau_2 \in [\tau_1, t]$ such that $X_{\tau_2} \le \alpha \log n$.*

Proof. As long as $X_\tau \in [\alpha \log n, 7\alpha \log n]$ the probability of a successful transmission from some buffer at time τ is at least p, where

$$p = \binom{X_\tau}{1} \frac{1}{8\alpha \log n} \left(1 - \frac{1}{8\alpha \log n}\right)^{X_\tau - 1} - \frac{n}{n^2} \ge \frac{X_\tau}{8\alpha \log n} - (\frac{X_\tau}{8\alpha \log n})^2 - \frac{n}{n^2} \ge \frac{1}{10}.$$

Let $\mathcal{E}_1$ denote the event: "$X_{\tau_1} \le 6\alpha \log n$, and there is a $\tau \in [\tau_1, \tau_1 + \gamma \log n]$ such that $X_\tau > 7\alpha \log n$."

By Claim 2, $\mathbf{Pr}[\mathcal{E}_1] \le n^{-13}\gamma \log n$ for any $\lambda < p$ and a sufficiently large constant α.

The expected number of new packets arriving in the time interval $[\tau_1, \tau_1 + \gamma \log n]$ is $\lambda\gamma \log n$. Let $\delta = (p - \lambda)/3\lambda$ and let $\mathcal{E}_2$ denote the event "More than $\lambda\gamma(1 + \delta) \log n$ packets arrived in the interval $[\tau_1, \tau_1 + \gamma \log n]$." Then $\mathbf{Pr}[\mathcal{E}_2] \le e^{-\delta^2 \lambda\gamma \log n/3} \le n^{-11}$ for a sufficiently large γ.

Let $\epsilon = (p-\lambda)/3p$ and define the event $\mathcal{E}_3$: "either $X_\tau \le \alpha \log n$ for some $\tau \in [t-\gamma\log n, t]$ or at least $(1-\epsilon)p\gamma\log n$ packets were delivered from the buffer at that interval." Then $\mathbf{Pr}[\mathcal{E}_3 \mid \bar{\mathcal{E}}_1] \ge 1 - e^{-\epsilon^2 p\gamma\log n/3}$.

Fix γ such that $\gamma(p-\lambda)/3 > 6\alpha$. Then the probability that there is no $\tau_2 \in [\tau_1, \tau_1+\gamma\log n]$ such that $X_{\tau_2} < \alpha\log n$ is at most

$$n^{-13}\gamma\log n + e^{-\lambda\gamma\delta^2\log n/3} + e^{-p\gamma\epsilon^2\log n/3} \le n^{-11}$$

for $\lambda < p$ and $\gamma \ge \max[\frac{18\alpha}{p-\lambda}, \frac{39}{\lambda\delta^2}, \frac{39}{p\epsilon^2}]$. □

CLAIM 4. *If $X_{\tau_1} \le \alpha\log n$ then the probability that there exists $\tau \in [\tau_1, \tau_1+T]$ such that $X_\tau > 2\alpha\log n$ is at most n^{-11}.*

Proof. If $X_\tau \in [\alpha\log n, 4\alpha\log n]$ then the probability of a successful transmission from a buffer at time τ is at least

$$p = \binom{X_t}{1}\frac{1}{8\alpha\log n}\left(1-\frac{1}{8\alpha\log n}\right)^{X_t-1} - \frac{n}{n^2}$$

$$\ge \frac{X_t}{8\alpha\log n} - \left(\frac{X_t}{8\alpha\log n}\right)^2 - \frac{n}{n^2} \ge \frac{1}{10}.$$

Using Claim 2, the probability of having $2\alpha\log n$ packets in buffers in the interval is at most Tn^{-13} for any $\lambda < p$ and a sufficiently large constant α. □

To conclude the proof of Lemma 1 we combine the error probabilities of the three lemmas above to show that for any value x,

$$\mathbf{Pr}[X_t > 2\alpha\log n \mid X_{t-T} = x] \le 3n^{-11} \le n^{-10}.$$ □

Comment. The statement of Lemma 1 implies that in the steady state, almost always $X_t \le 2\alpha\log n$. This observation is not used directly in the proof but gives a good intuition for the long-term behavior of the protocol.

We turn to the analysis of the queues. We focus on the performance of the protocol after the first $T_0 = 4\mu n^2\log n + \gamma\log n + 4\mu n\log n$ steps. We assume that at time 0 all queues are empty (this assumption affects only the length of the initial segment T_0).

CLAIM 5. *Let $t \ge T_0$. The probability that any sender switches to the reset state at step t is at most n^{-8}.*

Proof. If $X_\tau \le 2\alpha\log n$, then the probability of a success in a transmission attempt at step τ is at least $p = 1 - (2\alpha\log n)/(8\alpha\log n) - (n/n^2) \ge 3/4 - 1/n$.

For a given sender s and given steps $\tau \ge \tau' \ge t - 4\mu n\log n$, let $y_\tau^{\tau',s}$ be a random variable defined as follows: $y_\tau^{\tau',s} = 1$ if sender s successfully transmitted a packet at time τ, $X_\tau \le 2\alpha\log n$, and s had less than $\mu\log n$ transition attempts in the interval $[\tau', \tau]$; else $y_\tau^{\tau',s} = 1$. Let $\mathcal{H}_\tau$ describe the state of the system at all times before step τ. Clearly

$$\mathbf{Pr}[Y_\tau^{\tau',s} \mid \mathcal{H}_\tau] \le \frac{1}{4} + \frac{1}{n},$$

and $\sum_{\tau=\tau'}^{\tau_0+4\mu n\log n-1} y_\tau^{\tau',s}$ is stochastically upper-bounded by a binomial distribution with expectation $(\frac{1}{4}+\frac{1}{n})\mu\log n$. Thus, by the Chernoff bound with probability

$$1 - n(4\mu n\log n)e^{-\frac{1}{3}(\frac{1}{4}+\frac{1}{n})(\frac{1}{2})^2\mu\log n}, \tag{5}$$

either $X_\tau > 2\alpha \log n$ for some τ in that interval or no sender had a success rate less than $\frac{5}{8}$ in a sequence of $\mu \log n$ attempts in that interval. By Lemma 1, $\mathbf{Pr}[X_\tau > 2\alpha \log n]$ at any time $\tau \in [t-4\mu n \log n, t]$ is at most $4\mu n^{-9}(\log n+2)$, which together with the above bound proves the lemma. □

LEMMA 2. *Consider a packet that was generated at time $t \geq T_0$.*

1. *The expected number of steps a given packet spends in a buffer is $O(\log n)$.*

2. *The probability that a given packet is delivered from the buffer is at least $1 - n^{-5}$.*

Proof. Assume that the packet was generated at sender s.

Consider an interval of n steps throughout which (1) s is never in the reset state, and (2) $X_\tau \leq 2\alpha \log n$. Let $p = 1 - (2\alpha \log n)/(8\alpha \log n) - n/n^2$ as in the proof of Claim 5. If the packet was in the buffer of sender s at the start of the interval, the probability that it is not delivered until the end of the interval is at most $(1 - p/8\alpha \log n)^n$. During the interval the expected number of steps between two attempts to transmit the packet is $8\alpha \log n$, and the expected number of attempts until the packet is delivered is $1/p$. Thus, if the packet is delivered during the interval, by Wald's identity [2] the expected delay of the packet from the start of that interval is at most $(8\alpha \log n)/p$.

The probability that the interval $[t, t+n]$ satisfies conditions (1) and (2) is (by Lemma 1 and Claim 5) at least $1-(4\mu n^2 \log n+n)n^{-8}-n^{-9}$, and with this probability, any subsequent interval of $4\mu n^2 \log +n$ steps has a segment of n steps that satisfies the above conditions. Thus, the expected number of steps a packet spends in the buffer is at most

$$\frac{8\alpha \log n}{p} + \sum_{k \geq 1} k(n + 4\mu n^2 \log n) \left((4\mu n^2 \log n + n)n^{-8} + n^{-9} + \left(1 - \frac{p}{8\alpha \log n}\right)^n \right)^k$$

$$= \frac{8\alpha \log n}{p} + o(1).$$

(The above estimate ignores the possibility that a packet is moved to the queue before it is delivered from the buffer. This can only decrease the expected number of steps a packet spends in the buffer.)

The probability that the packet is delivered from the buffer is bounded from below by the probability that the first n steps following the creation of the packet satisfy conditions (1) and (2) and the packet is delivered in that time. This probability is at least

$$1 - (4\mu n^2 \log n + n)n^{-8} - n^{-9} - \left(1 - \frac{p}{8\alpha \log n}\right)^n \geq 1 - n^{-5}. \qquad \square$$

LEMMA 3. *Consider a packet that was generated at time $t > T_0$. Given that the packet enters the queue, the expected length of time it spends in the queue is $O(n^4 \log n)$.*

Proof. Assume that the packet was generated at sender s. When a sender switches to the reset state it moves all the (up to $2\alpha \log n + 1$) packets in its buffer to its queue. In addition, all the (up to $4\mu n^2 \log n + \gamma \log n$) packets it receives in the next $4\mu n^2 \log n + \gamma \log n$ steps are placed in the queue. Let $D = 4\mu n^2 \log n + \gamma \log n + 2\alpha \log n + 1$.

When a queue is not empty and the total number of packets in all buffers in the system is at most $2\alpha \log n$, a sender succeeds in transmitting a packet from its queue in each step with probability at least

$$p = \frac{1}{n^2}\left(1 - \frac{1}{8\alpha \log n}\right)^{2\alpha \log n}\left(1 - \frac{1}{n^2}\right)^{n} \geq \frac{1}{2n^2}.$$

Let τ_1 be the first step that X_t exceeds $2\alpha \log n$ (i.e., $X_{\tau_1 - 1} \leq 2\alpha \log n$ and $X_{\tau_1} > 2\alpha \log n$). Let τ_i, $i > 1$, be the first step that X_t exceeds $2\alpha \log n$ after step $\tau_{i-1} + T$ ($T = 4\mu n^2 \log n + \gamma \log n$ as defined in Lemma 1). We say that an interval $[\tau_i, \tau_{i+1}]$ is *good* (with respect to sender s) if either the queue of s was empty at some step in that interval or at least $2D$ packets were delivered from the queue. Note that no more than D packets can enter the queue in each interval, since a sender cannot switch twice to a reset state in the same interval.

By Lemma 1 the probability that an interval does not have a segment of $L = 4n^2 D$ steps in which $X_t \leq 2\alpha \log n$, conditioning on all events before that segment is bounded by Ln^{-10}. The probability that in L steps in which $X_t \leq 2\alpha \log n$, fewer than $2D$ packets are transmitted, is bounded by $e^{-\frac{1}{3}\frac{1}{2n^2}L} \leq e^{-\frac{2D}{3}}$. Thus, in a given interval the probability that either $Z_t = 0$ at some point in the interval or at least $2D$ packets are delivered from the queue is at least $1 - Ln^{-10} - e^{-\frac{2D}{3}} \geq 1 - \frac{1}{n^5}$, and for subsequent intervals these events are independent.

The probability that there are iD packets in the queue at time t is bounded by the probability that there are fewer than $(k-i)/2$ *good* intervals in the k most recent intervals, for some $k \geq i$. This probability is bounded by $\sum_{k \geq i} \binom{k}{(k+i)/2}(\frac{1}{n^5})^{(k+i)/2} \leq 2(\frac{e}{n^5})^i$, and the expected number of packets in the queue at time t is bounded by $\sum_{i \geq 0} iD(\frac{e}{n^5})^i = O(1)$. Our packet, however, arrives with another $D = O(n^2 \log n)$ packets, and the expected number of steps to send a packet from the head of the queue is $O(n^2)$. Thus, the expected time until all these packets are transmitted is $O(n^4 \log n)$. □

By Claim 2 the expected time a packet spends in the buffer in $O(\log n)$. With probability $O(n^{-5})$ the packet is transferred to the queue, where its expected delay is $O(n^4 \log n)$. Thus, the expected delay of a given packet is $O(\log n)$. □

3. The protocol for n senders and m receivers. In our protocol each sender runs m "one-channel protocols" simultaneously. Each sender keeps a counter of transmission failures for each of the receivers. When this count is too large the sender switches to the reset state only with respect to packets bound for that receiver.

The "one-channel protocol" requires that if there are X packets in the buffer of sender s, the sender tries to transmit a packet from the buffer with probability $X/8\alpha \log n$. To accommodate the m protocols simultaneously in one sender we modify step 2 in the "one-channel protocol" as follows:

2. Let $Y^i(s)$ denote the number of packets with destination i in the buffer of sender s. Let $Y(s) = \sum_{i=1}^{m} Y^i(s)$.
If $Y(s) > 2\alpha \log n$ **then** move all packets to the queue,
else with probability $Y(s)/8\alpha \log n$ transmit a random packet from the buffer.

It remains to show that the probability that a sender has more than $2\alpha \log n$ packets in the buffer is small. Thus, moving them to the queue does not significantly change the performance of the protocol.

LEMMA 4. *Let $Y_t(s)$ denote the total number of packets in the buffer of sender s at time t. In the stationary distribution,*

$$\mathbf{Pr}[\exists s \text{ such that } Y_t(s) > 2\alpha \log n] \leq n^{-7}.$$

Proof. As in the proof of Lemma 1 we show that there exists a constant γ such that if $T = 4\mu n^2 \log n + \gamma \log n$, then for any value y of $Y_{t-T}(s)$,

$$\mathbf{Pr}[Y_t(s) > 2\alpha \log n \mid Y_{t-T}(s)] = y \leq n^{-8}.$$

Let X_τ^i denote the total number of packets with destination i in the buffers of all senders. Let $\mathcal{E}$ denote the event: "For any $1 \leq i \leq m$ and $t - \beta \log n \leq \tau \leq t$, $X_\tau^i \leq 2\alpha \log n$."

By Lemma 1, regardless of the state of the system at time $t-T$, $\mathbf{Pr}[\mathcal{E}] \geq 1 - n^{-9}$ (since there are no more than n protocols).

The total injection rate of sender s is at most λ. Conditioning on the event $\mathcal{E}$, the probability that s successfully delivers a message at time τ, $\tau \in [t - \beta \log n, t]$ when $Y_\tau(s) > \alpha \log n$ is at least $p = \frac{\alpha \log n}{8\alpha \log n}(1 - 1/8\alpha \log n)^{2\alpha \log n}(1 - 1/n^2)^n \geq 1/20$. As in the proof of Claim 3 we show that with probability $1 - n^{-9}$ there is a step $\tau \in [t - \beta \log n, t]$, such that $Y_t(s) < \alpha \log n$, and using Claim 2 we show that once the number of packets is below $\alpha \log n$ the probability that this number reaches $2\alpha \log n$ before time T is at most n^{-8}. □

When a sender has $2\alpha \log n$ packets in its buffer it transfers them to its queue. Thus, the expected additional contribution to the queue from the modified protocol is $O(n^{-8} \log n)$ per step. This contribution does not change the delivery time of packets in the queue or the overall performance of the protocol significantly.

THEOREM 2. *There is a constant $\lambda_0 > 0$ such that the above protocol is stable for any $0 \leq \lambda \leq \lambda_0$ and the delay is $O(\log n)$.*

4. The lower bound. We state and prove the lower bound for the case $m = 1$; a similar bound holds for the general case. We consider n senders all running the same protocol. The protocol running at a sender does not change state unless (1) it attempts a transmission or (2) it generates a packet. We further assume that a sender is in a unique idle state when its queue is empty. (Thus, a sender does not keep information on the state of the channel when it does not have packets to transmit.) For any such protocol governed by a (probabilistic) automaton we show that there is a $\lambda_0 < 1$ such that for any injection rate $\lambda > \lambda_0$ the delay is $\Omega(n)$. Note that the class of protocols includes all variants of backoff protocols.

THEOREM 3. *If all senders follow protocols from the above class, there is a fixed constant $\lambda_0 < 1$ such that if $\lambda > \lambda_0$, the expected delay is $\Omega(n)$.*

Proof. Let $\lambda_i = \lambda/n$ for all i. Given a system of senders, we say that $X_t = j$ if the total number of packets pending transmission at all senders at step t is j. If the expected delay is bounded then the system has a steady state distribution. Consider the steady state distribution of X_t. We distinguish between two cases.

Case 1. In the steady state distribution, $\mathbf{Pr}[X_t] \geq n/2 \geq 1/2$. Since the system can transmit no more than one packet per step the expected delay in this case in clearly $\Omega(n)$.

Case 2. In the steady state distribution $\mathbf{Pr}[X_t] < n/2 > 1/2$. □

LEMMA 5. *If $X_t < n/2$, the probability of a collision at step $t + n/3$ is at least a constant q.*

Proof. If $X_t < n/2$, we have at least $n/2$ senders having no packets pending transmission at time t (call these *empty senders*). A protocol in our class defines a probability distribution p_i on the nonnegative integers i such that when an empty sender generates a packet at step t, it transmits it at step $t+i$ with probability p_i unless it changes its behavior (due to the generation of a packet) in the interval $[t, t+i]$.

An empty sender that generates a packet at time τ does not change the probabilities p_i in the interval $(\tau, \tau + n/3]$ unless it generates another packet in the interval. The probability that it generates such a packet in the interval $(\tau, \tau + n/3]$ is at most $(1-\lambda/n)^{n/3}$, which is at most a constant α. Thus the empty sender does transmit the packet at time $\tau + i$ with probability at least $(1-\alpha)p_i$. Clearly, $\sum_{i=0}^{n/3} p_i$ must be at least a constant β (for otherwise the delay is already at least $\alpha(1-\beta)n/3$). If at least $n/2$ senders are empty at time t, the probability that any one of them transmits at time $t + n/3$ is thus at least

$$(1-\alpha)\sum_{k=0}^{n/3} \frac{\lambda}{n} p_k \geq (1-\alpha)\beta\lambda/n.$$

Then, the probability of a collision at time $t + n/3$ is at least a constant q. ∎

Thus, if in the steady state distribution, $\mathbf{Pr}[X_t] < n/2 > 1/2$, then the probability of a collision in a given step in the steady state is at least a constant $q/2$. Since the expected number of new packets arriving in each step is λ and the system can transmit no more than one packet per step, the system cannot be stable if $\lambda > 1 - q/2$.

Acknowledgment. We thank Aravind Srinivasan for helpful comments.

REFERENCES

[1] D. Aldous, *Ultimate instability of exponential back-off protocol for acknowledgment based transmission control of random access communication channels*, IEEE Trans. Inform. Theory, 33 (1987), pp. 219–223.

[2] W. Feller, *An Introduction to Probability Theory and Its Applications, Vol.* II, John Wiley, New York, 1968.

[3] J. Goodman, A.G. Greenberg, N. Madras, and P. March, *Stability of binary exponential backoff*, J. Assoc. Comput. Mach., 35 (1988), pp. 579–602.

[4] A.G. Greenberg, P. Flajolet, and R.E. Ladner, *Estimating the multiplicities of conflicts to speed their resolution in multiple access channels*, J. Assoc. Comput. Mach., 34 (1987), pp. 289–325.

[5] L.A. Goldberg and P.D. MacKenzie, *Analysis of practical backoff protocols for contention resolution with multiple servers*, in Proceedings of the 7th Annual Symp. on Discrete Algorithms, SIAM, Philadelphia, 1996, pp. 554–563.

[6] J. Håstad, T. Leighton, and B. Rogoff, *Analysis of backoff protocols for multiple access channels*, in Proceedings of the 19th ACM Symp. on Theory of Computing, 1987, pp. 241–253.

[7] F.P. Kelly, *Stochastic models of computer communication systems*, J. Roy. Statist. Soc. B, 47 (1985), pp. 379–395.

[8] P.D. MacKenzie, C.G. Plaxton, and R. Rajaraman, *On contention resolution protocols and associated probabilistic phenomena*, in Proceedings of the 26th ACM Symp. on Theory of Computing, 1994, pp. 153–162.

[9] R. Metcalfe and D. Boggs, *Ethernet: Distributed packet switching for local computer networks*, Comm. ACM, 19 (1976), pp. 395–404.

[10] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, New York, 1995.

[11] M. Paterson and A. Srinivasan, *Contension resolution with bounded delays*, in Proceedings of the 36th Annual ACM Symp. on Foundations of Computer Science, 1995, pp. 104–113.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 720–732

ASYMPTOTICALLY OPTIMAL ELECTION ON WEIGHTED RINGS*

LISA HIGHAM† AND TERESA PRZYTYCKA‡

Abstract. In a network of asynchronous processors, the cost to send a message can differ significantly from one communication link to another. In such a setting, it is desirable to factor the cost of links into the cost of distributed computation. Assume that associated with each link is a positive *weight* representing the cost of sending one message along the link, and the cost of an algorithm executed on a *weighted* network is the sum of the costs of all messages sent during its execution. We determine the asymptotic complexity of distributed leader election on a weighted unidirectional asynchronous ring assuming this notion of cost, by exhibiting a simple algorithm and a matching lower bound for the problem for any collection of edge weights. As a consequence, we see that algorithms designed for unweighted rings are not in general efficient for the weighted case.

Key words. distributed election, weighted ring, message complexity, asynchronous network

AMS subject classifications. 68Q22, 68M10

PII. S0097539795288799

1. Introduction.

Consider a network of asynchronous processors that communicate via message passing. The typical measure of the cost of a distributed algorithm on such a network is the number of messages sent. This measure assumes that the cost of sending a message along any link is equal to 1. In practice, the cost of sending a message may depend upon the link that the message traverses. This motivates the study of distributed algorithms where the cost of transmitting a message over a link is factored into the communication complexity of the algorithm. Awerbuch, Baratz, and Peleg [1] called this notion of communication complexity "cost-sensitive analysis." A *weighted network* is a network of processors where each link e of the network has associated with it a positive weight $w(e)$, which is the *cost* of sending a message along link e. The cost of a distributed algorithm for a given weighted network and input is the maximum, over all message delay patterns, of the sum of the costs of all message traffic that occurs while executing the algorithm on that input. When designing a distributed algorithm for a weighted network we try to limit the message traffic over heavy edges.

In this paper, we study the weighted cost of leader election when the network topology is an asynchronous unidirectional weighted ring with distinct identifiers. The leader election problem is to design a distributed algorithm that distinguishes exactly one processor from among all the processors of the network as a unique processor called the *leader*. Leader election on asynchronous unweighted rings has been very well studied. Early papers by LeLann [8], Chang and Roberts [2], and Hirschberg and Sinclair [7] solved the unidirectional and bidirectional version for rings with identifiers

*Received by the editors July 7, 1995; accepted for publication (in revised form) June 10, 1996; published electronically August 4, 1998.

http://www.siam.org/journals/sicomp/28-2/28879.html

†Department of Computer Science, University of Calgary, Calgary T2N 1N4, AL, Canada (higham@cpsc.ucalgary.ca). This research was carried out in part while visiting the University of Odense, Denmark. The support of the Natural Sciences and Engineering Research Council of Canada is also gratefully acknowledged.

‡Department of Mathematics and Computer Science, Odense University, Denmark. Current address: Department of Biophysics, the John Hopkins School of Medicine, Baltimore, MD 21205 (przytyck@grserv.med.jhmi.edu). This research was carried out in part while the author was visiting the University of Calgary, Canada.

using at most $O(n^2)$ and $O(n \log n)$ messages, respectively. Then, in 1982, Peterson [10] and Dolev, Klawe, and Rodeh [3] independently solved the unidirectional version of the problem using $O(n \log n)$ messages. By the results of Pachl, Korach, and Rotem [9] these algorithms are asymptotically optimal. Some effort has been made to reduce the constant [10, 3, 6] leading to the constant 1.271 [6]. Research has also established the possibility and complexity of leader election on rings in which processors lack distinct identifiers. (In this case, randomization is required.) Also, there has been substantial work generalizing and strengthing the lower bound for election on rings and other networks under a variety of assumptions about the model. See [5] for a list of research that addresses algorithms and lower bounds related to the leader election problem.

Running an algorithm designed for an unweighted network on a weighted network will, in general, not be cost efficient. Let W be the sum of the weights of all links of a weighted ring. Peterson's classical algorithm [10], when executed on the weighted ring, will incur a cost of $\Omega(W \log n)$. In fact, all other known leader election algorithms have the same bound on complexity. The results of this paper show that this is not optimal. What is needed is a new technique for breaking the symmetry of the ring based on the weights of the edges. Such a technique is developed in this paper.

Let $\mathcal{R}$ be any ring with n_i edges having weight in $(2^{i-1}, 2^i]$. We present an algorithm for the leader election problem on unidirectional weighted rings that has cost $O(\sum_{n_i \geq 1} n_i 2^i \lg(n_i+1))$ on $\mathcal{R}$. We show that this algorithm is optimal in the following sense: given a multiset W of weights where n_i weights are in the interval $(2^{i-1}, 2^i]$ and a leader election algorithm A, we can design a ring $\mathcal{R}$ with edge weights equal to the set W such that the weighted message cost of A on $\mathcal{R}$ is $\Omega(\sum_{n_i \geq 1} n_i 2^i \lg(n_i+1))$. The matching lower bound of our cost-sensitive analysis establishes that our algorithm is optimal in the "universal" sense of Garay, Kutten, and Peleg [4]—that is, that the algorithm is optimal for any collection of weights. This universality ensures that the parameters that determine the complexity of election on a unidirectional weighted ring have been precisely identified. This is the first lower bound that applies in this strong, universal sense to cost-sensitive analysis of any problem on any weighted network.

Our algorithm for weighted rings is in some sense a generalization of the basic algorithm for the unweighted case [6]. The perspective of this algorithm facilitates an extension to the weighted case. However, the analysis in the weighted case requires completely different techniques. The new contribution of our lower bound is that it explicitly incorporates the weights into the result. The basic idea to achieve this bound is derived from the work of Pachl, Korach, and Rotem [9]. However, we need to adjust the technique to overcome the complications introduced by weighted links.

The optimal algorithm is presented in section 2, its analysis, in section 3, and the lower bound, in section 4.

2. The leader election algorithm for weighted rings.

2.1. Algorithm description. Although, in the literature, the leader election algorithms for unweighted rings are presented in a variety of ways, there is a high-level perspective that can be used to describe them all (see [5]). Initially each processor creates an *envelope* containing a *label* set to its own identifier, a *round number* (or sometimes a round parity bit), and possibly additional information, and forwards the envelope to its neighbor. Upon receipt of an envelope, a processor applies a *casualty test*, which compares the contents of the envelope with the processor's stored information, to determine whether or not to destroy the envelope. If the receiving

processor determines not to destroy the envelope, it applies a *promotion test*, to determine whether or not to increment the round number. It then updates the content of the envelope and its own information as required and forwards the envelope to its neighbor. Eventually only one envelope remains and a leader is elected. The various algorithms differ in four ways: the content of an envelope in addition to label and round number, the local information stored by each processor, the specification of the casualty test, and the specification of the promotion test.

In both the basic algorithm [6], which we refer to as the MIN-MAX algorithm, and our algorithm for weighted rings, called WEIGHTED ELECT, the label of an envelope is never changed. In MIN-MAX, each envelope contains only its label and a round number initialized to 1. Each processor stores the label and the round of the last envelope it sent. The casualty test is simply: the envelope and the receiving processor have the same round number and this round number is odd (respectively, even) and the label of the envelope is larger (respectively, smaller) than that stored by the processor. The promotion test is simply: the envelope and the receiving processor have the same round number.

One way to visualize MIN-MAX is to imagine that execution proceeds in rounds. In an odd round any envelope that directly follows an envelope with label smaller than its own label is destroyed, while in an even round any envelope that directly follows an envelope with a larger label is destroyed. Notice that in MIN-MAX, as well as in other election algorithms for unweighted rings, in every round (or sometimes in every second round) message traffic covers every link of the ring. One central idea in WEIGHTED ELECT is to accelerate processing of envelopes that have travelled a large weighted distance by promoting them to a higher round as soon as they incur a sufficient weighted cost. Algorithm WEIGHTED ELECT can be thought of as combining MIN-MAX with this idea of "early promotion by weighted distance." We will see that by using early promotion, message traffic does not necessarily cover every link in each round, thus reducing the weighted distance an envelope travels before the algorithm terminates.

The intuition is to have a processor p adopt a high round number if it sends a message over a heavy link. This causes p to destroy envelopes with lower round number that it later receives, and thus the high cost of those envelopes traveling the heavy link is avoided. It is safe for p to destroy those envelopes, since the message that p sends over the heavy link carries with it a high round number, and so can only be destroyed by processors that have adopted even higher round numbers than p. Of course, this idea must be combined with some mechanism such as the MIN-MAX algorithm, to take care of the case when many successive links have similar weights.

For algorithm WEIGHTED ELECT, in addition to the *label*, each envelope contains a *round* and a *credit*. Both are initialized as a function of the weight of the link adjacent to the processor that creates the envelope. The initial credit is proportional to this weight and the initial round number is the logarithm of this weight. The label of an envelope remains unchanged as long as the envelope survives, whereas the round and credit are adjusted during the course of the algorithm. Throughout the algorithm, each processor stores the label and the round of the last envelope that it sent. The casualty test for WEIGHTED ELECT is: the round number of the received envelope is less than that of the last envelope sent, or the casualty test of MIN-MAX holds. If an envelope is not destroyed then it may be promoted, resulting in an increased credit and a larger round. The promotion test for WEIGHTED ELECT is: the credit is less than the weight of the outgoing edge or the promotion test of MIN-MAX holds. For

```
Processor(proc-id, adj_wt):

    id ⟵ proc-id ; p ⟵ ⌈lg adj_wt⌉ ; t ⟵ 0 ; cnt ⟵ 2^(p+1) ;
    fwd_id ⟵ −∞ ; fwd_p ⟵ −1 ; fwd_t ⟵ 0 ;

    repeat
        if not Casualty-test then
            if Promotion-test then
                t ⟵ (t+1) mod 4 ;
                if t = 0 then p ⟵ p+1 ;
                cnt ⟵ 2^(p+t+1) ;
            fi
            fwd_id ⟵ id ; fwd_p ⟵ p ; fwd_t ⟵ t ;
            send(id, p, t, cnt − adj_wt) ;
        fi
        receive(id, p, t, cnt) ;
    until Leader-test .
```

FIG. 1. *Algorithm* WEIGHTED ELECT.

any surviving envelope (whether promoted or not) the processor reduces its credit by the weight of its adjacent edge before sending the envelope forward.

The complete protocol for WEIGHTED ELECT is given in Figure 1. The protocol for each processor is parameterized by its identifier (*proc-id*) and the weight of its outgoing edge (*adj_wt*). Four consecutive rounds of WEIGHTED ELECT are grouped together to form a *phase*; hence round r is represented by an ordered pair (p, t), where p is the phase number, $t \in \{0, 1, 2, 3\}$, and $r = 4 * p + t$.

The pseudocode assumes the following three tests that are employed when an envelope containing label *id*, round (p,t), and credit *cnt* arrives at a processor that has recorded a label *fwd_id* and a round (*fwd_p*,*fwd_t*) and has an outgoing edge with weight *adj_wt*.

Casualty-test
($p <$ *fwd_p*) or
((p,t) = (*fwd_p*,*fwd_t*) and $t \in \{1, 3\}$ and *id* > *fwd_id*) or
((p,t) = (*fwd_p*,*fwd_t*) and $t \in \{0, 2\}$ and *id* < *fwd_id*).

Promotion-test
((p,t) = (*fwd_p*,*fwd_t*) and $t \in \{1, 3\}$ and *id* < *fwd_id*) or
((p,t) = (*fwd_p*,*fwd_t*) and $t \in \{0, 2\}$ and *id* > *fwd_id*) or
($p >$ *fwd_p* and *cnt* < *adj_wt*) .

Leader-test *id* = *fwd_id*.

2.2. Correctness of WEIGHTED ELECT**.** Correctness of WEIGHTED ELECT follows immediately after establishing:

safety: the algorithm never deletes all message envelopes;

progress: if there is more than one envelope then after a finite number of messages the number of envelopes is reduced; and

correct termination: the algorithm elects a leader exactly when one envelope remains.

Because the ring is unidirectional and the algorithm is deterministic and message-driven with messages processed in first-in-first-out order, the messages received by each processor and the order in which each processor processes its messages is entirely determined by the initial configuration of identifiers and edge weights. Thus the scheduler is powerless to influence the outcome of the computation. We emphasize that for *message-driven algorithms on unidirectional rings*, correctness and complexity

under any fixed scheduler implies correctness and complexity under all schedulers. Thus without loss of generality we assume a scheduler that proceeds by the round number of the envelopes. That is, an envelope with a given round number is not delivered until there does not exist an envelope of smaller round number. (This *round-driven* scheduler exists because WEIGHTED ELECT ensures that the round number of an envelope never decreases and the casualty test guarantees that the sequence of envelopes traveling any edge have nondecreasing round number.)

Suppose, contrary to safety, that some execution of WEIGHTED ELECT removes all envelopes under the round-driven scheduler and let (p, t) be the maximum round achieved. Suppose t is odd, and let S be the set of identifiers in envelopes that achieve round (p, t). According to Casualty-test, an envelope in round (p, t) with identifier i can only be destroyed by meeting a processor that either (1) last forwarded an envelope with round larger than (p, t), or (2) last forwarded an envelope with round equal to (p, t) and identifier less than i. Since (p, t) is the maximum round, case (1) is impossible. Furthermore, in case (2), the envelope in S with minimum identifier cannot be destroyed. So the envelope will be eventually promoted and its round increases contradicting that (p, t) was the maximum round. A symmetric argument applies if t is even.

Suppose, contrary to progress, that after some point, $k \geq 2$ envelopes remain alive under the round-driven scheduler. Then eventually, say, in round (p, t), each of these envelopes will receive a credit at least as large as the weight of the ring. At this point each envelope has a large enough credit to allow it to travel to the processor that promoted the envelope that precedes it. Since all undestroyed envelopes have the same round number, if t is odd (respectively, even) the envelope with maximum label (respectively, minimum label) must be destroyed, contradicting that no more envelopes are destroyed.

The algorithm cannot prematurely elect a leader because a processor will receive an envelope with *id* equal to its *fwd_id* if and only if there are no other envelopes, thus passing the Leader-test and confirming correct termination.

3. Message complexity of WEIGHTED ELECT**.** We first introduce some definitions and notation. The pth *phase* consists of all the message traffic of envelopes with round (p, t) for $t \in \{0, 1, 2, 3\}$. Since algorithm WEIGHTED ELECT never changes the label of an envelope for the duration of its existence, we use *envelope* a as an abbreviation for the envelope with label a. For an envelope a in phase p, let $\text{host}_p(a)$ denote the processor that promoted the envelope to phase p (that is, from round $(p-1, 3)$ to round $(p, 0)$), or the processor that created the envelope if it is initialized with phase p. The *weighted distance* from processor x to processor y, denoted $\delta(x, y)$, is the sum of the weights of all links between processor x and processor y, traveling in the direction of the ring.

As discussed in section 2, the scheduler cannot influence the communication complexity in a message passing unidirectional ring. Therefore, for simplicity, assume as before the round-driven scheduler, which delivers all envelopes in order of increasing round number. Under this scheduler, all undestroyed envelopes either *participate* in the round, say (p, t), or just *exist* in round (p, t) because they have been created with round number $(q, 0)$ where $q > p$ but have not yet been delivered across any link. Envelope b is the *immediate predecessor in phase* p of envelope a if, when all participating envelopes are in round $(p, 0)$ and all existing envelopes are in phase p or greater, the first envelope encountered after envelope a, traveling in the direction of the ring, is envelope b. Let envelope b be the immediate predecessor of envelope a in

phase p and suppose b is in phase $q \geq p$. Then the *horizon of envelope a in phase p* is $\delta(\text{host}_p(a), \text{host}_q(b))$.

Let n_i be the number of links with weight in $(2^{i-1}, 2^i]$ in an asynchronous ring with distinct identifiers. Let d_p denote the number of envelopes that participate in phase p. An envelope that participates in phase p is *sparse in phase p* if its horizon is greater than 2^p; otherwise, it is *dense*. Let s_p denote the number of envelopes that are sparse in phase p. Let J denote the number of phases until there are at most three remaining envelopes when WEIGHTED ELECT is run on the ring. Notice that after J phases, there can be at most three more passes of message traffic on the ring, so it suffices to bound the weighted message complexity for the first J phases.

The next three lemmas allow us to bound the number of sparse envelopes and the total number of envelopes participating in each phase as a function of the weights on the ring.

LEMMA 3.1. *The number s_p of sparse envelopes that participate in phase p satisfies $s_p \leq \sum_{i=0}^{p} n_i 2^{i-p}$.*

Proof. In phase p, any processor whose outgoing edge has weight greater than 2^p is host of an envelope in a phase $q > p$. Thus any envelope whose weighted distance to such a processor is less than 2^p cannot be sparse. The total weight of edges with weight at most 2^p is $\sum_{1=0}^{p} n_i 2^i$. These edges can accommodate at most $(\sum_{1=0}^{p} n_i 2^i)/2^p$ envelopes that are separated by a distance of at least 2^p. Thus, the number of sparse envelopes that participate in phase p is at most $\sum_{i=0}^{p} n_i 2^{i-p}$. ∎

We expect each pair of successive rounds in a phase to reduce the number of dense envelopes by at least one half. This is because in the first round any envelope meeting a processor in the same round with *fwd_id* smaller than its own *id* is eliminated, and in the next round any envelope meeting a processor in the same round with *fwd_id* larger than its own is eliminated. This is made precise in the next lemma.

LEMMA 3.2. *For $p < J$ the number d_p of envelopes that participate in phase p satisfies the recurrence:*

$$d_0 = n_0, \quad d_{p+1} \leq \frac{d_p - s_p}{4} + n_{p+1} + s_p, \qquad p \geq 0.$$

Proof. The proof relies on the following observation.

FACT 3.3. *Let x be an envelope that exists in phase $p+1$. Assume that at the beginning of phase p, x is immediately followed by k dense envelopes. Then the $\min\{k, 3\}$ dense envelopes that immediately follow x do not survive to phase $p+1$.*

The fact holds because consecutive dense envelopes have enough credit to reach the host of the next envelope with the same round number. By applying the min-max comparison for four consecutive rounds it is easily checked that if x survives for four rounds, the $\min\{k, 3\}$ dense envelopes that follow x must be eliminated.

Consider a maximal chain of dense envelopes in round $(p, 0)$. Suppose there is a nondense envelope s (a sparse envelope or an envelope with higher phase) that immediately precedes the leading dense envelope of this chain. If y is a dense envelope in this chain that survives to phase $p+1$ and is followed by at least three dense envelopes then, by Fact 3.3, we can attribute three eliminated envelopes to y. Suppose y is followed by fewer than three dense envelopes. If s exists in phase $p+1$ then we attribute to y the three eliminated envelopes that, by Fact 3.3, follow s at the beginning of phase p. If s does not survive to phase $p+1$ then in our count of surviving envelopes we can count y as eliminated instead of s.

If there is no such s then all exiting envelopes are dense and participating, and the result again follows immediately from Fact 3.3 as long as $d_p \geq 4$. Otherwise $d_p \leq 3$ and hence $p \geq J$.

In all cases at most $(d_p - s_p)/4 + s_p$ phase p envelopes survive to phase $p+1$. In addition, there are n_{p+1} new envelopes that begin in phase $p+1$. □

LEMMA 3.4. *The number d_p of envelopes that participate in phase p for $p \leq J$ satisfies $d_p < 4\sum_{i=0}^{p} n_i 2^{i-p}$.*

Proof. By Lemma 3.2, $d_p \leq (d_{p-1})/4 + n_p + s_{p-1}$. Thus, by Lemma 3.1, $d_p < (d_{p-1})/4 + n_p + \sum_{i=0}^{p-1} n_i 2^{i-p+1}$. Solving this recurrence with $d_0 = n_0$ yields: $d_p < 4\sum_{i=0}^{p} n_i 2^{i-p}$. □

The bound for d_p given by Lemma 3.4 can be less than 1, and since d_p is an integer, it must therefore be zero in this case. This reflects the situation when envelopes created by processors adjacent to light edges have all been eliminated and envelopes due to heavy edges are not yet participating. In the following theorem we exploit the fact that there can be phases with no message traffic by not charging for those phases of computation when $4\sum_{i=0}^{p} n_i 2^{i-p} < 1$.

THEOREM 3.5. *Let R be a ring with n_i edges having weight in $(2^{i-1}, 2^i]$. Then the weighted message cost of* WEIGHTED ELECT *on R is $O(\sum_{n_i \geq 1} n_i 2^i \lg(n_i + 1))$.*

Proof. Denote the worst-case weighted message cost of WEIGHTED ELECT on ring R up to phase J by $\mathrm{cost}(R)$. It suffices to bound $\mathrm{cost}(R)$ since the remaining phases have complexity $O(\sum_{n_i \geq 1} n_i 2^i)$. There are at most d_p envelopes participating in round (p,t), each of which travels at most a weighted distance of 2^{p+t+1}. Since there are four rounds per phase, phase p costs less than $d_p 2^{p+5}$. Let $\hat{n}_i$ be the least integer that is a power of 2 and satisfies $\hat{n}_i \geq n_i$. We have:

$$\begin{aligned}
\mathrm{cost}(R) &< \sum_{p=0, d_p \geq 1}^{J} d_p \cdot 2^{p+5} \\
&< \sum_{p=0, d_p \geq 1}^{J} 4\sum_{i=0}^{p} n_i 2^{i-p} \cdot 2^{p+5} \qquad \text{by Lemma 3.4} \\
&\in O\left(\sum_{p=0, d_p \geq 1}^{J} \sum_{i=0}^{p} \hat{n}_i 2^i \right).
\end{aligned}$$

We will now show that $S = \sum_{p=0, d_p \geq 1}^{J} \sum_{i=0}^{p} \hat{n}_i 2^i \in O(\sum_{n_i \geq 1} \hat{n}_i 2^i \lg(\hat{n}_i + 1))$, which implies the theorem.

Let $I = \max\{i : n_i \geq 1\}$. By Lemma 3.4, when p satisfies $I \leq p \leq J$, the number of existing envelopes in phase p is at most $4\sum_{i=0}^{p} n_i 2^{i-p} = 4\sum_{i=0}^{I} n_i 2^{i-p} \leq 4\sum_{i=0}^{I} \hat{n}_i 2^{i-p}$. Thus J (the number of phases until there are three or fewer envelopes) is at most the minimum p satisfying $(4\sum_{i=0}^{I} \hat{n}_i 2^i)/2^p < 4$.

Let A be a $(J+1)$ by $(I+1)$ matrix defined by $A(p,i) = \hat{n}_i 2^{i-p}$ for $p \geq i$ and $A(p,i) = 0$ for $p < i$. Let $D_p = 4\sum_{i=0}^{p} A(p,i)$. Then each element of the summation S can be interpreted as the corresponding element of A multiplied by an appropriate scaling factor. Specifically, $S \leq \sum_{0 \leq p \leq J, D_p \geq 1, 0 \leq i \leq I} A(p,i) \cdot 2^p$. Note that entries for any p such that $D_p < 1$ do not contribute to the sum S. Among the remaining entries of A we consider three types. Entries satisfying $A(p,i) \geq 1$ are called *whole entries* (denoted $\mathcal{W}$). Other entries are called *fractional* entries. Among fractional entries we distinguish *heavy entries* (denoted $\mathcal{H}$) and *light entries* (denoted $\mathcal{L}$), to be defined later.

CLAIM 3.6. $\sum_{A(p,i)\in\mathcal{W}} A(p,i)2^p \in O(\sum_{i=0}^{I} \hat{n}_i 2^i \lg(\hat{n}_i+1))$.

Proof. For any fixed i and $p \geq i$, $A(p,i)\cdot 2^p = \hat{n}_i 2^i$. By definition, the largest entry in column i is $A(i,i) = \hat{n}_i$, and the smallest whole entry is $A(i+\lg\hat{n}_i, i) = 1$. Therefore, there are $\lg\hat{n}_i + 1$ whole entries in column i, which, when multiplied by the corresponding scaling vector, make up exactly those terms included in the summation $\sum_{i=0}^{I} \hat{n}_i 2^i \lg(\hat{n}_i+1)$. □

Therefore, it remains to show that the contribution of the fractional entries in the summation S is of the same order.

For each i satisfying $\hat{n}_i > 0$, let level(i) be the number k satisfying $A(k,i) = 1$. That is, level(i) $= i + \lg\hat{n}_i$. Let $T_k = \{i : \text{level}(i) = k\}$ and $t_k = |T_k|$. A fractional entry $A(p,i)$ is called *heavy* if $p \leq \text{level}(i) + t_{\text{level}(i)}$. A fractional entry which is not heavy is called *light*.

The next claim states that the contribution of all heavy fractional elements to the sum S is of the same order as the contribution of whole elements.

CLAIM 3.7. *For any k such that $T_k \neq \emptyset$,*

$$\sum_{i\in T_k} \sum_{p=k+1}^{k+t_k} A(p,i)2^p \leq 3 \sum_{i\in T_k} \hat{n}_i 2^i \lg\hat{n}_i.$$

Proof. By the definition of T_k, for all $i \in T_k$, $\hat{n}_i/2^{k-i} = 1$. Thus $\hat{n}_i \cdot 2^i = 2^k$. So, $i,j \in T_k$ and $i \neq j$ implies $\hat{n}_i \neq \hat{n}_j$. Therefore, for all $i \in T_k$ the corresponding $\hat{n}_i$ are distinct and are powers of two. Hence

$$\sum_{i\in T_k} \hat{n}_i \cdot 2^i \lg\hat{n}_i = 2^k \cdot \lg\left(\prod_{i\in T_k} \hat{n}_i\right) \geq 2^k \cdot \lg(2^0 \cdot \cdots \cdot 2^{t_k-1}) > 2^k \cdot (t_k)^2/3.$$

On the other hand, for any $i \in T_k$ and for any row p, $A(p,i)\cdot 2^p = n_i 2^i = 2^k$. Hence

$$\sum_{i\in T_k} \sum_{p=k+1}^{k+t_k} A(p,i)\cdot 2^p = (t_k)^2 2^k < 3 \sum_{i\in T_k} \hat{n}_i \cdot 2^i \lg\hat{n}_i. \quad \square$$

To complete the proof we estimate the contribution of light fractional elements. First we show the following claim.

CLAIM 3.8. *For any p,*

$$\sum_{i, A(p,i)\in\mathcal{L}} A(p,i)2^p \in O(2^p).$$

Proof.

$$\sum_{i, A(p,i)\in\mathcal{L}} A(p,i)2^p = \sum_{k, k+t_k<p} \sum_{i\in T_k} A(p,i)2^p = \sum_{k, k+t_k<p} t_k 2^k.$$

Since $k < p$ we have $t_k < p-k$ and thus the last sum is bounded by $\sum_{k=1}^{p-1}(p-k)2^k \in O(2^p)$. □

By Claim 3.8, all light fractional entries that are in the same row as some whole entry contribute to the sum S approximately the same amount as that whole entry. Thus we need to take care of light fractional entries that do not belong to the same

row as a whole entry. Note that each row j that contains a whole entry can be directly followed by at most $\lg D_j$ rows that do not contain whole entries. (Any further row $p > j + \lg D_j$ that does not contain a whole entry has $D_p < 1$ and thus is not counted in the summation.) By Claim 3.8, the contribution of light fractional entries in all these rows is bounded by $\sum_{p=j+1}^{j+\lg D_j} 2^p \leq 2^{j+1+\lg D_j} \in O(D_j 2^j)$. Thus the contribution of light fractional entries that belong to a row that does not contain a whole entry is dominated by the contribution of the closest row that contains a whole entry. This concludes the proof of Theorem 3.5. □

4. Lower bound for election on weighted rings. Let W be a multiset of weights with n_i weights in the interval $(2^{i-1}, 2^i]$ (called *weight class* i). In this section we prove that if an algorithm successfully elects a leader for all unidirectional asynchronous weighted rings, then for every set of n distinct identifiers and every multiset of n weights, there exists an arrangement of these identifiers and weights on a ring of n processors such that the algorithm has weighted message cost $\Omega(\sum_{n_i \geq 1} n_i 2^i \max(\lg n_i, 1))$ on this ring. Thus we establish that the message complexity of algorithm WEIGHTED ELECT is asymptotically optimal for asynchronous unidirectional weighted rings with distinct identifiers.

We assume an asynchronous but reliable model. That is, every message is eventually received unaltered, and messages sent over one link arrive in the same order as they were sent. The proof assumes that no knowledge of ring size is known at the start of the algorithm. Also, our proof applies to message-driven algorithms only. However, a well-established argument extends message-driven lower bounds to lower bounds for algorithms that are not message-driven. See, for example, Pachl, Korach, and Rotem [9]. Also, we borrow and adapt the notation, techniques, and some terminology from that paper.

An asynchronous unidirectional weighted ring R with n processors is denoted by a sequence $R = ((id_0, w_0), \ldots, (id_{n-1}, w_{n-1}))$, called a *labeling sequence*, where id_i is the identifier of the ith processor and w_i is the weight of the link from the ith to the $(i+1)$st processor. For $i \neq j, id_i \neq id_j$, whereas w_i may or may not equal w_j. An algorithm A is a *leader election algorithm* if for every positive integer n and for every weighted ring R with n processors, when algorithm A is run on R:

(i) all messages travel clockwise around the ring;
(ii) computation halts after a finite number of messages;
(iii) upon termination, exactly one processor is in the state "leader."

For algorithm A, the *cost of* A *on ring* R, denoted $\text{cost}_A(R)$, is the total weighted cost of all messages sent by A when executed on ring R. Let W be a multiset of n weights and let I be a set of n distinct identifiers. (Elements of both W and I are assumed to be positive integers.) Let $\mathcal{R}(I, W)$ be the set of all rings $R = ((id_0, w_0), \ldots, (id_{n-1}, w_{n-1}))$ such that $\{id_0, \ldots, id_{n-1}\} = I$ and $\{w_0, \ldots, w_{n-1}\} = W$. Denote by $\text{cost}_A(I, W)$ the maximum over all rings $R \in \mathcal{R}(I, W)$ of $\text{cost}_A(R)$. The *cost of leader election for* $\mathcal{R}(I, W)$ is the minimum over all leader election algorithms A of $\text{cost}_A(I, W)$. Given these definitions, our goal is to show that for any set I of n distinct identifiers, and any multiset W of n weights where n_i weights are in weight class i, the cost of leader election for $\mathcal{R}(I, W)$ is $\Omega(\sum_{n_i \geq 1} n_i 2^i \max(\lg n_i, 1))$.

Call a ring R with edge weights taken from W *well constructed* over W if, for each i, all n_i weights in $(2^{i-1}, 2^i]$ are on consecutive links. Such a sequence of links with weights in the same weight class forms a *segment*. Let $\hat{\mathcal{R}}(I, W)$ denote that subset of $\mathcal{R}(I, W)$ that is well constructed over W.

For each ring R in $\hat{\mathcal{R}}(I, W)$, imagine barriers inserted between the segments of

R and run algorithm A on R with these barriers. That is, schedule the messages of A so that all message traffic from one segment to another segment is delayed at the receiver arbitrarily while message delay within each segment is just one time unit, and run A under this scheduler computing the weighted message cost only until all remaining messages are queued at the barriers. Clearly, this can only decrease the total cost of the message traffic; we show that the total cost of the messages sent in only this part of the execution suffices to give the lower bound. Hence, to establish the lower bound we need only show that the average cost of a segment constructed from the n_i weights in weight class i is bounded below by $\Omega(n_i 2^i \max(\lg n_i, 1))$.

First observe that every edge of the network must carry at least one message. This is because, if there is a process that does not send an initial message, then whatever conditions caused the process to not initiate could be reproduced around the ring and result in deadlock. So, for each segment with $n_i = 1$, the required bound for that segment is trivial. To achieve the bound for $n_i \geq 2$ we examine the expected message traffic that ensues within a segment. Once this is determined it is a simple matter to sum these costs, appropriately weighted, over all segments.

Define the *trace* of a message envelope created by the kth processor when it arrives at the pth processor to be the sequence $(id_k, w_k), (id_{k+1}, w_{k+1}), \ldots, (id_p, w_p)$. Because the ring is unidirectional, the trace of a message captures the maximum possible information that a message may possess. Notice that a message envelope with trace $(id_k, w_k), (id_{k+1}, w_{k+1}), \ldots, (id_p, w_p)$ has contributed a weighted cost of $\sum_{i=k}^{p-1} w_i$ to the weighted message cost of the algorithm.

If s is a sequence, then let $|s|$ denotes its length and let $C(s)$ denote the set of cyclic permutations of s. A sequence t is a *subsequence* of s if $s = utv$ for some sequences u and v.

Consider an arbitrary but fixed weight class i. Denote by D the set of all finite nonempty labelling sequences where all weights are in weight class i. For $s \in D$ and $E \subseteq D$ and positive integer k, define

$$B(s, E) = |\{t : t \in E \text{ and } t \text{ is a subsequence of } s\}| \qquad \text{and}$$

$$B_k(s, E) = |\{t : t \in E \text{ and } |t| = k \text{ and } t \text{ is a subsequence of } s\}|.$$

A set $E \subseteq D$ is *exhaustive* if the following two properties hold.

1. *Prefix property*: if $tu \in E$ and $|t| \geq 1$ then $t \in E$.
2. *Cyclic permutation property*: if $s \in D$ then $C(s) \cap E \neq \emptyset$.

For any algorithm A for unidirectional rings, define $m(s, A)$ to be the number of messages sent by A on a segment with labelling sequence s (equivalently, on a ring labelled with s when a barrier is placed between s_n and s_1). Define $E(A) \subseteq D$ to be the set of those $t \in D$ for which a message with trace t is sent when executed on a ring labelled t.

LEMMA 4.1. *For every leader election algorithm A, the set $E(A)$ is an exhaustive set satisfying $m(s, A) \geq B(s, E(A))$ for every $s \in D$.*

Proof. Suppose that $s \in E(A)$ and t is a prefix of s and $1 \leq k = |t| < |s| = n$. Then on a ring $R = p_1, \ldots, p_n$ labelled by s, a message envelope travels from p_1 to p_n. Hence on a ring $p'_1, \ldots, p'_k$ labelled by t, the envelope created by p'_1 travels to p'_k. Thus $t \in E(A)$ and the prefix property holds.

Suppose $s = s_1, \ldots, s_n \in D$, and consider a ring labelled with s. Since A is a leader election algorithm there must be one message envelope that travels the whole

ring because otherwise algorithm A could not successfully elect a leader on rings labelled with an extension of s. This message envelope has a trace of length at least n and hence has a prefix of length exactly n. Thus this prefix is a cyclic permutation of s. So $E(A)$ is exhaustive.

A similar argument confirms that if trace t is sent by A when executed on a ring labelled with t, then trace t is sent by A when executed on any sequence that contains t as a subsequence. Therefore, $m(s, A) \geq B(s, E(A))$. □

Now Lemma 4.1 is used to establish the expected weighted cost of message traffic within a segment. Let W be a multiset of n weights in $(2^{i-1}, 2^i]$. Let I be a set of n distinct integer identifiers, and consider $\mathcal{R}(I, W)$.

LEMMA 4.2. *For any leader election algorithm A, the average of $m(s, A)$ over all labelling sequences $s \in \mathcal{R}(I, W)$ is bounded below by $H_n \cdot n - n$.*

Proof.

$$\begin{aligned} ave\{m(s,A)\}_{s\in\mathcal{R}(I,W)} &= \frac{1}{|\mathcal{R}(I,W)|} \sum_{s\in\mathcal{R}(I,W)} m(s,A) \\ &\geq \frac{1}{|\mathcal{R}(I,W)|} \sum_{s\in\mathcal{R}(I,W)} B(s,E(A)) \\ &\geq \frac{1}{|\mathcal{R}(I,W)|} \sum_{k=1}^{n} \sum_{s\in\mathcal{R}(I,W)} B_k(s,E(A)). \end{aligned}$$

For fixed k and a fixed $s \in \mathcal{R}(I, W)$, there are $n-k+1$ subsequences of s with length k, so there are $|\mathcal{R}(I, W)|(n-k+1)$ length k subsequences over all $s \in \mathcal{R}(I, W)$. Partition these into $\frac{|\mathcal{R}(I,W)|(n-k+1)}{k}$ sets, where each set consists of all cyclic permutations of one sequence. By the cyclic permutation property, each set has at least one element in common with $E(A)$. Hence:

$$\begin{aligned} ave\{m(s,A)\}_{s\in\mathcal{R}(I,W)} &\geq \frac{1}{|\mathcal{R}(I,W)|} \sum_{k=1}^{n} \frac{|\mathcal{R}(I,W)|(n-k+1)}{k} \\ &= (n+1) \sum_{k=1}^{n} \frac{1}{k} - n \\ &= (n+1)H_n - n \in \Omega(n \lg n). \end{aligned}$$ □

THEOREM 4.3. *Let I be a set of n distinct identifiers, and let W be a multiset of weights with n_i weights in the interval $(2^{i-1}, 2^i]$. Then the cost of leader election for $\mathcal{R}(I, W)$ is*

$$\Omega\left(\sum_{n_i \geq 1} n_i 2^i \max(\lg n_i, 1)\right).$$

Proof. For any i such that $n_i = 1$, there is at least one initial message sent by the segment on a link of weight at least 2^{i-1}. If $n_i \geq 2$, then by Lemma 4.2, on average, $\Omega(n_i \lg n_i)$ messages are sent over the segment formed from the n_i elements of W that are in the weight class i and each message incurs a weighted cost of at least 2^{i-1}. Hence, the lower bound for each segment is $\Omega(n_i 2^i \max(\lg n_i, 1))$. Thus, the lower bound for leader election on weighted rings follows by summing over all segments. □

5. Concluding remarks. Our algorithm and lower bound together establish the asymptotic communication complexity of leader election on weighted unidirectional rings. The result is quite strong from several perspectives. First, our algorithm is universally optimal in the sense of Garay, Kutten, and Peleg [4]. That is, our lower bound is asymptotically tight for every set of identifiers and for every multiset of weights. Note that for some specially constructed rings (for example, rings with identifiers arranged in increasing order), there are leader election algorithms that are very efficient. Therefore, the universality of the lower bound cannot be further generalized from all possible sets of identifiers and weights to all possible rings. We conclude that the parameters that determine the complexity of election on a unidirectional weighted ring are precisely the number of edges in each weight class.

Also, we have shown that it is not possible to specially tune a leader election algorithm to be inexpensive for some chosen collection of weights and identifiers without it being incorrect for others. Specifically, our lower bound establishes that as long as an algorithm correctly elects a leader for all unidirectional rings, then, for every multiset W of weights and every set of identifiers I, it will be at least as expensive (asymptotically) as WEIGHTED ELECT on $\mathcal{R}(I, W)$.

There is one essential constraint on our lower bound. It applies only to those algorithms that elect a leader for *all* unidirectional weighted rings. So, for example, the proof of our lower bound does not apply if an algorithm need only work for a fixed ring size or total weight. However, having knowledge of ring size or weight while having no more specific knowledge of the arrangement of the weights on the ring seems to be an unreasonable assumption. And, as we have observed, if there is some knowledge of arrangement of weights, then this knowledge could possibly be exploited to achieve a very efficient algorithm for this arrangement.

The importance of cost-sensitive analysis of distributed algorithms was pointed out by Awerbuch, Baratz, and Peleg [1]. The cost-sensitive complexity of election on unidirectional rings is only a first step; it remains to study the cost-sensitive complexity of other problems and other networks.

Acknowledgments. We are especially grateful to two anonymous referees for their careful reading of the manuscript, their helpful comments, and their very timely response.

REFERENCES

[1] B. Awerbuch, A. Baratz, and D. Peleg, *Cost-sensitive analysis of communication protocols*, in Proc. 9th Annual ACM Symp. on Principles of Distributed Computing, 1990, pp. 177–187.

[2] E. Chang and R. Roberts, *An improved algorithm for decentralized extrema-finding in circular configurations of processes*, Comm. ACM, 22 (1979), pp. 281–283.

[3] D. Dolev, M. Klawe, and M. Rodeh, *An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle*, J. Algorithms, 3 (1982), pp. 245–260.

[4] J. A. Garay, S. Kutten, and D. Peleg, *A sub-linear time distributed algorithm for minimum weight spanning trees*, in Proc. 34th Annual Symp. on Foundations of Comput. Sci., 1993, pp. 659–668.

[5] L. Higham and T. Przytycka, *A Simple, Efficient Algorithm for Maximum Finding on Rings*, Tech. Report 92/494/32, University of Calgary, Alberta, Canada, 1992.

[6] L. Higham and T. Przytycka, *A simple, efficient algorithm for maximum finding on rings*, Inform. Proc. Lett., 58 (1996), pp. 319–324.

[7] D. Hirschberg and J. B. Sinclair, *Decentralized extrema-finding in circular configurations of processes*, Comm. ACM, 23 (1980), pp. 627–628.

[8] G. LeLann, *Distributed systems — towards a formal approach*, in Info. Proc. 77, New York, 1977, Elsevier Science, New York, pp. 155–160.

[9] J. Pachl, E. Korach, and D. Rotem, *Lower bounds for distributed maximum finding*, J. Assoc. Comput. Mach., 31 (1984), pp. 905–918.
[10] G. Peterson, *An $O(n \log n)$ algorithm for the circular extrema problem*, ACM Trans. Prog. Lang. Systems, 4 (1982), pp. 758–762.

SIAM J. COMPUT.
Vol. 28, No. 2, pp. 733–769

THE QUEUE-READ QUEUE-WRITE PRAM MODEL: ACCOUNTING FOR CONTENTION IN PARALLEL ALGORITHMS*

PHILLIP B. GIBBONS†, YOSSI MATIAS†‡, AND VIJAYA RAMACHANDRAN§

Abstract. This paper introduces the *queue-read queue-write* (QRQW) parallel random access machine (PRAM) model, which permits concurrent reading and writing to shared-memory locations, but at a cost proportional to the number of readers/writers to any one memory location in a given step. Prior to this work there were no formal complexity models that accounted for the contention to memory locations, despite its large impact on the performance of parallel programs. The QRQW PRAM model reflects the contention properties of most commercially available parallel machines more accurately than either the well-studied CRCW PRAM or EREW PRAM models: the CRCW model does not adequately penalize algorithms with high contention to shared-memory locations, while the EREW model is too strict in its insistence on zero contention at each step.

The QRQW PRAM is strictly more powerful than the EREW PRAM. This paper shows a separation of $\sqrt{\log n}$ between the two models, and presents faster and more efficient QRQW algorithms for several basic problems, such as linear compaction, leader election, and processor allocation. Furthermore, we present a work-preserving emulation of the QRQW PRAM with only logarithmic slowdown on Valiant's BSP model, and hence on hypercube-type noncombining networks, *even when latency, synchronization, and memory granularity overheads are taken into account.* This matches the best-known emulation result for the EREW PRAM, and considerably improves upon the best-known efficient emulation for the CRCW PRAM on such networks. Finally, the paper presents several lower bound results for this model, including lower bounds on the time required for broadcasting and for leader election.

Key words. models of parallel computation, parallel algorithms, PRAM, memory contention, work-time framework

AMS subject classifications. 68Q05, 68Q22, 68Q25

PII. S009753979427491

1. Introduction.

The parallel random access machine (PRAM) model of computation is the most-widely used model for the design and analysis of parallel algorithms (see, e.g., [40, 39, 58]). The PRAM model consists of a number of processors operating in lock-step and communicating by reading and writing locations in a shared memory. Existing PRAM models can be distinguished by their rules regarding contention for shared memory locations. These rules are generally classified into two groups:

- *Exclusive read/write*: Each location can be read or written by at most one processor in each unit-time PRAM step.
- *Concurrent read/write*: Each location can be read or written by any number of processors in each unit-time PRAM step. For concurrent writing, the value written depends on the *write-conflict rule* of the model, e.g., in the *arbitrary concurrent-write* PRAM, an arbitrary processor succeeds in writing its value.

*Received by the editors September 21, 1994; accepted for publication (in revised form) January 8, 1997; published electronically August 4, 1998.
http://www.siam.org/journals/sicomp/28-2/27491.html
†Bell Laboratories, Lucent Technologies, 600 Mountain Avenue, Murray Hill, NJ 07974 (gibbons@research.bell-labs.com).
‡Current address: Department of Computer Science, Tel-Aviv University, Tel-Aviv, Israel (matias@math.tau.ac.il).
§Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712 (vlr@cs.utexas.edu). This author was supported in part by NSF grant CCR-90-23059 and Texas Advanced Research Projects grants 003658480 and 003658386.

These two rules can be applied independently to reads and writes; the resulting models are denoted in the literature as the EREW, CREW, ERCW, and CRCW PRAM models.

In this paper, we argue that neither the *exclusive* nor the *concurrent* rules accurately reflect the contention capabilities of most commercial and research machines, and propose a new PRAM contention rule, the *queue* rule, that permits concurrent reading and writing, but at an appropriate cost:

- *Queue read/write*: Each location can be read or written by any number of processors in each step. Concurrent reads or writes to a location are serviced one at a time.

Thus the worst case time to read or write a location is linear in the number of concurrent readers or writers to the same location.

The queue rule more accurately reflects the contention properties of machines with simple, noncombining interconnection networks[1] than either the exclusive or concurrent rules. The exclusive rule is too strict, and the concurrent rule ignores the large performance penalty of high-contention steps. Indeed, for most existing machines, including the Cray T3D, IBM SP2, Intel Paragon, MasPar MP-2 (global router), MIT J-Machine, nCUBE 2S, Stanford DASH, Tera Computer, and Thinking Machines CM-5 (data network), the contention properties of the machine are well approximated by the queue-read, queue-write (QRQW) rule. For the Kendall Square KSR1, the contention properties can be approximated by the concurrent-read, queue-write (CRQW) rule. Further details are in section 3.

This paper defines the QRQW PRAM model, a variation on the standard PRAM that employs the queue rule for both reading and writing. In addition, the processors are permitted to each have multiple reads or writes in progress at a time. We show that the power of the QRQW PRAM model falls strictly between the CRCW and EREW models. We show separation results between the models by considering the 2-compaction problem, the broadcasting problem, and the problem of computing the OR function. To illustrate some of the techniques used to design low-contention algorithms that improve upon the best known zero-contention algorithms, we consider algorithms for two fundamental problems, *leader election* and *linear compaction*, under various scenarios. Finally, this paper extends the work-time framework for parallel algorithms (see, e.g., [39]) into a QRQW work-time framework that considers the contention at each step, and relates the QRQW PRAM model to the QRQW work-time framework.

The QRQW PRAM, like the other PRAM models mentioned above, abstracts away many features of real machines, including the latency or delay in accessing the shared memory, the cost of synchronizing the processors, and the fact that memory is partitioned into modules that service requests serially. A model that incorporates these features is the bulk-synchronous parallel (BSP) model of Valiant [61]. In its general form this model is parameterized by its number of processing/memory *components* p, *throughput* g, and *periodicity* L. A particular case studied by Valiant sets g to be a constant and L to be $\Theta(\log p)$; we denote this the *standard* BSP model. We show in this paper that the QRQW PRAM can be effectively emulated on the standard BSP model: A p-processor QRQW PRAM algorithm running in time t can be emulated on a $p/\log p$-

[1] In a *combining network*, when two messages destined for the same memory location meet at an intermediate node in the network, the messages are "combined" so that only one message continues toward the destination. For example, if two writes meet, then only a single write is sent on. In a *noncombining network*, messages are not combined, so that *all* messages destined for the same memory location are delivered to the home node for that location.

processor standard BSP in $O(t \log p)$ time with high probability (w.h.p.). It follows by Valiant's simulation of the standard BSP on hypercubes that the QRQW PRAM can be emulated in a work-preserving manner on parallel machines with hypercube-type, noncombining networks with only logarithmic slowdown, *even when latency, memory granularity, and synchronization overheads are taken into account.* This matches the best-known emulation for the EREW PRAM on these networks given in [61]. In contrast, work-preserving emulations for the CRCW PRAM on such networks are only known with *polynomial* slowdown (i.e., $O(p^\epsilon)$ slowdown, for a constant $\epsilon > 0$).

Note that the standard $\Theta(\log p)$ time emulation of CRCW on EREW (see, e.g., [40]) is not work preserving, in that the EREW performs $\Theta(\log p)$ times more work than the CRCW it emulates. Since we consider work-preserving speedups to be the primary goal in parallel algorithms, with fast running times the secondary goal, this emulation is unacceptable: The $\Theta(\log p)$ overhead in work ensures that the algorithms will not exhibit linear or near-linear speedups. Similarly, the best-known emulations for the CREW PRAM (or ERCW PRAM) on the EREW PRAM (or standard BSP or hypercube) require logarithmic work overhead for logarithmic slowdown or, alternatively, polynomial slowdown for constant work overhead.

Since the QRQW PRAM is strictly more powerful than the EREW PRAM, effectively emulated on hypercube-type noncombining networks (unlike the CRCW, CREW, or ERCW PRAM models), and a better match for real machines, we advocate the QRQW PRAM with its *queue*-contention rule as a more appropriate model for high-level algorithm design than a PRAM with either the *exclusive*- or *concurrent*-contention rules. The queue-contention rule can also be incorporated into lower-level shared-memory models, trading model simplicity for additional accuracy in modeling the cost of communication (e.g., explicitly modeling the communication bandwidth). In this initial paper on the queue-contention rule, we restrict our focus to high-level algorithm design on PRAM models.

In addition to the QRQW PRAM model, we define in this paper the SIMD-QRQW PRAM model, a strictly weaker model suitable for SIMD machines, in which all processors execute in lock-step and each processor can have at most one read/write in progress at a time. In a subsequent paper [30] we define the QRQW ASYNCHRONOUS PRAM model, for general asynchronous algorithms running on MIMD machines (see also [26]).

We present several algorithms and a lower bound for leader election and for computing the OR function. The lower bound is $\Omega(\log n / \log \log n)$ time for the deterministic computation of the OR function on a CRQW PRAM with arbitrarily many processors. The algorithms for both problems take linear work and $O(\log n / \log \log n)$ time with high probability. In contrast, the OR function requires $\Omega(\log n)$ expected time on a randomized CREW PRAM with arbitrarily many processors ([17], following [12]). Also presented is a linear work, $O(\sqrt{\log n})$ time w.h.p. algorithm for the linear-compaction problem. This problem has applications to automatic processor allocation for algorithms that are given in the QRQW work-time presentation. In contrast, the best linear-compaction algorithm known on the EREW PRAM is the logarithmic time prefix sums algorithm [42]. On the other hand, for the problem of broadcasting the value of a bit to n processors, we show that we can do no better on the QRQW PRAM than the simple $\Theta(\log n)$ time EREW PRAM algorithm. Specifically, we show a tight $\Omega(\log n)$ expected-time lower bound for the QRQW PRAM.

Important technical issues arise in designing algorithms for the queue models that are present in neither the concurrent nor the exclusive PRAM models. For example,

much of the effort in designing algorithms for the QRQW is in estimating the maximum contention in a step; our algorithms for leader election illustrate this point. In the QRQW, one high-contention step can dominate the running time of the algorithm: we cannot afford to underestimate the contention significantly.

In a companion paper [29], we present a number of other algorithmic results for the QRQW PRAM. Our algorithmic results include linear work, logarithmic or sublogarithmic time randomized QRQW algorithms for the fundamental problems of multiple compaction, load balancing, generating a random permutation, parallel hashing, and sorting from $U(0,1)$. These algorithms improve upon the best-known EREW algorithms for these problems, while avoiding the high-contention steps typical of CRCW algorithms. Additionally, we present new algorithms for integer sorting and general sorting.

Most of the results in [29], and some of the results in this paper, are obtained w.h.p. A probabilistic event occurs w.h.p. if, for any prespecified constant $\delta > 0$, it occurs with probability $1 - 1/n^{\delta}$, where n is the size of the input. Thus, we say a randomized algorithm runs in $O(f(n))$ time w.h.p. if for every prespecified constant $\delta > 0$, there is a constant c such that for all $n \geq 1$, the algorithm runs in $c \cdot f(n)$ steps or less with probability at least $1 - 1/n^{\delta}$.

The rest of this paper is organized as follows. Section 2 defines the QRQW PRAM and SIMD-QRQW PRAM models. Section 3 gives further motivation for the queue models, and comparison with related work. Section 4 describes the extension of the work-time framework to the QRQW models. Section 5 presents our results for realizing the QRQW PRAM on feasible networks. Section 6 gives upper and lower bounds for computing the OR and leader election under various scenarios. Section 7 presents our linear-work, sublogarithmic-time algorithm for linear compaction on a SIMD-QRQW PRAM. Section 8 presents tight $\Omega(\log n)$ expected-time lower bounds on the QRQW PRAM for broadcasting and related problems. Concluding remarks appear in section 9.

The results in this paper appeared in preliminary form in [26, 27, 28].

2. The queue models. This section defines our two QRQW models:

- the SIMD-QRQW PRAM, for algorithms running on SIMD machines, and
- the QRQW PRAM, for bulk-synchronous algorithms[2] running on MIMD machines.

In both of the QRQW models, the time cost for reading or writing a shared location, x, is proportional to the number of processors concurrently reading or writing x. This cost measure models machines in which accesses to a location queue up and are serviced one at a time, i.e., most current commercial and research machines. The SIMD-QRQW models machines in which processors synchronize at every step, waiting for all the queues to clear. The QRQW models machines in which processors synchronize less frequently, waiting for all the queues to clear only at synchronization points. In a subsequent paper [30] we define the QRQW ASYNCHRONOUS PRAM model, for general asynchronous algorithms running on MIMD machines (see also [26]). This model has an *asynchronous* queue-contention rule in which processors read and write locations at their own pace, without waiting for the queues encountered by other processors to clear. This model allows the asynchronous nature of MIMD machines to be exploited, at the cost of more complexity in the model.

[2] In a *bulk-synchronous* algorithm [61, 24, 25], synchronization among the processors is limited to global synchronization barriers involving all the processors; between such barriers, processors execute asynchronously using shared-memory values written prior to the preceding barrier.

In order to preserve the simplicity of the SIMD-QRQW PRAM and QRQW PRAM models, neither model incorporates the cost of synchronizing after a step. We note, however, that our result on a work-preserving emulation of both models on a BSP shows that the cost of synchronization can be hidden (up to a constant factor) by using a target machine with a somewhat smaller number of processors.

The complexity metric for the QRQW models will use the notion of maximum contention, defined as follows.

DEFINITION 2.1. *Consider a single step of a* PRAM*, consisting of a read substep, a compute substep, and a write substep. The* maximum contention *of the step is the maximum, over all locations x, of the number of processors reading x or the number of processors writing x. For simplicity in handling a corner case, a step with no reads or writes is defined to have maximum contention* 1.

2.1. The SIMD-QRQW PRAM model.

DEFINITION 2.2. *The* SIMD-QRQW PRAM *model is a (synchronous)* PRAM *in which concurrent reads and writes to the same location are permitted, and the* time cost *for a step with maximum contention κ is κ. If there are multiple writers to a location x in a step, an arbitrary write to x succeeds in writing the value present in x at the end of the step. The* time *of a* SIMD-QRQW PRAM *algorithm is the sum of the time costs for its steps. The* work *is its processor-time product.*

This cost measure models, for example, a SIMD machine such as the MasPar MP-1 [51] or MP-2, in which each processor can have at most one read/write in progress at a time, reads/writes to a location queue up and are serviced one at a time, and all processors await the completion of the slowest read/write in the step before continuing to the next step. Existing SIMD machines provide for the required synchronization of all processors at each step, regardless of the varying contention encountered by the individual processors. Unlike previous PRAM models, the work is not the number of operations, because with the SIMD-QRQW time metric, operations encountering nonconstant contention are charged nonconstant time.

If a PRAM model is to be used to design bulk-synchronous algorithms on MIMD machines, then the SIMD-QRQW PRAM is unnecessarily restrictive. A better model for this scenario is the QRQW PRAM, defined next.

2.2. The QRQW PRAM model.

DEFINITION 2.3. *The* QRQW PRAM *model consists of a number of processors, each with its own private memory, communicating by reading and writing locations in a shared memory. Processors execute a sequence of synchronous steps, each consisting of the following three substeps:*

1. *Read substep: Each processor i reads r_i shared-memory locations, where the locations are known at the beginning of the substep.*
2. *Compute substep: Each processor i performs c_i* RAM *operations involving only its private state and private memory.*[3]
3. *Write substep: Each processor i writes to w_i shared-memory locations (where the locations and values written are known at the beginning of the substep).*

Concurrent reads and writes to the same location are permitted in a step. In the case of multiple writers to a location x, an arbitrary write to x succeeds in writing the value present in x at the end of the step.

[3] As in the existing PRAM models, each processor is assumed to be a sequential random access machine. See, e.g., [58]. For the QRQW PRAM, a processor may perform multiple RAM operations in a compute substep, e.g., summing c_i numbers stored in its private memory, and is charged accordingly.

DEFINITION 2.4. *Consider a* QRQW PRAM *step with maximum contention* κ*, and let* $m = \max_i\{r_i, c_i, w_i\}$ *for the step, i.e., the maximum over all processors* i *of its number of reads, computes, and writes. Then the* time cost *for the step is* $\max\{m, \kappa\}$*. The* time *of a* QRQW PRAM *algorithm is the sum of the time costs for its steps. The* work *of a* QRQW PRAM *algorithm is its processor-time product.*

This cost measure models, for example, a MIMD machine such as the Tera Computer [2], in which each processor can have multiple reads/writes in progress at a time, and reads/writes to a location queue up and are serviced one at a time. Neither the EREW PRAM nor the CRCW PRAM model allows a processor to have multiple reads/writes in progress at a time, as this generalization is unnecessary when reads/writes complete in unit time. This feature, which distinguishes the QRQW PRAM from the SIMD-QRQW PRAM as well as the EREW PRAM and CRCW PRAM, enables the processors to do useful work while awaiting the completion of reads/writes that encounter contention. Nevertheless, as we show below, the CRCW PRAM can simulate the QRQW PRAM to within constant factors.

The restriction that the processors in a read substep know, at the beginning of the substep, the locations to be read reflects the intended emulation of the QRQW PRAM model on a MIMD machine in which the reads are issued in a pipelined manner, to amortize against the delay (latency) on such machines in reading the shared memory. Likewise writes in a write substep are to be pipelined in the intended emulation. On the other hand, each of the local operations performed in a compute substep can depend on compute operations in the same substep; since these operations are assumed to take constant time in the intended emulation, there is no need for pipelining (to within constant factors). The emulation inserts a barrier synchronization among all the processors between every read and write substep, so that the processors notify each other when it is safe to proceed with the next substep. This synchronization is accounted for in the emulation. A formal description of the intended emulation and its performance appears in section 5.

On existing parallel machines, there are a number of factors that determine the time to process shared-memory read and write requests, including contention in the interconnection network and at the memory modules. Often, reads and writes to *distinct* shared-memory locations may delay one another. Moreover, issued memory requests cannot be withdrawn. To reflect these realities of existing machines, the QRQW PRAM (as well as the SIMD-QRQW PRAM) does not permit processors to make inferences on the contention encountered based on the delays incurred. In addition, issued memory requests may not be withdrawn, and an algorithm has not completed until all issued memory requests have been processed. In this way, the QRQW models, although explicitly accounting only for the delays resulting from multiple requests to the *same* locations, can be efficiently emulated on models that account for these additional concerns, as shown in section 5.

As with the SIMD-QRQW PRAM, the work is not the number of operations, since operations encountering nonconstant contention may be charged nonconstant time. (In fact, the only situation where the work is a good reflection of the number of operations is when pipelining is extensively employed, i.e., when the average over i of $(r_i + c_i + w_i)$ is $\Omega(\kappa)$.)

Also, as with the SIMD-QRQW PRAM, there is no explicit metric for the number of steps in an algorithm. As we show in section 5, there is no need for such a metric in the context of the intended emulation. On the other hand, the synchronization at the end of each bulk-synchronous step is a source of overhead on existing machines,

and hence we may wish to include this additional metric when analyzing algorithms on the QRQW models.

2.3. Relations between models. The primary advantage of the QRQW PRAM model over the SIMD-QRQW PRAM model is that the QRQW permits processors each to perform a series of reads and writes in a step while incurring only a single penalty for the contention of these reads and writes. In the SIMD-QRQW, a penalty is charged after each read or write in the series; often the resulting aggregate charge for contention is far greater than the single charge under the QRQW model. On the other hand, by adding more processors to the SIMD-QRQW, we can match the time bounds (but not the work bounds) obtained for the QRQW.

OBSERVATION 2.1. *A p-processor* QRQW PRAM *algorithm running in time t can be emulated on a pt-processor* SIMD-QRQW PRAM *in time* $O(t)$.

Proof. For each QRQW processor $i \in [1..p]$, we assign a team, T_i, of t SIMD-QRQW processors, with each team having a leader, l_i. Each leader l_i maintains the entire local state of QRQW processor i during the emulation. For each team T_i, we have an auxiliary array, A_i, of size t for communications between l_i and each member of its team. Consider the jth step of a QRQW PRAM algorithm, with time cost t_j and maximum contention $k_j \leq t_j$. For each QRQW processor i, let r_i, c_i, and w_i be the number of reads, RAM operations, and writes performed by processor i in this step. Processor i is emulated as follows. (1) The leader l_i writes the r_i locations to be read to A_i, one location per cell. (2) Each member of T_i reads its cell in A_i, reads the designated location (if any) in the shared memory, and then writes the value read to its cell in A_i. (3) The leader l_i reads the values in A_i, performs the c_i RAM operations, and then writes the w_i locations and values to be written to A_i, one per cell. (4) Finally, each member of T_i reads its cell in A_i, and then writes the designated value to the designated location (if any) in the shared memory. Step 1 takes $O(r_i)$ time, step 2 takes $O(k_j)$ time, step 3 takes $O(r_i + c_i + w_i)$ time, and step 4 takes $O(k_j)$ time. Thus the overall time to emulate the jth QRQW step is $O(t_j)$, and the observation follows. □

Note that in fact only $p \cdot \tau$ processors are needed in the above emulation, where $\tau \leq t$ is the maximum time for any one step of the QRQW PRAM algorithm.

The SIMD-QRQW PRAM model permits each processor to have at most one shared-memory request outstanding at a time, as in the standard PRAM model. This places an upper bound on the number of requests that must be handled by the interconnection network of the parallel machine. For most MIMD machines, permitting only one request per processor is artificially restrictive, and the QRQW PRAM model has no such restriction. On the other hand, since there is no bound in the QRQW PRAM on the number of outstanding requests, there is a danger that QRQW PRAM algorithms will flood the network with requests beyond its capacity to efficiently process them. One approach toward alleviating this potentially serious problem is to divide steps with many shared-memory requests into a sequence of steps with fewer requests per step. In general we could indicate, for each QRQW PRAM algorithm, the maximum number of requests in any one step of the algorithm. Then when implementing the algorithm on a given parallel machine, this number could be compared with the maximum effective network capacity of the machine to determine if the memory requests can be efficiently processed by the network.

Let M_1 and M_2 be two models. We define $M_1 \preceq M_2$ to denote that any one step of M_1 with time cost $t \geq 1$ can be emulated in $O(t)$ time on M_2 using the same number of processors. For concurrent and queue writes we assume throughout this

TABLE 1
Time separation results for the QRQW. *Results on problems inducing a time separation of the* QRQW *from the* EREW *model and the* CRCW *from the* QRQW *model, including both deterministic time (*det.*) and randomized expected or w.h.p. time (*rand.*).*

Stronger model	Weaker model	Separation	Problem	Section
{det.,rand.} SIMD-QRQW	{det.,rand.} EREW	$\Omega(\sqrt{\log n})$	2-compaction	7
det. CRCW	det. QRQW	$\Omega(\frac{\log n}{\log \log n})$	OR function	6.1
{det.,rand.} CRCW	{det.,rand.} QRQW	$\Omega(\log n)$	broadcasting	8

paper that an arbitrary processor succeeds in the write; however, the relations stated below hold as long as both machines use the same write-conflict rule.

OBSERVATION 2.2. EREW PRAM $\preceq$ SIMD-QRQW PRAM $\preceq$ QRQW PRAM $\preceq$ CRCW PRAM.

Proof. The observation can be proved by straightforward emulation. For the CRCW emulating a QRQW step of time cost t: (1) for $j = 1, \ldots, \max_i\{r_i\}$, perform the jth read operation (if any) at each processor in one step using CR; then (2) for $j = 1, \ldots, \max_i\{c_i\}$, perform the jth compute operation (if any) at each processor; then (3) for $j = 1, \ldots, \max_i\{w_i\}$, perform the jth write operation (if any) at each processor in one step using CW. This takes time $\max_i\{r_i\} + \max_i\{c_i\} + \max_i\{w_i\} \leq 3t$. □

Let M_1 and M_2 be two models such that $M_1 \preceq M_2$. A computational problem P *induces a separation of* $O(f(n))$ *time with* $q(n)$ *processors of* M_2 *from* M_1 if there exists a function $t(n)$ such that, on inputs of length n, P can be solved on M_2 in time $O(t(n))$ with $q(n)$ processors, but P requires $\Omega(t(n) \cdot f(n))$ on M_1 if only $q(n)$ processors are available. We say that there is a *separation of* $f(n)$ *time with* $q(n)$ *processors of* M_2 *from* M_1 if there exists a problem that induces such a separation. Most of the separation results we derive in this paper hold for any $q(n) = \Omega(n)$; in such cases we omit $q(n)$ when stating the result.

Results on problems inducing a separation of the QRQW from the EREW model and the CRCW from the QRQW model appear in Table 1.

2.4. A family of queue models. The definitions of SIMD-QRQW PRAM and QRQW PRAM can be generalized so that the charge for maximum contention κ is $f(\kappa)$, a nondecreasing function of κ. When $f(\kappa) = 1$ for all κ, both models are equivalent to the CRCW PRAM. Likewise, when $f(1) = 1$ and $f(\kappa) = \infty$ for $\kappa \geq 2$, both models are equivalent to the EREW PRAM. Note that the distinction between the SIMD-QRQW PRAM and the QRQW PRAM arises only when $f(\kappa) > 1$ and is finite for some κ.

Another possible cost function is $f(\kappa) = \log \kappa$; such a function may occur in a *hypothetical* variant of combining networks, but it is not known to be relevant to any existing machines (there are no known techniques for achieving this cost function for an arbitrary set of readers/writers). The log cost function may prove to be relevant to future machines that employ an optical crossbar to interconnect the processors [34, 48]. However, in this paper, we will focus our attention on the cost function, $f(\kappa) = \kappa$, that reflects the realities of proven technologies. (For some machines that do not handle contention well, superlinear functions such as $f(\kappa) = \kappa \log \kappa$ may be appropriate; such cost functions are not considered in this paper.) Other possible variants of the model permit write-conflict rules other than *arbitrary*; however, we note that the arbitrary rule reflects the realities of most current commercial and research machines.

As the queue rule can be applied independently to reads or writes, we can also

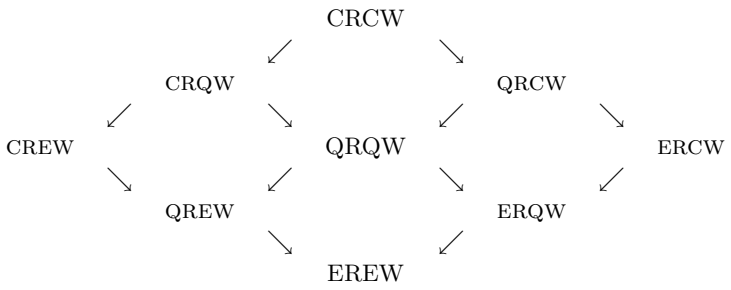


FIG. 1. *The relative power of various* PRAM *concurrency rules. The same relationships hold for the* SIMD *versions of the queue models. For concurrent write, we assume an arbitrary processor succeeds in writing. In this figure an arrow denotes that the* PRAM *model,* M_1*, at the tail of the arrow can simulate the* PRAM *model,* M_2*, at its head with at most a small constant loss in performance (and possibly some improvement), i.e.,* $M_2 \preceq M_1$*. Our results characterize more precisely the relative power of some of the concurrency rules.*

TABLE 2

*Time separation results for the hybrid queue models, including both deterministic time (*det.*) and randomized expected time or w.h.p. time (*rand.*). All results listed here hold for the* SIMD *versions as well.*

Stronger model	Weaker model	Separation	Problem	Section
{det.,rand.} ERQW	{det.,rand.} EREW	$\Omega(\sqrt{\log n})$	2-compaction	7
det. {QR,CR}QW	det. {QR,CR}EW	$\Omega(\log\log n)$ (with n procs)	2-compaction	7
rand. CRQW	rand. CREW	$\Omega(\log\log n)$	OR function	6.3
det. {ER,QR,CR}CW	det. {ER,QR,CR}QW	$\Omega(\frac{\log n}{\log\log n})$	OR function	6.1
det. CR{EW,QW} rand. CR{EW,QW}	det. QR{EW,QW} rand. QR{EW,QW}	$\Omega(\log n)$ $\Omega(\log n)$	broadcasting broadcasting	8

consider models such as the SIMD-CRQW or CRQW PRAM. For each such hybrid model, the PRAM version can trivially simulate the SIMD version with no loss. Figure 1 depicts the relative power of the various models immediately apparent from the definitions, extending the results in Observation 2.2 to the hybrid models. Likewise, Table 2 presents additional separation results for the hybrid models.

3. The case for QRQW. The PRAM model was introduced in 1978 [22], with the CREW contention rule. Since that time, a variety of contention rules has been proposed and studied, with the most widely studied being the EREW, CREW, and CRCW rules. Variants of the CRCW PRAM such as ARBITRARY, COLLISION, COMMON, PRIORITY, ROBUST, and TOLERANT have been proposed and studied (see, e.g., [52] for definitions); these differ in their write-conflict rules. Given the plethora of contention rules already in the literature, it is reasonable to ask if there is a need for yet another contention rule, and in particular, whether the QRQW PRAM is an important new PRAM model.

The QRQW PRAM is a fundamental departure from standard PRAM models because it is *the first* PRAM *model to properly account for contention*, as reflected in most current commercial and research machines. By permitting contention, it reflects the realities of current machines, and enables simpler and more efficient algorithms for many basic problems. By charging for contention, it reflects the realities of machines with noncombining networks, e.g., most current commercial and research machines. In the remainder of this section, we elaborate on these points and then compare

the QRQW models to related work. We begin with a critique of the exclusive and concurrent rules.

3.1. EREW is too strict. The *exclusive* contention rule is almost universally considered by PRAM proponents to be a realistic rule for parallel machines. In the EREW PRAM, it is forbidden to have two or more processors attempt to read or write the same location in the same step. We know of no existing shared-memory parallel machine with this restriction on its global communication. Moreover, the exclusive rule leads to unnecessarily slow algorithms. A simple example is the 2-compaction problem, in which there are two nonempty cells at unknown positions in an array of size n, and the contents of these cells must be moved to the first two locations of the array. An EREW PRAM requires $\Omega(\sqrt{\log n})$ time to solve the 2-compaction problem; an n-processor CREW PRAM requires $\Omega(\log \log n)$ time [20]. However, as shown in section 7, there is a trivial constant-time n-processor QRQW PRAM algorithm for this problem.

The exclusive contention rule eliminates many *randomized* algorithmic techniques. Randomization used to determine *where* a processor should read or write (e.g., random sampling, random hashing) cannot avoid some small likelihood of concurrent reading or writing and hence cannot be incorporated directly into EREW algorithms.[4] Likewise, most *asynchronous* algorithms cannot avoid scenarios in which concurrent reading or writing occurs. Hence existing ASYNCHRONOUS PRAM models (e.g., [11, 24, 54, 50]) do not enforce the exclusive rule, assuming instead a CRCW cost measure.[5]

3.2. CRCW may be too powerful. At the other extreme, the *concurrent*-contention rule may be too powerful. In the CRCW PRAM, each step takes unit time, independent of the amount of contention in the step. Thus no distinction is made between low-contention and high-contention algorithms. On parallel machines with noncombining networks, high-contention read steps or write steps can be quite slow, as each of the requests for a highly contended location is serviced one by one, creating a serial bottleneck or "hot spot" [55]. Moreover, intermediate nodes on the path to the contended destination become congested as well, so a single hot spot can even delay requests destined for other nodes in the network. If all p processors request the same location, a common occurrence in CRCW PRAM algorithms, a direct implementation of the algorithm can incur a p-fold loss in speedup due to contention, sometimes becoming no better than a sequential algorithm.

An active area of research is how to execute a CRCW step that includes high-contention reads or writes without creating hot spots. Software approaches, e.g., using sorting [61], may incur an overhead considered unacceptable in practice, even on machines that support them. This is arguably true of the MasPar MP-1, for example, where the concurrent-write primitive provided for the MP-1 is around 20 times slower than writing according to a random permutation [56]. As indicated in section 1, the asymptotically best work-preserving emulation known for simulating the CRCW PRAM on machines with noncombining networks suffers polynomial slowdown [61, 63]. Thus, the running time on the parallel machine will be a polynomial factor slower than the running time indicated by the CRCW model.

[4] These techniques can be incorporated into CRCW algorithms, and emulated on the EREW, but at logarithmic cost in time and work.

[5] An exception is the EREW variant of Gibbons' ASYNCHRONOUS PRAM model [24], which permits contention in synchronization primitives, at a cost, but enforces the exclusive rule on reads and writes occurring between synchronization points.

TABLE 3
Contention rules of some existing multiprocessors.

Multiprocessor	$/P	A/S	Contention rule
Cray T3D [41]	$	A	QRQW
IBM SP2 [38]	$	A	QRQW
Intel Paragon [5]	$	A	QRQW
Kendall Square KSR1 [23]	$	A	CRQW
MasPar MP-1 [51], MP-2	$		
global router		S	QRQW
xnet		S	limited CREW
nCUBE 2S [59]	$	A	QRQW
Thinking Machines CM-5 [44]	$		
data network		A	QRQW
control network		S	fast SCAN ops
Bus-based machines	$	A	limited CRQW
Fluent [57, 1]	P	S	CRCW
MIT J-Machine [15]	P	A	QRQW
Stanford DASH [46]	P	A	QRQW
Tera Computer [2]	P	A	QRQW

Hardware approaches for executing high-contention CRCW steps without hot spots incorporate combining logic into the interconnection network. Ranade's work [57] shows that any CRCW step can be simulated on certain hypercube-based networks in the same asymptotic time as an EREW step, and development of machines based on his technique have been reported (e.g., [1, 18]). It is an open question whether the system cost of supporting CRCW efficiently in hardware is justified, particularly on MIMD machines, and work continues in this area (e.g., [16]). Existing commercial machines are primarily designed to process low-contention steps efficiently; high-contention steps are slow operations.

Note that the weaknesses of the exclusive- and concurrent-contention rules apply independently to reading and writing. Thus hybrids such as the CREW PRAM or the ERCW PRAM are too strict for writing (reading, respectively) and may be too powerful for reading (writing, respectively).

3.3. Most existing machines are QRQW. Table 3 classifies some existing multiprocessors according to the concurrent-read and write capabilities of their interprocessor communication. We have included message-passing machines, as well as shared-memory ones, since they are often used to run (slightly modified versions of) shared-memory algorithms or programs. The second column indicates commercial product ($) or working prototype (P). The third column indicates synchronous (S) or asynchronous (A) machines. In the last column, ER or EW denotes that programs for the machine are forbidden from having multiple requests for a location. QR or QW denotes that multiple requests to a location may be issued, and requests are generally serviced one at a time. CR or CW denotes that multiple requests to a location may be issued, and requests are combined in the network.

A few entries do not quite fit the taxonomy and require further explanation. In the XNET of the MP-1 and MP-2, processors are limited to reading or writing values stored at nodes a given distance away in a given compass direction; each processor may broadcast a value to all intermediate nodes on the path. The control network of the CM-5 provides fast SCAN primitives [6]; such primitives provide concurrent reading and writing and more (only) for well-structured sets of requests that fit the

segmented-SCAN paradigm [7]. In bus-based machines, the bus typically services only one shared-memory location at a time; all processors requesting to read the location can be serviced at the same time without penalty. Finally, a number of these machines provide caches that permit fast concurrent rereading of shared-memory locations: once a set of processors has read a location, they may subsequently reread the location without incurring a penalty for contention, as long as no processor has written to the location in the meantime.

As seen from the table, the contention rule for most of these machines, including the Cray T3D, IBM SP2, Intel Paragon, MIT J-Machine, nCUBE 2S, Stanford DASH, and Tera Computer, is well approximated by the QRQW rule. For the synchronous MasPar MP-1 and MP-2, the contention rule is well approximated by the SIMD-QRQW rule.

For the Kendall Square KSR1, the contention rule is well approximated by the CRQW rule. The Thinking Machines CM-5 provides a second network that can be used to perform fast SCAN operations [6]. An appropriate model for this machine would be a QRQW model with unit-time SCAN operations.

Note that each of the asynchronous machines (marked *A* in Table 3) allows for general asynchronous algorithms. Thus their contention rule in its full generality is well approximated by the asynchronous queue-contention rule provided by the QRQW ASYNCHRONOUS PRAM [30] (except for the KSR1, which is well approximated by an asynchronous CRQW contention rule). On the other hand, their contention rule with respect to bulk-synchronous algorithms is well approximated by the (bulk-synchronous) queue-contention rule provided by the simpler QRQW or CRQW PRAM.

A number of these machines, such as Stanford DASH, provide caches local to each processor; on reading a shared-memory location, a copy is stored in the processor's cache for future reuse. Multiple processors with cached copies of a location may then request to read the location and will be serviced in parallel from their local caches. To maintain a single consistent value for a location, these machines typically invalidate all cached copies of the location before permitting a processor to write to the location. This fast concurrent rereading of memory locations is not modeled in the QRQW models due to the following. If the contents of a shared-memory location are stored in a private-memory location when first read by a processor, then there is no need to issue a subsequent shared-memory read for this location unless some other processor *may have* changed the value: the private copy may be used instead. Moreover, if some other processor *did* change the value, then fast rereading is not possible and there will be a penalty for high contention with or without the caches. Thus fast rereading of memory locations seems to have only a secondary effect on the contention encountered in parallel algorithms, and hence has been omitted from the model, for simplicity.

We have conducted experiments to measure the effect of contention on a 16,384 processor MasPar MP-1. The results of these experiments are given in Figure 2. The experiments show that the SIMD-QRQW rule is a far more accurate reflection of running time on the MasPar MP-1 than a CRCW contention rule. Indeed, the overall time for the read (write) step is dominated by the cost of contention at a fairly small value for the contention, and then the time grows nearly linearly with the contention. In contrast, the CRCW contention rule would predict that the overall time would not change with the contention. The differences between the left and right plots in the figure demonstrate that charging k for contention k, as in the SIMD-QRQW rule, becomes an accurate reflection of the running time only when each processor has its

MasPar running times (in milliseconds)				
contention	1024 processors		16384 processors	
in step	write	read	write	read
1	0.563	0.518	7.321	6.849
2	0.595	0.554	7.435	6.957
4	0.755	0.703	7.415	6.944
8	1.414	1.332	7.449	6.976
16	2.765	2.589	7.870	7.369
32	5.445	5.090	10.283	9.636
64	10.784	10.116	15.354	14.391
128	21.503	20.167	25.952	24.329
256	42.922	40.271	47.127	44.205
512	85.761	80.459	89.746	84.194
1024	171.441	160.846	175.485	164.635
2048	—	—	346.781	325.357
4096	—	—	689.218	646.656
8192	—	—	1374.849	1289.970
16384	—	—	2744.192	2574.748

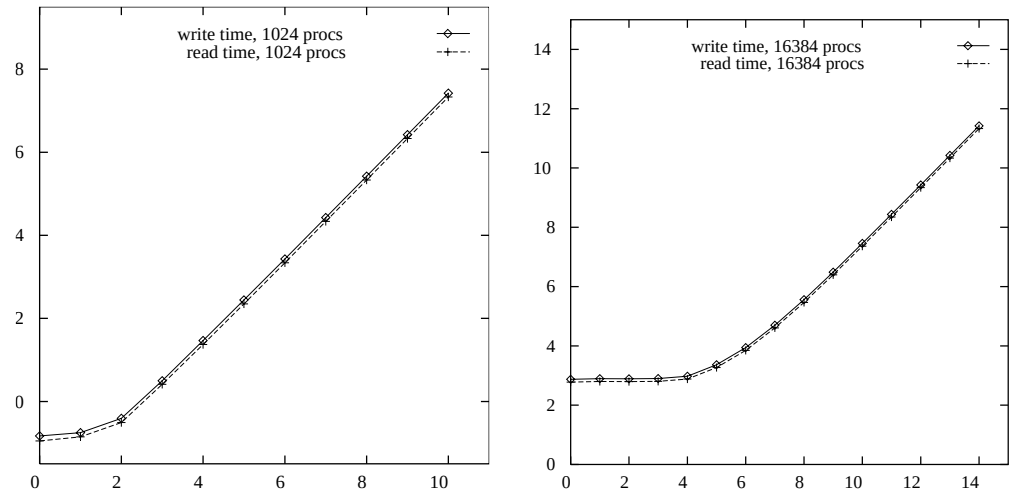


FIG. 2. *Performance measurements on the MasPar MP-*1 *for a read or write step, under increasing contention to a location. Top, timing measurements. Bottom, plot of the measurements on a* log-log *scale, showing the running time (y-coordinate) as a function of the contention in the step (x-coordinate). Results for* 2^{10} *and* 2^{14} *processors are shown. In the base experiment (contention* 1*, x-coordinate* 0*), each processor reads (writes) according to a random permutation. In the general experiment (contention* 2^i*, x-coordinate* i*), the first* 2^i *processors read (write) the same location* M*, while the remaining processors read (write) according to the original random permutation. Shown are the cumulative times of repeating the experiment on* 20 *different random permutations. In the plots, the y-coordinate depicts the base-*2 *logarithm of the number of milliseconds needed.*

The experiments show that high-contention steps are several orders of magnitude slower than random permutations, and moreover, that doubling the contention nearly doubles the running time, at least for medium- to high-contention steps. The dependence of the running time on the contention is more dramatic in the experiments with 1024 *processors than with* 16,384 *processors, for the following reason. In the* 16,384*-processor MasPar MP-*1*, each global router port is shared by* 16 *processors, creating an additional serial bottleneck. The experiments with* 1024 *processors use only one processor per port, thereby avoiding this serial bottleneck.*

own global router port; otherwise, a more complicated metric would be more accurate. Note that the MasPar MP-2, the successor of the MP-1, provides additional router

ports to help alleviate the bottleneck in the MP-1 caused by having one port for every 16 processors. Thus we would expect the MP-2 to behave more like the plot on the left, i.e., more according to the SIMD-QRQW rule.

3.4. Related work. In an early related work, Greenberg [36] considered broadcast communication schemes, such as the Ethernet, that have queues for submitted messages. More recently, Cypher [14] analyzed the performance of a maximum-finding algorithm under assumptions similar to the SIMD-QRQW PRAM. Dietzfelbinger, Kutylowski, and Reischuk [17] defined the *few-write* PRAM, that permits one-step concurrent writing of up to κ writes, where κ is a parameter of the model, as well as unlimited concurrent reading. Valiant [61] introduced the BSP model (see section 5) and studied a specialization of the model with logarithmic periodicity and constant throughput, which we call here the *standard* BSP model. In [61] it is shown that a v-processor PRAM step with contention κ can be simulated on a p-processor standard BSP in $O(v/p + \kappa \log p)$ time w.h.p. A large number of papers have studied the *Distributed Memory Machine*, in which the shared memory is partitioned into modules such that at most one memory location within each module can be accessed at a time. Concurrent reads and writes may or may not be allowed depending on the model. (See [43, 62] and the references therein.) An early example is the *Candidate Type Architecture* (*CTA*) machine model proposed by Snyder [60] which consists of a set of processors connected by a sparse communication network of unspecified topology and linked to a controller. The CTA is parameterized by the number of processors and the latency of interprocessor communication. Aumann and Rabin [3] showed that a PRAM algorithm can be simulated on a very general asynchronous parallel system that permits $O(\log n)$ contention to a location in unit time.

There have been several recent papers presenting independent work in related areas. Culler et al. [13] proposed the *LogP* model, a lower-level message-passing model in which there is limited communication bandwidth: a processor can send or receive at most one message every g cycles, where g is a parameter of the model. There is also a limit on the number of messages in the network at the same time. The LogP model permits general asynchronous algorithms. Liu, Aiello, and Bhatt [47] studied a message-passing model in which messages destined for the same processor are serviced one at a time in an arbitrary order. Their model permits general asynchronous algorithms, but each processor can have at most one message outstanding at a time. Dwork, Herlihy, and Waarts [19] defined an asynchronous shared-memory model with a *stall* metric. If several processes have reads or writes pending to a location, v, and one of them receives a response, then all the others incur a stall. Hence the charge for contention is linear in the contention, with requests to a location being serviced one at a time. Their model permits general asynchronous algorithms, but each processor can have at most one read or write outstanding at a time. Unlike their model, the QRQW models capture *directly* how the contention delays the overall running time of the algorithm, and are proposed as alternatives to other PRAM models for high-level algorithm design. Unlike each of these models, the QRQW PRAM does not explicitly limit the number of outstanding requests. The SIMD-QRQW PRAM, on the other hand, has the same restriction as the Liu, Aiello, and Bhatt [47] and Dwork, Herlihy, and Waarts [19] models, namely, one request per processor.

In contrast to many of the models mentioned above, the QRQW model focuses on the contention to locations, rather than to memory modules or processors. Any algorithm with high location contention will perform poorly on machines with non-combining networks, regardless of the number of memory modules; any lower bound

on location contention is a lower bound on memory-module contention. By focusing on locations, the QRQW model is independent of the particular layout of memory on the machine, e.g., the number of memory modules. Moreover, it is more relevant to cache-only memory architectures (COMA), such as the KSR1, that dynamically map memory locations to processors as the computation proceeds. Location contention is also a relevant metric for cache-coherence overhead, since the number of invalidates or updates that must be sent on a write is often proportional to the number of processors concurrently accessing the location being written [45]. The QRQW models, like the standard PRAM and other similar models, are true shared-memory models, providing a simple view of the shared memory as a collection of independent cells.

4. Adding contention to the work-time framework. In the *work-time* presentation, a parallel algorithm is described in terms of a sequence of steps, where each step may include any number of concurrent read, compute, or write operations [39]. In this context, the *work* is the total number of operations, and the *time* is the number of steps. This is sometimes the most natural way to express a parallel algorithm, and forms the basis of many data parallel languages (e.g., NESL [8]). For standard PRAM models, Brent's scheduling principle [10] can often be applied to obtain an efficient $O(work/p + time)$ time algorithm for a p-processor PRAM.

4.1. The QRQW work-time framework. We show here that the work-time paradigm can be used to advantage for the QRQW PRAM. It is extended into a QRQW work-time presentation by adding at each parallel step i the additional parameter k_i, the maximum contention at this step. Given an algorithm $\mathcal{A}$ in the QRQW work-time presentation, define the *work* to be the total number of operations[6] and the *time* to be the sum over all steps of the maximum contention k_i of each step (as in the SIMD-QRQW PRAM model). We note that one of the useful features of the traditional work-time presentation is that the time evaluation is independent of the work evaluation. Perhaps somewhat surprisingly, in the QRQW work-time presentation, too, the time evaluation (which is based on the contention at each step) is independent of the work evaluation: there is no benefit or loss in having steps with high contention also have high work, as long as the total contention and work remain the same. An algorithm given in the QRQW work-time presentation can be transformed into an efficient QRQW PRAM algorithm, as follows.

THEOREM 4.1. *Assume processor allocation is free. Any algorithm in the* QRQW *work-time presentation with x operations and time t (where t is the sum of the maximum contention at each step) runs in at most $x/p + t$ time on a p-processor* QRQW PRAM.

Proof. Let the number of parallel steps in the algorithm be r. Let x_i be the number of operations in the ith parallel step, and let $k_i \geq 1$ be the maximum contention in the ith parallel step, $1 \leq i \leq r$. Hence $t = \sum_{i=1}^{r} k_i$. We map the operations in the ith step uniformly onto the p QRQW PRAM processors. Thus each QRQW PRAM processor will receive at most $n_i = \lceil x_i/p \rceil$ operations. The maximum contention at any memory location remains the same as in the original work-time algorithm, i.e., at most k_i. Hence the time cost for the ith step on a p-processor QRQW PRAM is

[6] This contrasts with the *work* in a QRQW PRAM or SIMD-QRQW algorithm, which is the processor-time product.

$\max\{n_i, k_i\}$. The overall algorithm, therefore, takes time

$$\sum_{i=1}^{r} \max\{\lceil x_i/p \rceil, k_i\} \leq \sum_{i=1}^{r} ((x_i/p) + k_i) = x/p + t. \qquad \square$$

Thus Brent's scheduling principle can indeed be extended to the QRQW work-time framework.

4.2. Automatic processor allocation. The mechanism of translating an algorithm from a work-time presentation into a PRAM description is not addressed by Theorem 4.1, which assumes processor allocation is free. If the PRAM model is extended to include a unit-time SCAN operation [6], as may be appropriate for some machines such as the CM-5, then the processor allocation issue can be resolved with only small overhead. The rest of this section deals with the standard PRAM models that do not incorporate the SCAN operation.

Traditionally, the processor allocation needed to implement Brent's scheduling principle has been devised in an ad hoc manner. However, it is known that in several common situations an efficient automatic implementation is feasible, especially on the CRCW, often using linear-compaction and load-balancing algorithms as essential tools (see [52] and references therein). In this section, we adapt these techniques to the QRQW PRAM model.

Rather than tracing the details of each technique, it would be helpful to show that in general the contention parameter on the QRQW does not change the validity of these CRCW techniques. Indeed, the fact that time evaluation and work evaluation are done independently in the QRQW work-time presentation suggests that scheduling techniques on the CRCW PRAM should be useful for the QRQW PRAM as well. Next we elaborate on this issue.

Let $\mathcal{A}$ be a class of algorithms given in the QRQW work-time presentation. A QRQW *scheduling scheme* $\mathcal{S}_\mathcal{A}$ for $\mathcal{A}$ is a scheme that maps any algorithm A in $\mathcal{A}$ into a QRQW PRAM algorithm. If algorithm A has work-time bounds of w and t, then $\mathcal{S}_\mathcal{A}$ will convert A into a p-processor QRQW PRAM algorithm for some suitable number of processors p that runs in time $\tau = t + t_\mathcal{A} + (w + w_\mathcal{A})/p$ and work $\tau \cdot p$, where $t_\mathcal{A}$ and $w_\mathcal{A}$ are the overhead in time and work for the scheduling scheme $\mathcal{S}_\mathcal{A}$. The scheduling scheme $\mathcal{S}_\mathcal{A}$ is *work preserving* if $\tau \cdot p = O(w)$.

Similar definitions hold for a scheduling scheme for a class of CRCW PRAM algorithms given in the work-time presentation.

Consider a class of algorithms $\mathcal{B}$ given in a CRCW work-time presentation, and let $\mathcal{S}_\mathcal{B}$ be a scheduling scheme that adapts each algorithm B in $\mathcal{B}$ into a CRCW PRAM algorithm B'. Let $\mathcal{A}$ be the class of algorithms in the QRQW work-time presentation corresponding to $\mathcal{B}$. That is, each algorithm A in $\mathcal{A}$ is identical to an algorithm B in $\mathcal{B}$ except that the time of each parallel step is taken to be the maximum contention of that step. Thus algorithms A and B perform the same amount of work, though the running time of algorithm A could be larger. Let $\mathcal{S}_\mathcal{A}$ be a scheduling scheme on a QRQW PRAM corresponding to the CRCW PRAM scheduling scheme $\mathcal{S}_\mathcal{B}$. That is, the scheduling scheme $\mathcal{S}_\mathcal{A}$ adapts each algorithm A in $\mathcal{A}$ into a QRQW PRAM algorithm A' which, except for the scheduling overhead, is identical in execution (but not necessarily in time complexity) to the CRCW PRAM algorithm B' derived by $\mathcal{S}_\mathcal{B}$ from the algorithm B in $\mathcal{B}$ to which algorithm A corresponds.

LEMMA 4.2. *Let $w_\mathcal{A}, t_\mathcal{A}$ and $w_\mathcal{B}, t_\mathcal{B}$ be the work-time overhead of $\mathcal{S}_\mathcal{A}$ and $\mathcal{S}_\mathcal{B}$ respectively. If $\mathcal{S}_\mathcal{B}$ is work preserving on the* CRCW PRAM *and $w_\mathcal{A} = O(w_\mathcal{B})$ then*

$\mathcal{S}_{\mathcal{A}}$ is work preserving on the QRQW PRAM. *In particular, an algorithm A in $\mathcal{A}$ with work-time bounds of w and t will run optimally on a* QRQW PRAM *in time $O(w/q)$ using q processors when $q \leq w/(t + t_{\mathcal{A}})$.*

Proof. Let A correspond to a CRCW work-time algorithm B in $\mathcal{B}$ that runs in time t' with work w'. Note that $t \geq t'$ and $w = w'$ since A corresponds to B. On a p-processor CRCW PRAM, $\mathcal{S}_{\mathcal{B}}$ maps algorithm B to run in time $t' + t_{\mathcal{B}} + (w + w_{\mathcal{B}})/p$. Thus $p \cdot (t' + t_{\mathcal{B}} + (w + w_{\mathcal{B}})/p) = O(w)$ for some value of p, since $\mathcal{S}_{\mathcal{B}}$ is work preserving. This implies that $w_{\mathcal{B}} = O(w)$, and hence $w_{\mathcal{A}} = O(w)$.

Now let $\mathcal{S}_{\mathcal{A}}$ map algorithm A into a QRQW PRAM algorithm A' with $q \leq w/(t + t_{\mathcal{A}})$ processors. Then algorithm A' will run in time $\tau = t + t_{\mathcal{A}} + (w + w_{\mathcal{A}})/q$ on the q-processor QRQW PRAM, which gives the desired work-preserving schedule since

$$q \cdot \tau = q \cdot (t + t_{\mathcal{A}}) \ + \ w \ + \ w_{\mathcal{A}} \ \leq \ w + w + O(w) = O(w). \qquad \square$$

Note that we can always transform a CRCW PRAM scheduling scheme into an *equivalent* QRQW PRAM scheduling scheme simply by viewing the overhead of the CRCW scheduling scheme in the work-time framework and interpreting it as a (possibly slower) QRQW scheduling scheme with the same work overhead. This leads to the following corollary to Lemma 4.2.

COROLLARY 4.3. *Let $\mathcal{B}$ be a class of algorithms given in a* CRCW *work-time presentation and let $\mathcal{A}$ be a class of algorithms in the* QRQW *work-time presentation corresponding to $\mathcal{B}$. Let $\mathcal{S}_{\mathcal{B}}$ be a* CRCW *scheduling scheme for $\mathcal{B}$ and let $\mathcal{S}_{\mathcal{A}}$ be the equivalent* QRQW *scheduling scheme for $\mathcal{A}$. If $\mathcal{S}_{\mathcal{B}}$ is work preserving on the* CRCW PRAM *then $\mathcal{S}_{\mathcal{A}}$ is work preserving on the* QRQW PRAM.

Corollary 4.3 shows that it is always possible to derive a work-preserving QRQW scheduling scheme for a class of QRQW work-time algorithms corresponding to a class of CRCW work-time algorithms that have a work-preserving schedule. However, such a QRQW scheduling scheme can be very slow. In particular if the algorithm for the CRCW scheduling scheme has a read or write with concurrency $\Theta(w_{\mathcal{B}})$, where $w_{\mathcal{B}}$ is the work overhead of the CRCW scheduling scheme, then the work-preserving QRQW scheduling scheme degenerates into a sequential algorithm. A more useful way to apply Lemma 4.2 is to substitute a fast work-preserving QRQW PRAM algorithm for the QRQW scheduling scheme in place of the CRCW scheduling scheme.

In what follows, we give three examples of general classes of algorithms for which automatic processor allocation techniques can be applied to advantage: geometric-decaying algorithms, general task-decaying algorithms, and spawning algorithms. Processor allocation is done by a scheduling scheme that uses an algorithm for *linear (approximate) compaction*. The *linear-compaction* problem generalizes the 2-compaction problem as follows. Given k nonempty cells at unknown positions in an array of size n, with k known, move the contents of the nonempty cells to an output array of $O(k)$ cells. The linear-compaction problem can be solved by a randomized CRCW PRAM algorithm in time $T'_{lc}(n) = O(\log^* n)$ time and linear work w.h.p. [33]. In section 7 (Theorem 7.4) we show that the linear-compaction problem can be solved by a randomized SIMD-QRQW PRAM algorithm in time $T_{lc}(n) = O(\sqrt{\log n})$ and linear work w.h.p.

Sometimes the linear-compaction algorithm is used under the assumption that the number of nonempty cells is at most k. An unsuccessful termination of the algorithm is used to determine that the input consists of more than k nonempty cells. To make such a determination possible, it is necessary to employ an algorithm for computing the OR function, as well as an algorithm for the broadcasting problem. Furthermore,

recall that a subtle property of the QRQW models is that unsuccessful steps may turn out to be overly expensive if they incur (unexpected) high contention. (This is a rather significant technical issue in the algorithms of section 6.) We assume here that the number of nonempty cells never exceeds αk for some constant $\alpha > 0$, where k is the estimated upper bound. In such cases, the running time of the linear-compaction algorithm of Theorem 7.4 will increase by at most a constant factor. Let $T_{lcd}(n)$ be the running time of a linear-compaction algorithm followed by a determination of whether the algorithm was successful or not on an n-processor QRQW PRAM, and let $T''_{lcd}(n)$ be the corresponding running time on a CRQW PRAM.

In section 8 we show that on the QRQW PRAM broadcasting requires $\Omega(\log n)$ expected time. Therefore when it is necessary to determine if a run of linear compaction was unsuccessful on the QRQW PRAM, it is best to use a $\Theta(\log n)$ time EREW PRAM algorithm for prefix sums [42]. Hence, $T_{lcd}(n) = \Theta(\log n)$. Performing a broadcast on the SIMD-CRQW PRAM is trivial in constant time. In section 6 (Theorem 6.5) we show that the OR problem can be solved by a SIMD-CRQW PRAM in time $O(\log n/\log\log n)$ and linear work w.h.p. Hence, $T''_{lcd}(n) = O(\log n/\log\log n)$ w.h.p.

Task-decaying algorithms. A *task-decaying* algorithm (or simply a *decaying* algorithm) is one that starts with a collection of unit tasks. Each of these tasks progresses for a certain number of steps of the algorithm, and then dies. A task is said to be a *live* task until it dies. No other tasks are created during the course of the algorithm. The *work load* w_i is the number of live tasks at step i of the algorithm.

Geometric-decaying algorithms. A decaying algorithm in either the QRQW or the CRCW work-time presentation is *geometric-decaying* if the sequence of work loads $\{w_i\}$ is upper bounded by a decreasing geometric series. Typically the work w of such algorithms is $O(n)$, where n is the problem size.

Let $\mathcal{A}$ and $\mathcal{B}$ be the class of geometric-decaying algorithms in the QRQW and CRCW work-time presentations respectively. Using techniques from [31, 32, 53] and Lemma 4.2 we have the following theorem.

THEOREM 4.4. *Let A be a geometric-decaying algorithm in a* QRQW *work-time presentation with time t and work n. Then Algorithm A can be implemented on a p-processor* QRQW PRAM *to run in time $O(n/p)$ when $p = O(n/(t + T_{lc}(n)\log(T_{lc}(n))))$.*

Proof. Let B be a geometric-decaying algorithm in the CRCW work-time presentation to which Algorithm A corresponds. A work-preserving scheduling scheme $\mathcal{S}_{\mathcal{B}}$ that can adapt Algorithm B into a p-processor CRCW PRAM algorithm B' is given in [53]. The scheduling scheme $\mathcal{S}_{\mathcal{B}}$ consists of $\log(n/p)$ applications of an algorithm for a linear-compaction problem of size p. On the QRQW PRAM we will use a scheduling scheme $\mathcal{S}_{\mathcal{A}}$ corresponding to $\mathcal{S}_{\mathcal{B}}$. When mapping into a p-processor QRQW PRAM, scheduling scheme $\mathcal{S}_{\mathcal{A}}$ will consist of $\log(n/p)$ applications of a QRQW PRAM algorithm for the linear-compaction problem of size p. The time overhead incurred by scheduling scheme $\mathcal{S}_{\mathcal{A}}$ is $t_{\mathcal{A}} = O(T_{lc}(p)\log(n/p))$, and the work overhead is $p \cdot t_{\mathcal{A}}$. We observe, as in [53], that if $T_{lc}(p)\log(n/p) \geq n/p$, then $\log(n/p) = O(\log(T_{lc}(n)))$, and hence for $p \leq n/(T_{lc}(p)\log(T_{lc}(n)))$, scheduling scheme $\mathcal{S}_{\mathcal{A}}$ has a work overhead of $O(n)$. Therefore, by Lemma 4.2, $\mathcal{S}_{\mathcal{A}}$ maps algorithm A into a p-processor QRQW PRAM algorithm A' to run in time $O(n/p)$ provided $p = O(n/(t + T_{lc}(p)\log(T_{lc}(n))))$. □

By Theorem 4.4 and the result for linear-compaction shown in section 7 (i.e., Theorem 7.4) we obtain the following corollary.

COROLLARY 4.5. *Algorithm A in* Theorem 4.4 *can be implemented on a p-processor* QRQW PRAM *to run in time $O(n/p)$ w.h.p. when $p = O(n/(t+\sqrt{\log n}\log\log n))$.*

General task-decaying algorithms. Recall that in a task-decaying algorithm

in either the QRQW or the CRCW work-time presentation, the sequence of work loads $\{w_i\}$ is a monotonically nonincreasing series. Thus, task-decaying algorithms generalize geometric-decaying algorithms. A task-decaying algorithm is *predicted* if an approximate bound on the sequence of work loads $\{w_i\}$ is known in advance; specifically, if a sequence $\{w'_i\}$ is given such that for all i, $w'_i \geq w_i$ and $\sum_i w'_i = O(\sum_i w_i)$. Let $\mathcal{A}$ and $\mathcal{B}$ be the class of general task-decaying algorithms in the QRQW and CRCW work-time presentations respectively.

THEOREM 4.6. *Let A be a task-decaying algorithm in a* QRQW *work-time presentation with time t and work n. Then Algorithm A can be implemented to run in time $O(n/p)$ on a p-processor* QRQW PRAM *when $p = O(n/(t + T_{lcd}(n)\log(T_{lcd}(n))))$ and on a p-processor* CRQW PRAM *when $p = O(n/(t + T''_{lcd}(n)\log(T''_{lcd}(n))))$. If Algorithm A is also predicted then it can be implemented on a p-processor* QRQW PRAM *to run in time $O(n/p)$ when $p = O(n/(t + T_{lc}(n)\log(T_{lc}(n))))$.*

Proof. Let B be a predicted task-decaying algorithm in a CRCW work-time presentation to which Algorithm A corresponds. A work-preserving scheduling scheme $\mathcal{S}_{\mathcal{B}}$ that can adapt Algorithm B into a p-processor CRCW PRAM algorithm B' is given in [53]. The scheduling scheme $\mathcal{S}_{\mathcal{B}}$ is based on several applications of an algorithm for the linear-compaction problem of size p. The analysis in [53] is based on showing that the cost of all but $\log(n/p)$ applications of the linear-compaction algorithm can be amortized against the execution of Algorithm B, with only a constant factor overhead. Hence the time overhead of $\mathcal{S}_{\mathcal{B}}$ is $t_{\mathcal{B}} = O(T'_{lc}(n)\log(n/p))$. As for the geometric-decaying algorithm, the time overhead can be shown to be $t_{\mathcal{B}} = O(T'_{lc}(n)\log(T'_{lc}(n)))$.

Consider a scheduling scheme $\mathcal{S}_{\mathcal{A}}$, corresponding to $\mathcal{S}_{\mathcal{B}}$, which adapts Algorithm A to a p-processor QRQW PRAM algorithm A'. An amortization argument similar to the one used for $\mathcal{S}_{\mathcal{B}}$ implies that the cost of all but $\log(n/p)$ applications of the linear-compaction algorithm can be amortized against the execution of Algorithm A, with only a constant factor overhead. The time overhead of $\mathcal{S}_{\mathcal{A}}$ is therefore $t_{\mathcal{A}} = O(T_{lc}(n)\log(n/p))$, and hence $t_{\mathcal{A}} = O(T_{lc}(n)\log(T_{lc}(n)))$, and the work overhead is $p \cdot t_{\mathcal{A}}$. Hence for $p = O(n/(T_{lc}(n)\log(T_{lc}(n))))$ this schedule has a work overhead of $O(n)$. By Lemma 4.2 the scheduling scheme $\mathcal{S}_{\mathcal{A}}$ maps A into a p-processor QRQW PRAM in $O(n/p)$ time provided $p = O(n/(t + T_{lc}(n)\log(T_{lc}(n))))$.

If Algorithm B is not predicted then each application of the linear compaction algorithm must be followed by a detection of whether there was a successful termination. In such a case, the underestimation is by at most a factor of two. Similar arguments to the above imply that the corresponding algorithm A can be adapted to a QRQW PRAM algorithm with running time $O(n/p)$ provided $p \leq n/(t + T_{lcd}(n)\log(T_{lcd}(n)))$ and to a CRQW PRAM algorithm with running time $O(n/p)$ provided $p \leq n/(t + T''_{lcd}(n)\log(T''_{lcd}(n)))$ ∎

By the result stated above we have the following corollary.

COROLLARY 4.7. *Algorithm A in* Theorem 4.6 *can be implemented to run in time $O(n/p)$ w.h.p. on a p-processor* QRQW PRAM *when $p = O(n/(t + \log n \log\log n))$ and on a p-processor* CRQW PRAM *when $p = O(n/(t + \log n))$. If Algorithm A is predicted then it can be implemented on a p-processor* QRQW PRAM *to run in time $O(n/p)$ w.h.p. when $p = O(n/(t + \sqrt{\log n}\log\log n))$.*

Spawning algorithms. A *spawning* algorithm starts with a collection of unit tasks, and at each step of the algorithm, each task can

i. progress to the next step of the algorithm;

ii. progress to the next step of the algorithm *and* spawn another new task; or

iii. not progress to the next step and die.

The total number of tasks in a spawning algorithm may increase or decrease in each step. Thus, the spawning model generalizes the model for task-decaying algorithms. As in the task-decaying model, a spawning algorithm is *predicted* if an approximate bound on the sequence of work loads $\{w_i\}$ is known in advance; specifically, if a sequence $\{w'_i\}$ is given such that for all i, $w'_i \geq w_i$ and $\sum_i w'_i = O(\sum_i w_i)$.

THEOREM 4.8. *Let A be a spawning algorithm in a* QRQW *work-time presentation running in time t and work n, and let t' be the number of parallel steps in A. Then Algorithm A can be implemented to run in time $O(n/p)$ on a p-processor* QRQW PRAM *when $p = O(n/(t+t' \cdot T_{lcd}(n)))$ and on a p-processor* CRQW PRAM *when $p = O(n/(t+t' \cdot T''_{lcd}(n)))$. If Algorithm A is also predicted then it can be implemented to run in time $O(n/p)$ on a p-processor* QRQW PRAM *when $p = O(n/(t+t' \cdot T_{lc}(n)))$.*

Proof. Let B be a predicted spawning algorithm in a CRCW work-time presentation to which Algorithm A corresponds. Then the running time of Algorithm B is t'. A work-preserving scheduling scheme $\mathcal{S}_\mathcal{B}$ that can adapt Algorithm B into a p-processor CRCW PRAM algorithm B' is given in [52]. The scheduling scheme $\mathcal{S}_\mathcal{B}$ consists of applying an algorithm for a linear-compaction problem of size p a constant number of times after each parallel step. The time overhead of $\mathcal{S}_\mathcal{B}$ is therefore $O(t' \cdot T'_{lc}(n))$.

Consider a scheduling scheme $\mathcal{S}_\mathcal{A}$, corresponding to $\mathcal{S}_\mathcal{B}$, which adapts Algorithm A to a p-processor QRQW PRAM algorithm A'. The scheduling scheme $\mathcal{S}_\mathcal{A}$ consists of applying an algorithm for a linear-compaction problem of size p a constant number of times after each parallel step. The time overhead incurred by $\mathcal{S}_\mathcal{A}$ is thus $t_\mathcal{A} = O(t' \cdot T_{lc}(n))$ and the work overhead is $w_\mathcal{A} = p \cdot t_\mathcal{A}$. Hence by Lemma 4.2 algorithm A' runs in time $O(n/p)$ on a p-processor QRQW PRAM provided $p = O(n/(t+t' \cdot T_{lc}(n)))$.

If Algorithm B is not predicted then, as in the case of the task-decaying algorithm of Theorem 4.6, each application of the linear-compaction algorithm must be followed by a detection of whether there was a successful termination. Similar arguments to the above imply that the corresponding algorithm A can be adapted to a p-processor QRQW PRAM algorithm running in time $O(n/p)$ provided $p = O(n/(t+t' \cdot T_{lcd}(n)))$, and to a p-processor CRQW PRAM algorithm running in time $O(n/p)$ provided $p = O(n/(t+t' \cdot T''_{lcd}(n)))$. □

COROLLARY 4.9. *Algorithm A in* Theorem 4.8 *can be implemented to run in time $O(n/p)$ w.h.p. on a p-processor* QRQW PRAM *when $p = O(n/(t+t' \cdot \log n))$ and on a p-processor* CRQW PRAM *when $p = O(n/(t+t' \cdot \log n/\log\log n))$. If Algorithm A is predicted then it can be implemented to run in time $O(n/p)$ w.h.p. on a p-processor* QRQW PRAM *when $p = O(n/(t+t' \cdot \sqrt{\log n}))$.*

The spawning model can be further generalized to include a *start* operation in which one task may spawn n new tasks to begin in the next time step. This extended model is called V-PRAM in [35], where it was suggested. It was shown in [35] that the work-preserving scheme for the spawning model can be extended to the V-PRAM model as well, with the same overhead. Accordingly, Theorem 4.8 and Corollary 4.9 apply to the V-PRAM model.

A more general type of spawning algorithm, the *L-spawning algorithm*, is studied in [29]. In the L-spawning model, each task can spawn up to $L-1$ additional tasks at each step. It is shown in [29] that an L-spawning algorithm with time t, work n, and t' parallel steps can be implemented on a p-processor QRQW PRAM to run in time $O(n/p)$ w.h.p. when $p = O(n/(t+t'\sqrt{\log n}\log\log L + t'\log L))$. This implementation applies a more general load-balancing algorithm given in [29].

5. Realization on feasible networks. The BSP model was introduced by Valiant [61, 62] as a model of parallel computation that takes into account overheads incurred by latency, synchronization, and memory granularity. It consists of components that can perform local RAM computations and communicate with one another through a router which delivers messages between pairs of components. Messages to a component are serviced one at a time. The BSP provides facilities for synchronizing the components at regular intervals. There are three parameters to the model: p, the number of components, *periodicity* L, the number of time units between synchronizations, and *throughput* g, a measure of the bandwidth limitations of the router. A particular case studied by Valiant is one that sets g to be a constant and L to be $\Theta(\log p)$, and has each synchronization involve all the components; we denote this the *standard* BSP model.

A standard BSP computation consists of a sequence of supersteps, with each superstep separated from the next by a global synchronization point among all the components. In each superstep, each component sends messages, receives messages, and performs local RAM steps. Operations at a component (message initiations, message receipts, RAM operations) are assumed to take constant time. No assumption is made about the relative delivery times of messages within a superstep, and local operations may only use data values locally available to the component prior to the start of the superstep. If the operations in a superstep, including message deliveries, do not complete in L time units, additional intervals of L time units are allocated to the superstep until it completes.

The BSP model has been advocated as one that forms a bridge between software and hardware in parallel machines, that is, between abstract models for algorithm design and realistic parallel machines. This approach is supported in [61, 62] by providing a fast, work-preserving emulation of the standard BSP model on hypercube-type noncombining networks on the one hand, and a fast, work-preserving emulation of the EREW PRAM on the standard BSP on the other hand. In particular, it is shown that the EREW PRAM can be emulated in a work-preserving manner with logarithmic slowdown on the standard BSP, while the standard BSP can be emulated in a work-preserving manner with constant slowdown on, e.g., the multiport hypercube. In the multiport hypercube on p nodes, each node can receive a message on each of its $\log p$ incoming wires and route them along the appropriate outgoing wires in constant time, subject to the constraint that at most one message can be sent along each outgoing wire. These emulations show that the choice of $L = \Theta(\log p)$ and $g = \Theta(1)$ used in the standard BSP is sufficient to hide the latency, synchronization, and memory granularity overheads occurring in the emulations.

Valiant [61] shows that a v-processor PRAM step with contention κ can be simulated on a p-processor standard BSP in $O(v/p + \kappa \log p)$ time w.h.p. It follows readily from this result that a p-processor SIMD-QRQW PRAM algorithm running in time t can be emulated on a $(p/\log p)$-component standard BSP model in $O(t \log p)$ time w.h.p.

In this section we show that the more powerful QRQW PRAM can also be emulated in a work-preserving manner with only logarithmic slowdown on the standard BSP as well as on hypercube-type networks. The proof of this result is complicated by the fact that a QRQW step with time cost k may have up to $2kp$ reads and writes, whereas in the previous emulation results, the PRAM step being emulated had at most $2p$ reads and writes, independent of k. As in the previous emulations of PRAM models on the standard BSP given in [61], we apply a random hash function to map the shared memory of the PRAM onto the BSP components; this function is assumed to map

each shared-memory location to a component chosen uniformly and independently at random.

THEOREM 5.1. *A p-processor* QRQW PRAM *algorithm (or* SIMD-QRQW PRAM *algorithm) running in time t, where t is polynomial in p, can be emulated on a $(p/\log p)$-component standard* BSP *model in $O(t \log p)$ time w.h.p.*

Proof. As stated above we apply a hash function that maps the shared memory of the QRQW PRAM to the BSP components such that each shared-memory location is mapped on to a BSP component chosen uniformly and independently at random. We first show that for each QRQW step with time cost k, the number of memory requests mapped to any BSP component is $O(k \log p)$ w.h.p. Then we use this claim to argue that the time to emulate the step on the BSP is $O(k \log p)$ w.h.p., and hence the time to emulate all the QRQW steps is $O(t \log p)$ w.h.p.

Consider the ith step of the QRQW PRAM algorithm, with time cost k_i. For simplicity of exposition, we assume that each processor has exactly k_i shared-memory accesses, where an access is either a read or a write. Let $m_1, \ldots, m_d$ be the different memory locations accessed in this step, and let q_j be the number of accesses to location m_j, $1 \leq j \leq d$. For the purpose of this analysis we add $\delta p k_i$ memory accesses to this step, for a constant $\delta \geq 23$, consisting of accesses with contention k_i to locations $m_{d+1}, \ldots, m_{d'}$, where $d' = d + \delta p$. With this addition, the ith step has $v_i{}' = (\delta+1)pk_i$ concurrent accesses to d' different memory locations, and the maximum contention is k_i. We set $q_j = k_i$ for $d+1 \leq j \leq d'$ and note that $v' = \sum_{j=1}^{d'} q_j$. We now show that the bound stated in the theorem holds for this augmented problem. Clearly, this implies that the bound holds for the original problem.

As indicated earlier we assume that the memory has been randomly hashed onto the $p/\log p$ components of the BSP. Consider a fixed component C. As in [61], we define a random variable $x_j, 1 \leq j \leq d'$, where $x_j = q_j/k_i$ if m_j is hashed onto C and zero otherwise. Let $X = \sum_{j=1}^{d'} x_j$. We note that $x_j = q_j/k_i$ with probability $\log p/p$, and $k_i \cdot X$ is the number of messages sent to C in the ith step. Then

$$E(x_j) = q_j \log p/(pk_i), \quad 1 \leq j \leq d' \ .$$

Let μ be the mean of the expectations of the x_j:

$$\mu = \sum_{j=1}^{d'} (q_j \log p)/(pk_i d') = v_i{}' \log p/(pk_i d') = (\delta+1)pk_i \log p/(pk_i d') \ .$$

So $\mu = (\delta+1)\log p/d'$. By Hoeffding's inequality [37],

$$Pr(X > (\mu + z)d') \leq e^{-z^2 d'/3\mu} \ ,$$

provided $z < \min(\mu, 1-\mu)$. Let $z = \mu/2$. Then

$$Pr\ (X > 3\mu d'/2) \leq e^{-\mu d'/12} = e^{-(\delta+1)\log p/12} = 1/p^{\Theta(\delta)} \ .$$

Let $t = O(p^r)$, and let $c > 0$ be an arbitrary constant. By choosing δ sufficiently large, we have that the probability that any component receives more than $3\mu d' k_i/2 = \Theta(k_i \log p)$ messages in the ith QRQW step is less than $1/p^{r+c}$.

Each BSP component emulates $\log p$ QRQW PRAM processors. It sends $O(k_i \log p)$ "read" messages and receives $O(k_i \log p)$ (w.h.p.) such messages. In the next superstep, it sends $O(k_i \log p)$ (w.h.p.) "read reply" messages and receives $O(k_i \log p)$ such

replies. Finally, in the last superstep, it performs $O(k_i \log p)$ local RAM operations, sends $O(k_i \log p)$ "write" messages, and receives $O(k_i \log p)$ (w.h.p.) such messages, updating the values of the appropriate locations. Since the periodicity L is $\Theta(\log p)$ and the gap g is constant, the time taken to complete the ith step on the BSP is $O(k_i \log p)$ w.h.p.

Thus, with probability greater than $(1 - 1/p^c)$ the BSP completes the emulation of the $O(t)$ time augmented QRQW computation in $O(\sum_{i=1}^{m} k_i \log p)$ time, where m is the number of steps in the QRQW computation, i.e., the BSP completes the emulation in $O(t \log p)$ time w.h.p. □

Note that unlike Valiant's emulation of the EREW PRAM on the standard BSP, the emulation above may result in a rather uneven distribution of messages among the components whenever there is an uneven distribution of contention among the locations. This raises concerns regarding possible contention in routing the messages between the components. However, the (standard) BSP model ignores all issues of routing other than the number of messages sent and received at each component, and hence the proof of Theorem 5.1 addresses only these same routing issues.

Further issues in routing do arise in emulating the PRAM or BSP on models such as the multiport hypercube. Valiant defines the *slackness* of a parallel algorithm being emulated to be the ratio of the number of virtual processors in the algorithm to the number of "physical" processors in the emulating model. In [61], Valiant showed that a p-component standard BSP algorithm with slackness at least $\log p$ and running in time t can be emulated on a p-node multiport hypercube in $O(t)$ time w.h.p. Since the slackness in the emulation in Theorem 5.1 is $\log p$, we have the following theorem.

THEOREM 5.2. *A p-processor* QRQW PRAM *algorithm (or* SIMD-QRQW PRAM *algorithm) running in time t can be emulated on a $(p/\log p)$-node multiport hypercube in $O(t \log p)$ time w.h.p.*

Thus the uneven distribution of messages that may result from emulating a QRQW PRAM algorithm on the standard BSP does not prevent a fast, work-preserving emulation of the QRQW PRAM on the multiport hypercube.

6. Leader election and computing the OR. Given a Boolean array of n bits, the OR function is the problem of determining if there is a bit with value 1 among the n input bits. The *leader election* problem is the problem of electing a leader bit from among the k out of n bits that are 1 (k unknown). The output is the index in $[1..n]$ of the bit if $k > 0$, or 0 if $k = 0$. This generalizes the OR function, as long as $k = 0$ is possible.

In this section we present several randomized and deterministic algorithms for solving these problems on queue-write PRAMs. Our main result is a randomized algorithm for the two problems on the CRQW PRAM that performs linear work and runs in $O(\log n/\log\log n)$ time w.h.p. This result is somewhat surprising since it improves on the best possible time bound (which is $\Theta(\log n)$) for any deterministic or randomized CREW PRAM algorithm for the two problems.

Most of the randomized algorithms we present are of the Las Vegas type, while a few are of the Monte Carlo type. A *Las Vegas* algorithm is a randomized algorithm that always outputs a correct answer and obtains the stated bounds with some stated probability. A *Monte Carlo* algorithm, in contrast, is a randomized algorithm that outputs a correct answer with some stated probability. In the analysis of some of our randomized algorithms, we apply the Chernoff bound

$$Pr\{X \geq \beta E[X]\} \leq e^{(1-1/\beta-\ln\beta)\beta E[X]} \quad \text{for all } \beta > 1 \ ,$$

and in particular, its following corollary.

OBSERVATION 6.1. *Let X be a binomial random variable. For all $f = O(\log n)$, if $E[X] \leq 1/2^f$, then $X = O(\log n/f)$ w.h.p. Furthermore, if $E[X] \leq 1$, then $X = O(\log n/\log\log n)$ w.h.p.*

Proof. Let $\beta = c\log n/(fE[X])$ for a constant $c > \max\{2, f/\log n\}$ to be determined. Then $\beta > 1/E[X] \geq 2^f$, since $\beta E[X] = c\log n/f > 1$. By the Chernoff bound,

$$\begin{aligned} Pr\{X \geq c\log n/f\} &\leq e^{(1-1/\beta-\ln\beta)\cdot(c/f)\log n} < e^{-(c/2f)\ln\beta\log n} \\ &= e^{-(c/2f)\log\beta\cdot\ln n} = 1/n^{(c/2f)\log\beta} < 1/n^{c/2} \ . \end{aligned}$$

Hence for any $\delta > 1$, there exists a constant $c = \max\{2\delta, f/\log n\}$ such that $Pr\{X \geq c\log n/f\} < 1/n^\delta$.

If $E[X] \leq 1$, we take $\beta = c\log n/(\log\log nE[X])$, for a constant $c > 2$ to be determined. Then $\log\beta \geq \log\log n - \log\log\log n \geq 2\log\log n/3$. By the Chernoff bound,

$$\begin{aligned} Pr\{X \geq c\log n/\log\log n\} &\leq e^{(1-1/\beta-\ln\beta)\cdot(c/\log\log n)\log n} < e^{-(c/2\log\log n)\ln\beta\log n} \\ &= e^{-(c/2\log\log n)\log\beta\cdot\ln n} = 1/n^{(c/2\log\log n)\log\beta} \\ &\leq 1/n^{c/3} \ . \end{aligned}$$

Hence for any $\delta > 1$, there exists a constant $c = 3\delta$ such that $Pr\{X \geq c\log n/\log\log n\} < 1/n^\delta$. □

6.1. Deterministic algorithms. By having each processor whose input bit is 1 write the index of the bit in the output memory cell, we obtain a simple deterministic SIMD-ERQW PRAM algorithm for leader election (and similarly for the OR function) that runs in $\max\{1, k\}$ time using n processors, where k is the number of input bits that are 1 (k unknown). This is a fast algorithm if we know in advance that the value of k is small. However, for the general leader election problem, a better algorithm is the natural EREW PRAM algorithm for leader election which uses a parallel prefix algorithm to compute the location of the first 1 in the input; this takes $\Theta(\log n)$ time and $\Theta(n)$ work.

We can derive an $\Omega(\log n/\log\log n)$ lower bound for the OR function using a lower bound result of Dietzfelbinger, Kutylowski, and Reischuk [17] for the *few-write* PRAM. Recall that the few-write PRAM models are parameterized by the number of concurrent writes to a location permitted in a unit-time step. (Exceeding this number is not permitted.) Let the *κ-write* PRAM denote the few-write PRAM model that permits concurrent writing of up to κ writes to a location, as well as unlimited concurrent reading. We begin by proving a more general result for emulating the CRQW on the few-write PRAM, and then provide the OR lower bound.

OBSERVATION 6.2. *A p-processor* CRQW PRAM *deterministic algorithm running in time t can be emulated on a p-processor t-write* PRAM *in time $O(t)$.*

Proof. Since the CRQW algorithm runs in time at most t on all inputs, then the maximum write contention is at most t on all inputs. Hence the t-write PRAM can be used to emulate each write substep, and the emulation proceeds as was done for the CRCW (Observation 2.2). □

THEOREM 6.1. *Any deterministic algorithm for computing the* OR *function on a* CRQW PRAM *with arbitrarily many processors requires $\Omega(\log n/\log\log n)$ time.*

Proof. Dietzfelbinger, Kutylowski, and Reischuk [17] proved an $\Omega(\log n/\log\kappa)$ lower bound for the OR function on the κ-write PRAM. Let T be the time for the

OR function on the CRQW PRAM. Then by Observation 6.2, the OR function can be computed on the T-write PRAM in $O(T)$ time. Thus $T = \Omega(\log n / \log T)$, and hence $T \log T = \Omega(\log n)$. Now if $T = o(\log n / \log\log n)$, then $\log T = o(\log\log n)$, contradicting $T \log T = \Omega(\log n)$. Thus $T = \Omega(\log n / \log\log n)$. □

Since the ERCW PRAM can compute the OR function in constant time, Theorem 6.1 implies the following separation result.

COROLLARY 6.2. *There is an $\Omega(\log n / \log\log n)$ time separation of a deterministic* {ER,QR,CR}CW PRAM *from a deterministic* {ER,QR,CR}QW PRAM.

Cook, Dwork, and Reischuk [12] proved that any deterministic algorithm for computing the OR function on a CREW PRAM with arbitrarily many processors requires $\Omega(\log n)$ time. Dietzfelbinger, Kutylowski, and Reischuk [17] later proved a similar lower bound for randomized CREW PRAM algorithms. The difficulty in extending either of these results to the CRQW PRAM is that in the CRQW PRAM, the running time of a step may be different on different inputs. Thus in a CRQW write step with contention k for a given input I, the lower bound argument of [12, 17] will allow processors to gain knowledge about input I as a function of the maximum contention, K, for the step over *all* inputs, and K could be much larger than k.

6.2. Randomized algorithms for special cases. In this subsection, we present a series of randomized leader election algorithms, under various scenarios. First, consider the leader election problem when the value of k is known. On the SIMD-QRQW PRAM, a simple, fast, randomized algorithm for this problem is to have the k processors whose input bits are 1 write to the output cell with probability $1/k$. This runs in constant time on the SIMD-QRQW, and, as a low-contention algorithm, will run fast in practice. The failure probability can be reduced by repeating the algorithm.

OBSERVATION 6.3. *Consider the problem of electing a leader bit from among the k out of n bits that are 1, where k is known. There is a (randomized) Monte Carlo* SIMD-ERQW PRAM *algorithm that runs in $O(1)$ expected time and $O(n)$ expected work, and probability of failure less than $1/e$. There is a (randomized) Las Vegas* SIMD-CRQW PRAM *algorithm that runs in $O(1)$ expected time and $O(n)$ expected work.*

Proof. The index of each bit whose value is 1 is written into the output cell with probability $1/k$. This has constant expected contention, and the probability that no value is written is $(1-1/k)^k < 1/e$. To obtain a Las Vegas algorithm, the write step is repeated until there is at least one writer. Termination is detected by using the concurrent-read capability. The expected time is $O(1+1/e+1/e^2+1/e^3+\cdots)$, which is $O(1)$. □

The expected time for this algorithm is constant; however, we are interested in high-probability results. The next two theorems deal with high-probability randomized algorithms for the case when a good estimate for k is known, and the case when a good upper bound for the value of k is known.

Given a good estimate for *k*. In the following, we describe a fast leader election algorithm when the number of bits competing for leadership is known to within a multiplicative factor of $2^{\sqrt{\log n}}$.

THEOREM 6.3. *Consider the problem of electing a leader bit from among the k out of n bits that are 1. Let $\hat{k}$ be known to be within a factor of $2^{\sqrt{\log n}}$ of k, i.e., $\hat{k}/2^{\sqrt{\log n}} \le k \le \hat{k}2^{\sqrt{\log n}}$. There is a Monte Carlo* SIMD-ERQW PRAM *algorithm that, w.h.p., elects a leader in $O(\sqrt{\log n})$ time with $O(n)$ work. On the* SIMD-CRQW PRAM, *or if $\hat{k} \le 2^{\sqrt{\log n}}$, the same bounds can be obtained for a Las Vegas algorithm.*

Proof. We describe the algorithm for $n/\sqrt{\log n}$ processors. Let $p = \min(1, \frac{2^{c\sqrt{\log n}}}{\hat{k}})$,

for a constant $c \geq 1$ to be determined by the analysis. Let A be an array of size $m = 2^{(c+2)\sqrt{\log n}}$, initialized to all zeros. The input bits are partitioned among the processors such that each processor is assigned $\sqrt{\log n}$ bits.

Step 1. Each processor selects a leader from among its input bits that are 1, if any.

Step 2. Each processor with a leader writes, with probability p, the index of the leader bit to a cell of A selected uniformly at random.

Step 3. m of the processors participate to select a nonzero index from among those written to A.

If $\hat{k} \leq 2^{\sqrt{\log n}}$, then $p = 1$ and this is a Las Vegas algorithm. Otherwise a Las Vegas algorithm is obtained by repeating steps 2 and 3 until there is a nonzero index in A. Termination is detected by using the concurrent-read capability.

Step 1 takes $O(\sqrt{\log n})$ time. Since $m = 2^{O(\sqrt{\log n})}$, an EREW binary fanin approach can be used to obtain the same time bounds for step 3. For step 2, we will show that the contention is $O(\sqrt{\log n})$ w.h.p. Let X_i be the number of writers to cell i of A. Then

$$E[X_i] \leq kp/m \leq k2^{c\sqrt{\log n}}/\hat{k}m \leq k/\hat{k}2^{2\sqrt{\log n}} \leq 1/2^{\sqrt{\log n}} \ .$$

It follows from Observation 6.1 that the maximum contention over all cells of A is $O(\sqrt{\log n})$ w.h.p.

It remains to show that w.h.p., there is at least one writer to A (assuming that $k > 0$). If $\hat{k} \leq 2^{c\sqrt{\log n}}$, then $p = 1$ and hence there will be one writer to A for each processor that has an input bit that is 1. Otherwise $\hat{k} > 2^{c\sqrt{\log n}}$, and the probability that there are no writers to A is at most

$$\begin{aligned}(1-p)^{k/\sqrt{\log n}} &= ((1-1/(1/p))^{1/p})^{pk/\sqrt{\log n}} < (1/e)^{pk/\sqrt{\log n}}, \\ &= (1/e)^{(k/\hat{k})2^{c\sqrt{\log n}}/\sqrt{\log n}} \leq (1/e)^{2^{(c-1)\sqrt{\log n}}/\sqrt{\log n}}.\end{aligned}$$

It follows that c can be chosen so that there is at least one writer w.h.p. □

Given an upper bound on k. We next consider the case where we only have an upper bound, k_{max}, on the number of input bits that are 1; the results we obtain are not quite as good as when k is known to within a factor of $2^{\sqrt{\log n}}$, but better than the case when no bound on k (other than n) is known. The algorithm is a straightforward modification of the previous algorithm (Theorem 6.3).

THEOREM 6.4. *Consider the problem of electing a leader bit from among k out of n bits that are* 1*, given an upper bound, k_{max}, on k. There is a Las Vegas* SIMD-EROW PRAM *algorithm that runs in $O(\log k_{\max} + \sqrt{\log n})$ time with $O(n)$ work w.h.p.*

Proof. We describe the algorithm for $n/(\log k_{\max} + \sqrt{\log n})$ processors. The input bits are partitioned among the processors such that each processor is assigned $\log k_{max} + \sqrt{\log n}$ bits. If $k_{max} = \Omega(n^{\epsilon})$ for some constant $0 < \epsilon \leq 1$, apply the EREW parallel prefix algorithm, as mentioned in section 6.1, to obtain the stated bounds. Otherwise, let A be an array of size $m = k_{max} \cdot 2^{\sqrt{\log n}}$, initialized to all zeros (note that $m = O(n)$). Each processor selects a leader from among its input bits that are 1, if any. Then each processor with a leader writes to a cell of A selected uniformly at random. Finally, m of the processors participate to select a nonzero index from among those written to A. The first and third steps take $O(\log k_{max} + \sqrt{\log n})$ time. In the second step, the expected contention to a cell i in A is at most $1/2^{\sqrt{\log n}}$. It follows from Observation 6.1 that the maximum contention over all cells of A is $O(\sqrt{\log n})$ w.h.p. □

6.3. A general randomized algorithm. It is shown in [17] that the OR function on n bits requires $\Omega(\log n)$ time on a randomized CREW PRAM. (This lower bound is for randomized algorithms that have zero probability of a concurrent write, and correctly compute the OR with probability bounded away from 1/2.) In contrast to this lower bound, we show in this subsection that a randomized SIMD-CRQW PRAM can compute the OR function on n bits in $O(\log n/\log\log n)$ time and linear work w.h.p.

THEOREM 6.5. *There is a Las Vegas* SIMD-CRQW PRAM *algorithm for the leader election problem (and the* OR *function) that runs in* $O(\log n/\log\log n)$ *time and linear work w.h.p.*

Proof. We first show the time bound using $n\log\log n$ processors. We describe the algorithm for the OR function, which can be trivially modified to solve the leader election problem. Since the number, k, of contending 1-bits is unknown, we will search for the true value of k. We take larger and larger samples until we either find a sample that contains at least one input bit that is 1, or learn that all input bits are 0. We must ensure that w.h.p. there will be at least one writer (with a 1) prior to the iteration in which there are too many writers (i.e., the iteration where the contention would *not* be $O(\log n/\log\log n)$). The new algorithmic result below is a technique for amplifying probabilities on the SIMD-QRQW model so that this occurs.

1. Let $s = c\log n/\log\log n$, with $c \geq 1$ a constant determined by the analysis. Let A be an array of $s^2\log\log n$ memory cells, A' be an array of $s\log\log n$ memory cells, and A'' be an array of $\log\log n$ memory cells, each initialized to all zeros. The output is to be written in memory cell x. We assign $\log\log n$ processors to each input bit. Each processor reads its input bit. Let $p = s^2/n$.
2. Each processor with input bit 1 is active with probability p. Each such active processor writes its index to some cell i of A chosen uniformly at random, and then reads that cell. If the cell contains its index (i.e., no other processor overwrote it), then it writes its index to cell i' of A', $i' = i \bmod s\log\log n$, and then reads that cell. If the cell contains its index, then it writes its index to cell i'' of A'', $i'' = i' \bmod \log\log n$, and then reads that cell. If the cell contains its index, then it writes a 1 into memory cell x.
3. Each processor reads x. If $x = 0$, repeat steps 2 and 3 with $p = ps$. If $p \geq 1$, repeat one last time with $p = 1$ and then stop.

Note that x is set to 1 only if there is a processor with a 1. Conversely, each processor whose input bit is 1 either writes a 1 into x, writes its index in a cell of A in the iteration that x is set to 1, or stops when $x = 1$; hence the algorithm always outputs the correct answer. There are $O(\log n/\log s)$ iterations. If no processor writes to A in an iteration, then the iteration takes $O(1)$ time. Otherwise there is one last iteration in which writes to A, A', A'', and x occur.

We now analyze the contention of these last four write steps. Let p_j be the probability used at iteration j; i.e., $p_j = s^{j+1}/n$. Let k be the number of (original) input bits that are 1. Since we have a write step, $1 \leq k \leq n$. Let $t \geq 0$ be an integer such that $n/s^t \geq k > n/s^{t+1}$. Consider iteration $t+1$ if it occurs. The probability that no processor writes is at most

$$\begin{aligned}(1-p_{t+1})^{k\log\log n} &< (1/e)^{p_{t+1}k\log\log n}\\ &= (1/e)^{ks^{t+2}\log\log n/n}\\ &< (1/e)^{s\log\log n} < (1/e)^{c\log n}\\ &= (1/e)^{c'\cdot\ln n} = 1/n^{c'}\end{aligned}$$

for some constant c'. Hence, if $k > 0$, there will be no iteration $t+2$ w.h.p.

Let W be the number of active processors at iteration $t+1$, if it occurs. Then

$$E[W] = p_{t+1}k \log\log n = s^{t+2}k \log\log n/n.$$

By the choice of t, $s \geq s^{t+1}k/n > 1$, and hence $s^2 \log\log n \geq E[W] > s \log\log n$. Let X_i be the number of writers to cell i of A in iteration $t+1$. Then

$$E[X_i] = E[W]/s^2 \log\log n \leq 1.$$

By Observation 6.1, and since there are $s^2 \log\log n = o(n)$ cells, the maximum contention for this write is $O(\log n/\log\log n)$ w.h.p.

This bounds as well the contention of any iteration less than $t+1$ in which a write to A occurs (and hence is the last iteration). Since there is at most one winner from each cell of A and exactly s cells of A that map to one cell of A', the maximum contention to a cell of A' is s. Likewise, the maximum contention to a cell of A'' is s and the maximum contention to cell x is $\log\log n$.

It follows that the overall running time is $O(\log n/\log\log n)$ w.h.p.

Finally, in order to make the algorithm work optimal, we should achieve the same time bound using only $n \cdot \log\log n/\log n$ processors. For this we use an initial computation phase in which we reduce the size of the input from n to $n/\log n$. For this we divide the processors into $n/\log n$ groups of $\log\log n$ processors, and assign to each group the simple task of finding the OR of a block of $\log n$ input bits in $O(\log n/\log\log n)$ time. We then apply the algorithm described above to the reduced array of $n/\log n$ bits. This gives us the desired work-optimal randomized algorithm for the OR function on n bits in $O(\log n/\log\log n)$ time w.h.p. □

We note that the only large concurrent read in the previous algorithm is the reading of x in step 3 of the algorithm.

COROLLARY 6.6. *There is an* $\Omega(\log\log n)$ *time separation of a randomized* SIMD-CRQW-PRAM *from a randomized* CREW PRAM.

7. Linear compaction. Consider an array of size n with k nonempty cells, with k known, but the positions of the k nonempty cells not known. The *k-compaction* problem is to move the contents of the nonempty cells to the first k locations of the array. The *linear-compaction* problem is to move the contents of the nonempty cells to an output array of $O(k)$ cells. The best known EREW PRAM algorithms for both problems take $\Theta(\log n)$ time, using parallel prefix sums [42]. Even for the case $k = 2$, there is a randomized $\Omega(\sqrt{\log n})$ expected-time lower bound for the EREW PRAM ([49], following [20]), and a deterministic lower bound of $\Omega(\log\log n)$ for an n-processor CREW PRAM [20].

The simple deterministic SIMD-ERQW PRAM algorithm for leader election discussed in section 6.1 can be trivially extended to the k-compaction problem as follows.

OBSERVATION 7.1. *There is a deterministic* SIMD-ERQW PRAM *algorithm for the k-compaction problem that runs in $O(k^2)$ time with $O(n)$ work.*

Proof. The input is partitioned into subarrays of k^2 cells. Each of the n/k^2 processors reads the cells in its subarray and creates a linked list of the items in its nonempty cells. Since there are only k nonempty cells, no processor can have more than k items in its linked list. The algorithm proceeds in k rounds, in which processors attempt to place each item on their list. At round i, each processor with an unplaced item writes its index to cell i of the array. A designated processor then reads the cell, and if the index found is j, it signals processor j (by writing to a cell designated for

j), which then transfers the contents of its current item to the cell and continues to the next round with its next unplaced item (if any). All other processors continue with the same item as before. The contention in round i is at most $k - i + 1$, so the algorithm runs in $O(k^2)$ time. □

By taking $k = 2$, and recalling the lower bounds mentioned earlier for the EREW and CREW PRAM, we obtain the following two results, which are cited in Table 1 and Table 2 in section 2.

COROLLARY 7.1. *There is an $\Omega(\sqrt{\log n})$ time separation of a (deterministic or randomized)* SIMD-ERQW PRAM *from a (deterministic or randomized)* EREW PRAM.

COROLLARY 7.2. *There is a separation of $\Omega(\log \log n)$ time with n processors of a deterministic* {QR,SIMD-QR,SIMD-CR}QW PRAM *from a deterministic* {QR,SIMD-QR,CR}EW PRAM.

In the remainder of this section, we develop a SIMD-QRQW PRAM algorithm for the linear-compaction problem that runs in $O(\sqrt{\log n})$ time with linear work w.h.p. Within our algorithm, we will employ the following well-known technique for k-compaction, which runs in $O(\log n)$ time using only k processors on an EREW PRAM.

OBSERVATION 7.2. *The k-compaction problem with one processor assigned to each nonempty cell can be solved by an* EREW PRAM *algorithm in $O(\log n)$ time.*

Proof. View the n elements as leaves of a full binary tree. At the ith step we work at level i above the leaves, and inductively, for each node v at this level, we have the solution (in the form of a linked list) for the leaves in the subtrees rooted at the two children of v. To combine these solutions at v we need only to make the last distinguished element in the subtree of the left child of v as the successor of the first distinguished element in the right subtree of v. This can be performed by a constant-time EREW computation. Finally we perform list ranking on the linked list of distinguished elements (using Wyllie's pointer-jumping approach [40]) and transfer the elements to their location in the output array.

Note that the input array need not be initialized: since we have an active processor for each distinguished element, we can detect distinguished elements by a *change* in the value of a memory cell. □

To prove our SIMD-QRQW PRAM result, we start by proving the following lemma, which shows how to achieve the desired time bound. However, the algorithm performs superlinear work when k is large. We then show how to use this lemma to obtain a linear work algorithm with the same time bound.

LEMMA 7.3. *There is a Las Vegas* SIMD-QRQW PRAM *algorithm for linear compaction that runs in $O(\sqrt{\log n})$ time w.h.p. if $\sqrt{\log n}$ processors are assigned to each nonempty cell.*

Proof. Let an *item* denote a nonempty input cell. Let $r = \sqrt{\log n}$, the number of processors assigned to each item. Let A be an auxiliary array of size $m = c_1 r k 2^{c_2\sqrt{\log n}}$ for constants $c_1 \geq 2$, $c_2 \geq 1$ determined by the analysis. View the array A as partitioned into $k/\log n$ subarrays of size $m' = c_1 r 2^{c_2\sqrt{\log n}} \log n$.

1. For each item, select a subarray of A uniformly at random. Each processor assigned to the item selects a cell in that subarray uniformly at random and tries to claim that cell.
2. At this point, between zero and r cells of A have been claimed on behalf of each item. Denote an item *successful* if at least one cell of A has been claimed on its behalf. For each successful item, select one of its cells in A, and mark the rest as unclaimed.
3. In parallel for all subarrays, compact the claimed cells within each subarray

using Observation 7.2. We compact within subarrays here since, for large k, compacting all of A is too slow.

4. View the output array as partitioned into $k/\log n$ subarrays of size $c_1 \log n$. For each j, if there are n_j unclaimed cells in subarray j of the output, then the contents of (up to) n_j claimed cells in subarray j of A are transferred to output subarray j. (In the first pass of the algorithm, $n_j = c_1 \log n$, but in any subsequent pass, n_j may be smaller.) If there are more than n_j claimed cells in a subarray j, then for $i > n_j$, the item associated with the ith claimed cell in subarray j of A is denoted unsuccessful.
5. For each unsuccessful item, each of its r processors returns to step 1.

Since the processors assigned to an item repeat the algorithm until at least one of them has successfully claimed an output cell, this is a Las Vegas algorithm. (Note that processors may complete their participation in the algorithm at different times, not knowing when all processors have terminated.) Let X_j be the number of items selecting subarray j of A in step 1. Then $E[X_j] = k/\lceil k/\log n \rceil \leq \log n$. By Chernoff bounds, for $c_1 \geq 2$ defined above,

$$Pr\{X_j \geq c_1 \log n\} \leq e^{(1-1/c_1-\ln\ c_1)c_1 \log n} < e/c_1^{c_1 \log n} < 1/n^{c_1}.$$

After step 2, there is at most one claimed cell for each item, so w.h.p., there are at most $c_1 \log n$ claimed cells in a subarray. A processor tries to claim a cell in step 1 by first writing its index to the cell, then reading the cell: if it reads its index, it has claimed the cell, and it writes the contents of its input cell to the claimed cell. For each subarray j, let $Y_{j,i}$ be the number of processors selecting cell i of subarray j of A in step 1. Then $E[Y_{j,i}] \leq r \cdot c_1 \log n/m' \leq 1/2^{c_2\sqrt{\log n}}$. It follows from Observation 6.1 that the time for step 1 is $O(\sqrt{\log n})$ w.h.p.

Step 2 can be done in $O(\log r)$ time. Step 3 applies Observation 7.2, and runs in $O(\log m')$ time, which is $O(\sqrt{\log n})$ time. For step 4, for each j, the current value of n_j, as well as the index of the first unclaimed output cell in subarray j, can be broadcast in $O(\log \log n)$ time; the transferring takes constant time.

As for step 5, there are two types of unsuccessful items. As argued above, w.h.p., there are at most $c_1 \log n$ claimed cells in a subarray. It follows that the probability that an item is unsuccessful in step 1 is less than $(r \cdot c_1 \log n/m')^r = (1/2^{c_2\sqrt{\log n}})^{\sqrt{\log n}} < 1/n^{c_2}$. Moreover, it follows that, w.h.p., no cells are marked unsuccessful in step 4. So w.h.p., all cells are successful in the first pass of the algorithm. □

THEOREM 7.4. *There is a Las Vegas* SIMD-QRQW PRAM *algorithm for linear compaction that runs in $O(\sqrt{\log n})$ time with $O(n)$ work w.h.p.*

Proof. We describe the algorithm for $n/\sqrt{\log n}$ processors. Let an *item* denote a nonempty input cell. Note that we make no assumption on the distribution of the items within the input array.

1. View the n input cells as partitioned into subarrays of size $2\log^2 n$. Assign $2\log^{1.5} n$ processors per subarray. In parallel for all subarrays compact the items in each subarray, using parallel prefix.
2. For subarrays with at most $2\log n$ items, we assign $\sqrt{\log n}$ processors per item, and apply Lemma 7.3.
3. For subarrays with more than $2\log n$ items, we view the items as partitioned into blocks of size $\log n$. There are at most $2\log n$ such blocks in a subarray, so we assign $\sqrt{\log n}$ processors per block. Viewing each block as a "superitem," apply Lemma 7.3 to compact the superitems into an array of size $O(k/\log n)$.

Then we transfer the items in each block to the output array of size $O(k)$, in the obvious way.

Each of steps 1–3 takes $O(\sqrt{\log n})$ time w.h.p. □

8. Broadcasting. Given $b \in \{0, 1\}$ in a single memory location, the *broadcasting* problem is to copy b into n fixed memory locations. There is a simple linear work, $O(\log n)$ time EREW PRAM algorithm for this problem. In this section we show that this algorithm is the best possible even for the (randomized) QRQW PRAM by providing an $\Omega(\log n)$ lower bound on the expected running time of any deterministic or randomized QRQW PRAM algorithm for this problem.

Our lower bound exploits the fact that the input domain for the broadcasting problem has only two values. We show that for any problem with an input domain of size 2, a SIMD-QRQW PRAM algorithm is no faster than the best EREW PRAM algorithm for the problem, and even a QRQW PRAM algorithm is at most two times faster than the best EREW PRAM algorithm for the problem. We also show that a randomized algorithm for the problem is at most two times faster than the best deterministic algorithm for the problem. These results, in turn, imply our lower bound for broadcasting and related problems due to a lower bound for broadcasting on the EREW PRAM given by [4].

Our simulation of the SIMD-QRQW PRAM and the QRQW PRAM on the EREW PRAM results in a *nonuniform* algorithm on the EREW PRAM. An algorithm is nonuniform if it consists of different programs for different input sizes, and the program for a given input size i cannot be generated easily simply by specifying the value of i. Most algorithms used in practice are *uniform*, such that a single program works for all input sizes. A nonuniform algorithm is not desirable from a practical point of view, since the time bound for the algorithm is not guaranteed to be achieved on a given input unless we have already generated the program for that input size. However, the lower bound of [4] holds for both uniform and nonuniform algorithms (as is the case with most lower bounds), and hence our simulation result gives the desired lower bound for the SIMD-QRQW PRAM and the QRQW PRAM.

8.1. Constant-size input domain problems. We first deal with the SIMD-QRQW PRAM. We show that any SIMD-QRQW PRAM algorithm for a problem defined on a domain with only two values that runs in time T can be converted into an EREW PRAM algorithm that also runs in time T. The EREW PRAM may be nonuniform and may have a description that is of unbounded size. For an exact definition of the model see [12].

LEMMA 8.1. *Let T be the running time for an algorithm A that solves a problem P with input domain of size 2 on a* SIMD-QRQW PRAM. *Then, there exists an algorithm B that solves P in time T on an* EREW PRAM, *using the same number of processors and the same working space. Algorithm B is nonuniform and its description is of size $O(T)$ memory locations per processor.*

Proof. Assume, without loss of generality, that the input domain is $\{0, 1\}$. The lemma is proved by constructing the EREW PRAM Algorithm B from Algorithm A. Consider the ith step in Algorithm A, and let $\kappa_i(b)$ be the maximum contention in this step on input b. Let $\kappa'_i = \min\{\kappa_i(0), \kappa_i(1)\}$ (recall from Definition 2.1 that $\kappa'_i \geq 1$). Step i will be implemented in Algorithm B in at most κ'_i substeps, as described below. Therefore, the running time of algorithm B is at most $\sum_i \kappa'_i = \sum_i \min\{\kappa_i(0), \kappa_i(1)\} \leq T$. We describe first the construction for the read step.

Let $\Phi_{i,j,b}$ be the set of processors that read from memory cell j in step i on input $b \in \{0, 1\}$. Let $\Phi_{i,j} = \Phi_{i,j,0} \cap \Phi_{i,j,1}$. For processors in each set $\Phi_{i,j,b} \setminus \Phi_{i,j}$, we can

prepare a priori copies of the contents of memory cell j, $c(i,b)$, so that they can do the read operation from their appropriate copies without conflict, as described below.

For processors in each set $\Phi_{i,j}$, we serialize their computation by providing an a priori ranking from $[1..|\Phi_{i,j}|]$ to all the processors in $\Phi_{i,j}$, and scheduling the processors according to their ranks. The program for Algorithm B includes for each processor a sequence $\langle i, M(i,b), r(i,b), \phi_i, c(i,b)\rangle$, $i = 1, \ldots, T$, $b \in \{0,1\}$, where $M(i,b)$ is the memory cell from which the processor reads in step i on input b; $r(i,b)$ is the rank of the processor at step i if the processor is in $\Phi_{i,M(i,b)}$, and is null otherwise; $c(i,b)$ is the contents at step i of memory cell $M(i,b)$ if the processor is in $\Phi_{i,M(i,b),b} \setminus \Phi_{i,M(i,b)}$, and is null otherwise; and $\phi_i = \max_j |\Phi_{i,j}|$. (Note that the processor does not need to know the value of b. If, however, $M(i,0) \neq M(i,1)$ or $r(i,0) \neq r(i,1)$ then it implicitly knows the value of b at this stage; this knowledge can be made explicit by replacing the quintuple above by the sextuple $\langle i, M(i,b), r(i,b), \phi_i, c(i,b), b'\rangle$ where $b' \in \{0,1,*\}$.) This sequence can be specified in $O(T)$ memory locations, and is nonuniform. In step i, each processor whose $r(i,b)$ is not null will execute its read operation from memory location $M(i,b)$ in substep $r(i,b)$. Each processor whose $r(i,b)$ is null will read $c(i,b)$. After a total of ϕ_i substeps, all processors proceed to step $i+1$.

It remains to show how to handle the write steps. Consider a memory location j in step i, and let $\Phi_{i,j}$, $\Phi_{i,j,0}$, and $\Phi_{i,j,1}$ be defined as for the read step. On input b, it is sufficient to select a priori one processor from $\Phi_{i,j,b}$ that will do the write step to location j. If $\Phi_{i,j}$ is not empty then one of the processors in $\Phi_{i,j}$ will be arbitrarily selected. If $\Phi_{i,j}$ is empty, one of the processors in $\Phi_{i,j,b}$ will be arbitrarily selected, unless it is empty. The write operation will be executed by the selected processor at substep ϕ_i. Thus, all the read operations will be completed before the write operation is executed; moreover, there is no additional time overhead due to the execution of the write operations.

With this scheme, the ith step of Algorithm A is executed in $\phi_i \leq \kappa'_i$ steps by Algorithm B, thus giving the desired result. □

We now strengthen the above result for the SIMD-QRQW PRAM to work for the QRQW PRAM with only a constant factor increase in the running time of the simulating EREW PRAM algorithm.

LEMMA 8.2. *Let T be the running time for an algorithm A that solves a problem P with input domain of size 2 on a* QRQW PRAM. *Then, there exists an algorithm B that solves P in time $O(T)$ on an* EREW PRAM, *using the same number of processors and the same working space. Algorithm B is nonuniform and its description is of size $O(T)$ memory locations per processor.*

Proof. We show how to handle the read steps of Algorithm A; write steps are treated similarly. Consider the ith read step in Algorithm A on input b. Let the time cost of this step be t_i. Let R_k be the set of reads for processor p_k, and let M_j be the set of read requests for memory location m_j in step i on input b. Note that t_i is the maximum cardinality of the sets R_k, M_j, over all processor and memory indices k, j.

We construct a bipartite graph $B_{i,b} = (P, M, E_{i,b})$, where P contains a vertex for each processor, M contains a vertex for each memory location, and there is an edge $(p_k, m_j) \in E_{i,b}$ if and only if processor p_k reads memory location m_j in step i on input b.

The maximum degree of any vertex in the graph $B_{i,b}$ is t_i. Since $B_{i,b}$ is bipartite, it has a proper edge coloring with t_i colors (Theorem 6.1 in [9]), i.e., a mapping $c : E_{i,b} \to \{1, 2, \ldots, t_i\}$ such that for any pair of edges e, f incident on the same vertex, $c(e) \neq c(f)$. Thus for a given input b we can serialize the ith step of Algorithm A into

t_i exclusive read substeps by performing the read corresponding to the edges colored l in the lth substep.

Since the input domain is of size 2, b can take on only two values, say 0 and 1, and each processor can be in at most two different states at a given time step, no matter what the input is. In Algorithm B for each step, we run the serialization of the step on input $b = 0$ followed by the serialization of the step on input $b = 1$. If processor p_k is in a state that corresponds only to input $\hat{b} \in \{0, 1\}$ then it performs the read only in the serialization for $b = \hat{b}$. If p_k is in the same state whether $b = 0$ or $b = 1$, then p_k performs the read only in the serialization for $b = 1$. This results in a (nonuniform) EREW PRAM algorithm that performs the same computation as Algorithm A, using the same number of processors and the same working space, and runs in time $O(T)$. The length of the program is the length of the serialization, which is $O(T)$.

There was no attempt to minimize the constants in the above algorithm. Techniques similar to those applied in the proof of Lemma 8.1 can be used here to reduce the constants. □

We now show that randomization cannot help too much when the input domain is small.

LEMMA 8.3. *Let T_d be a lower bound on the time required by a deterministic algorithm to solve a problem P with input taken from a domain of size $|I|$. Then, for any randomized algorithm that solves P, the expected running time T_r on any input is bounded by $T_r \geq T_d/|I|$.*

Proof. Let T_a be the average running time for the uniform-input distribution, minimized over all possible deterministic algorithms, to solve P. Clearly, since the number of possible inputs is $|I|$, $T_a \geq T_d/|I|$. Further, by a classic result of Yao [64], $T_r \geq T_a$. (Yao's result is more general; for a short proof of this claim see [21].) Therefore, $T_r \geq T_a \geq T_d/|I|$. □

8.2. Lower bounds for broadcasting and related problems. Beame, Kik, and Kutylowski [4] showed that computing the broadcasting problem on a nonuniform EREW PRAM with unbounded program size, an unbounded number of processors, and unbounded space requires $\Omega(\log n)$ time. The results of the previous subsection give us the following theorem.

THEOREM 8.4. *Any deterministic or randomized algorithm that computes the broadcasting problem into n memory locations on a* QRQW PRAM *with an unbounded number of processors and unbounded space requires expected time $\Omega(\log n)$.*

Proof. The lower bound for deterministic algorithms follows by the lower bound in [4] and Lemma 8.2 since the size of the input domain for the broadcasting problem is 2. The lower bound for randomized algorithms follows by Lemma 8.3. □

Since a CREW PRAM can broadcast into n memory locations in constant time, Theorem 8.4 immediately implies the following separation results.

COROLLARY 8.5. *There is an $\Omega(\log n)$ time separation of a (deterministic or randomized)* {SIMD-CRQW,CRQW} PRAM *from a (deterministic or randomized)* {SIMD-QRQW,QRQW} PRAM. *The same separation result holds of a* CREW PRAM *from a queue-read, exclusive-write (*QREW*)* PRAM.

The following generalization of the broadcasting problem is used in a lower bound for load balancing given in [29].

THEOREM 8.6. *Any deterministic or randomized algorithm that broadcasts the value of a bit to any subset of k processors in a* QRQW PRAM *requires expected time $\Omega(\log k)$.*

Proof. Let Algorithm A be a QRQW algorithm that succeeds in broadcasting the

value of a bit to some subset of k processors in time t. We use Algorithm A to derive a (nonuniform) QRQW PRAM algorithm for the broadcasting problem into k (fixed) memory locations as follows. We first run Algorithm A to broadcast the value of the bit to some subset of k processors. We then transmit the value of the bit from the ith processor in the subset to the ith output memory location, $1 \leq i \leq k$. This can be performed in one step with time cost 1 since we can precompute from Algorithm A the exact indices of the k processors to which the value of the bit will be transmitted. Thus we can solve the broadcasting problem in $t+1$ steps. It follows from Theorem 8.4 that $t = \Omega(\log k)$. □

9. Conclusions. This paper has proposed a new model for shared-memory machines, the QRQW PRAM model, that takes into account the amount of contention in memory accesses. This model is motivated by the contention characteristics of currently available commercial machines. We have presented several results for this model, including a fast, work-preserving emulation of the QRQW PRAM on hypercube-type, noncombining networks, a work-time framework and some automatic processor allocation schemes for the model, several linear work, sublogarithmic time algorithms for the fundamental problems of leader election on a CRQW PRAM and linear compaction on a QRQW PRAM, and some lower bounds.

In a companion paper [29], we present many new results for the QRQW PRAM. Among the algorithmic results presented are low-contention, fast, work-optimal QRQW algorithms for multiple compaction, load balancing, generating a random permutation, and parallel hashing. These results and the results presented in this paper demonstrate the advantage of the QRQW over the EREW. Together with the penalty in running high-contention CRCW or CREW algorithms on existing machines, this supports the QRQW PRAM as a more appropriate model for high-level algorithm design.

Finally, in a related work [30] we explore the properties of the asynchronous QRQW PRAM.

Acknowledgments. Richard Cole, Albert Greenberg, Maurice Herlihy, Honghua Yang, and the anonymous referees provided useful comments on this work.

REFERENCES

[1] F. Abolhassan, J. Keller, and W. J. Paul, *On the cost-effectiveness of PRAMs*, in Proc. 3rd IEEE Symp. on Parallel and Distributed Processing, Dallas, TX, 1991, pp. 2–9.

[2] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith, *The Tera computer system*, in Proc. 1990 International Conf. on Supercomputing, Amsterdam, The Netherlands, 1990, pp. 1–6.

[3] Y. Aumann and M. O. Rabin, *Clock construction in fully asynchronous parallel systems and PRAM simulation*, in Proc. 33rd IEEE Symp. on Foundations of Computer Science, Pittsburgh, PA, 1992, pp. 147–156.

[4] P. Beame, M. Kik, and M. Kutyłowski, *Information broadcasting by exclusive-write PRAMs*, Parallel Process. Lett., 4 (1994), pp. 159–169.

[5] G. Bell, *Ultracomputers: A teraflop before its time*, Comm. Assoc. Comput. Mach., 35 (1992), pp. 26–47.

[6] G. E. Blelloch, *Scans as primitive parallel operations*, IEEE Trans. Comput., C-38 (1989), pp. 1526–1538.

[7] G. E. Blelloch, *Prefix sums and their applications*, in A Synthesis of Parallel Algorithms, J. H. Reif, ed., Morgan-Kaufmann, San Mateo, CA, 1993, pp. 35–60.

[8] G. E. Blelloch, S. Chatterjee, J. C. Hardwick, J. Sipelstein, and M. Zagha, *Implementation of a portable nested data-parallel language*, in Proc. 4th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming, San Diego, CA, 1993, pp. 102–111.

[9] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, Elsevier, New York, 1976.

[10] R. P. Brent, *The parallel evaluation of general arithmetic expressions*, J. Assoc. Comput. Mach., 21 (1974), pp. 201–208.

[11] R. Cole and O. Zajicek, *The APRAM: Incorporating asynchrony into the PRAM model*, in Proc. 1st ACM Symp. on Parallel Algorithms and Architectures, Santa Fe, NM, 1989, pp. 169–178.

[12] S. A. Cook, C. Dwork, and R. Reischuk, *Upper and lower time bounds for parallel random access machines without simultaneous writes*, SIAM J. Comput., 15 (1986), pp. 87–97.

[13] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, *LogP: Towards a realistic model of parallel computation*, in Proc. 4th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming, San Diego, CA, 1993, pp. 1–12.

[14] R. Cypher, *Valiant's Maximum Algorithm with Sequential Memory Accesses*, Tech. Rep. TR 88-03-08, Department of Computer Science, University of Washington, Seattle, WA, 1988.

[15] W. J. Dally, J. S. Keen, and M. D. Noakes, *The J-Machine architecture and evaluation*, in Proc. 1993 IEEE Compcon Spring, San Francisco, CA, 1993, pp. 183–188.

[16] S. R. Dickey and R. Kenner, *Hardware combining and scalability*, in Proc. 4th ACM Symp. on Parallel Algorithms and Architectures, San Diego, CA, 1992, pp. 296–305.

[17] M. Dietzfelbinger, M. Kutyłowski, and R. Reischuk, *Exact lower time bounds for computing boolean functions on CREW PRAMs*, J. Comput. System Sci., 48 (1994), pp. 231–254.

[18] R. Drefenstedt and D. Schmidt, *On the physical design of butterfly networks for PRAMs*, in Proc. 4th IEEE Symp. on the Frontiers of Massively Parallel Computation, McLean, VA, 1992, pp. 202–209.

[19] C. Dwork, M. Herlihy, and O. Waarts, *Contention in shared memory algorithms*, J. Assoc. Comput. Mach., 44 (1997), pp. 779–805.

[20] F. E. Fich, M. Kowaluk, K. Loryś, M. Kutylowski, and P. Ragde, *Retrieval of scattered information by EREW, CREW, and CRCW PRAMs*, Computational Complexity, 5 (1995), pp. 113–131.

[21] F. Fich, F. Meyer auf der Heide, P. Ragde, and A. Wigderson, *One, two, three,..., infinity: Lower bounds for parallel computation*, in Proc. 17th ACM Symp. on Theory of Computing, Providence, RI, 1985, pp. 48–58.

[22] S. Fortune and J. Wyllie, *Parallelism in random access machines*, in Proc. 10th ACM Symp. on Theory of Computing, San Diego, CA, 1978, pp. 114–118.

[23] S. Frank, H. Burkhardt III, and J. Rothnie, *The KSR1: Bridging the gap between shared memory and MPPs*, in Proc. 1993 IEEE Compcon Spring, San Francisco, CA, 1993, pp. 285–294.

[24] P. B. Gibbons, *A more practical PRAM model*, in Proc. 1st ACM Symp. on Parallel Algorithms and Architectures, Santa Fe, NM, 1989, pp. 158–168. Full version in *The Asynchronous PRAM: A Semi-synchronous Model for Shared Memory MIMD Machines*, Ph.D. thesis, U.C. Berkeley, CA, 1989.

[25] P. B. Gibbons, *Asynchronous PRAM algorithms*, in A Synthesis of Parallel Algorithms, J. H. Reif, ed., Morgan-Kaufmann, San Mateo, CA, 1993, pp. 957–997.

[26] P. B. Gibbons, Y. Matias, and V. Ramachandran, *QRQW: Accounting for Concurrency in PRAMs and Asynchronous PRAMs*, Tech. Rep., AT&T Bell Laboratories, Murray Hill, NJ, 1993.

[27] P. B. Gibbons, Y. Matias, and V. Ramachandran, *Efficient low-contention parallel algorithms*, in Proc. 6th ACM Symp. on Parallel Algorithms and Architectures, Cape May, NJ, 1994, pp. 236–247.

[28] P. B. Gibbons, Y. Matias, and V. Ramachandran, *The QRQW PRAM: Accounting for contention in parallel algorithms*, in Proc. 5th ACM-SIAM Symp. on Discrete Algorithms, Arlington, VA, 1994, pp. 638–648.

[29] P. B. Gibbons, Y. Matias, and V. Ramachandran, *Efficient low-contention parallel algorithms*, J. Comput. System Sci., 53 (1996), pp. 417–442. Special issue devoted to selected papers from the 1994 *ACM Symp. on Parallel Algorithms and Architectures*.

[30] P. B. Gibbons, Y. Matias, and V. Ramachandran, *The queue-read queue-write asynchronous PRAM model*, Theoret. Comput. Sci., 196 (1998), pp. 3–29. Special issue devoted to selected papers from *EURO-PAR'*96.

[31] J. Gil and Y. Matias, *Fast hashing on a PRAM—designing by expectation*, in Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms, San Francisco, CA, 1991, pp. 271–280.

[32] J. Gil and Y. Matias, *An effective load balancing policy for geometric decaying algorithms*, J. Parallel Distributed Comput., 36 (1996), pp. 185–188.

[33] J. Gil, Y. Matias, and U. Vishkin, *Towards a theory of nearly constant time parallel algorithms*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, San Juan, Puerto Rico, 1991, pp. 698–710.

[34] L. A. Goldberg, Y. Matias, and S. Rao, *An optical simulation of shared memory*, in Proc. 6th ACM Symp. on Parallel Algorithms and Architectures, Cape May, NJ, 1994, pp. 257–267.
[35] M. Goodrich, *Using approximation algorithms to design parallel algorithms that may ignore processor allocation*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, San Juan, Puerto Rico, 1991, pp. 711–722.
[36] A. Greenberg, *On the time complexity of broadcast communication schemes*, in Proc. 14th ACM Symp. on Theory of Computing, San Francisco, CA, 1982, pp. 354–364.
[37] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, J. Amer. Statist. Assoc., 58 (1963), pp. 13–30.
[38] IBM Corporation, *IBM Scalable POWERparallel Systems* 9076 *SP*2 *and Enhancements for SP*1, 1994. Hardware announcement.
[39] J. JáJá, *An Introduction to Parallel Algorithms*, Addison–Wesley, Reading, MA, 1992.
[40] R. M. Karp and V. Ramachandran, *Parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, The Netherlands, 1990, pp. 869–941.
[41] R. E. Kessler and J. L. Schwarzmeier, *CRAY T3D: A new dimension for Cray research*, in Proc. 1993 IEEE Compcon Spring, San Francisco, CA, 1993, pp. 176–182.
[42] R. E. Ladner and M. J. Fischer, *Parallel prefix computation*, J. Assoc. Comput. Mach., 27 (1980), pp. 831–838.
[43] F. T. Leighton, *Methods for message routing in parallel machines*, in Proc. 24th ACM Symp. on Theory of Computing, Victoria, British Columbia, Canada, 1992, pp. 77–96. Invited paper.
[44] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C.R. Feynman, M. N Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak, *The network architecture of the Connection Machine CM-5*, in Proc. 4th ACM Symp. on Parallel Algorithms and Architectures, San Diego, CA, 1992, pp. 272–285.
[45] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, *The directory-based cache coherence protocol for the DASH multiprocessor*, in Proc. 17th International Symp. on Computer Architecture, Seattle, WA, 1990, pp. 148–159.
[46] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam, *The Stanford DASH multiprocessor*, IEEE Comput., 25 (1992), pp. 63–79.
[47] P. Liu, W. Aiello, and S. Bhatt, *An atomic model for message-passing*, in Proc. 5th ACM Symp. on Parallel Algorithms and Architectures, Velen, Germany, 1993, pp. 154–163.
[48] P. MacKenzie and V. Ramachandran, *ERCW PRAMs and optical communication*, Theoret. Comput. Sci., 196 (1998), pp. 153–180. Special issue devoted to selected papers from *EURO-PAR'*96.
[49] P. D. MacKenzie, *private communication*, Austin, TX, 1994.
[50] C. Martel, A. Park, and R. Subramonian, *Work-optimal asynchronous algorithms for shared memory parallel computers*, SIAM J. Comput., 21 (1992), pp. 1070–1099.
[51] MasPar Computer Corporation, *MasPar System Overview, document* 9300-0100*, revision A*3, 749 North Mary Avenue, Sunnyvale, CA 94086, Mar. 1991.
[52] Y. Matias, *Highly Parallel Randomized Algorithmics*, Ph.D. thesis, Tel Aviv University, Israel, 1992.
[53] Y. Matias and U. Vishkin, *Converting high probability into nearly-constant time—with applications to parallel hashing*, in Proc. 23rd ACM Symp. on Theory of Computing, New Orleans, LA, 1991, pp. 307–316.
[54] N. Nishimura, *Asynchronous shared memory parallel computation*, in Proc. 2nd ACM Symp. on Parallel Algorithms and Architectures, Crete, Greece, 1990, pp. 76–84.
[55] G. F. Pfister and V. A. Norton, *"Hot spot" contention and combining in multistage interconnection networks*, IEEE Trans. Comput., C-34 (1985), pp. 943–948.
[56] L. Prechelt, *Measurements of MasPar MP*-1216*A Communication Operations*, Tech. Rep., Institut für Programmstrukturen und Datenorganisation, Universität Karlsruhe, Karlsruhe, Germany, 1992.
[57] A. G. Ranade, *Fluent parallel computation*, Ph.D. thesis, Department of Computer Science, Yale University, New Haven, CT, 1989.
[58] J. H. Reif, ed., *A Synthesis of Parallel Algorithms*, Morgan-Kaufmann, San Mateo, CA, 1993.
[59] M. Schmidt-Voigt, *Efficient parallel communication with the nCUBE* 2*S processor*, Parallel Comput., 20 (1994), pp. 509–530.
[60] L. Snyder, *Type architecture, shared memory and the corollary of modest potential*, Annual Review of CS, I (1986), pp. 289–317.

[61] L. G. Valiant, *A bridging model for parallel computation*, Commun. Assoc. Comput. Mach., 33 (1990), pp. 103–111.

[62] L. G. Valiant, *General purpose parallel architectures*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, The Netherlands, 1990, pp. 943–972.

[63] L. G. Valiant, *A Combining Mechanism for Parallel Computers*, Tech. Rep. TR-24-92, Harvard University, Cambridge, MA, 1992.

[64] A. Yao, *Probabilistic computations: Towards a unified measure of complexity*, in Proc. 18th IEEE Symp. on Foundations of Computer Science, Providence, RI, 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 771–781

A CONSTANT-FACTOR APPROXIMATION ALGORITHM FOR THE GEOMETRIC k-MST PROBLEM IN THE PLANE*

JOSEPH S. B. MITCHELL†, AVRIM BLUM‡, PRASAD CHALASANI§, AND SANTOSH VEMPALA¶

Abstract. We show that any rectilinear polygonal subdivision in the plane can be converted into a "guillotine" subdivision whose length is at most twice that of the original subdivision. "Guillotine" subdivisions have a simple recursive structure that allows one to search for "optimal" such subdivisions in polynomial time, using dynamic programming. In particular, a consequence of our main theorem is a very simple proof that the k-MST problem in the plane has a constant-factor polynomial-time approximation algorithm: we obtain a factor of 2 (resp., 3) for the L_1 metric, and a factor of $2\sqrt{2}$ (resp., 3.266) for the L_2 (Euclidean) metric in the case in which Steiner points are allowed (resp., not allowed).

Key words. minimum spanning trees, k-MST, guillotine subdivisions, quota traveling salesman problem, prize-collecting salesman problem, bank robber (orienteering) problem, network optimization, computational geometry, dynamic programming, approximation algorithms polynomial

AMS subject classifications. 68Q25, 68R10, 68U05

PII. S0097539796303299

1. Introduction.

We introduce a new technique that can be used to obtain simple approximation algorithms for geometric network design problems. The method is based on the concept of a "guillotine subdivision." Roughly speaking, a "guillotine subdivision" is a rectilinear polygonal subdivision with the property that there exists a horizontal or vertical line (a "cut") whose intersection with the edge set is *connected* and the subdivisions on either side of the line are also guillotines. The connectedness property allows one to apply dynamic programming to optimize over guillotine subdivisions, as there is a succinct specification of how the subdivision interacts with the "cuts" that make up the boundary of a rectangle that specifies a "subproblem" of the dynamic program.

Key to our method is a theorem showing that any rectilinear polygonal subdivision can be converted into a guillotine subdivision by adding a set of edges whose total length is small (at most that of the original subdivision).

To illustrate the power of the method, we show how it can be used to give a very simple constant-factor approximation algorithm for the geometric k-MST problem, obtaining a substantially better factor than previously known. We also apply it to some related problems (the "quota TSP," "prize-collecting salesman," and "bank robber (orienteering)" problems).

*Received by the editors May 10, 1996; accepted for publication (in revised form) April 18, 1997; published electronically September 14, 1998.

http://www.siam.org/journals/sicomp/28-3/30329.html

†Department of Applied Mathematics and Statistics, State University of New York, Stony Brook, NY 11794-3600 (jsbm@ams.sunysb.edu). This research was supported in part by Hughes Research Laboratories and NSF grants CCR-9204585 and CCR-9504192.

‡School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891 (avrim@cs.cmu.edu). This research was supported in part by NSF National Young Investigator grant CCR-9357793 and a Sloan Foundation Research Fellowship.

§Los Alamos National Laboratory, Los Alamos, NM 87544 (chal@lanl.gov).

¶School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891 (svempala@cs.cmu.edu). This research was supported in part by NSF National Young Investigator grant CCR-9357793.

A motivating application. A special case of the "quota TSP" problem is the following: you are a salesman who must sell k items. You can sell one item in each of n cities ($n \geq k$). You want to find a shortest tour that visits at least k cities, so that you can sell your quota of k items. A solution (exact or approximate) to the k-MST problem on the n cities immediately gives an approximation to the desired optimal tour (simply by doubling the tree, in the usual manner).

Related work. In the minimum-weight k-tree, or k-MST, problem, we are given a graph on n vertices with nonnegative distances on the edges and an integer $k \leq n$, and our goal is to find a tree of least total weight that spans some subset of k vertices. The k-MST problem was introduced independently by Fischetti et al. [11], Zelikovsky and Lozevanu [22], and Ravi et al. [21]. In those papers, the problem is shown to be NP-complete, and Ravi et al. give an approximation algorithm with factor $O(\sqrt{k})$. Algorithms with improved approximation factors have since been discovered: Awerbuch et al. [2] obtain factor $O(\log^2 k)$, and Rajagopalan and Vazirani [20] obtain $O(\log k)$. In the time since our work first appeared (in [6] and [18]), there have been further improvements in the approximation factors: Blum, Ravi, and Vempala [7] obtain a factor of 17, and, most recently, Garg [12] obtains a factor of 3. Cheung and Kumar [8] have also considered the problem, which they call the "quorum-cast" problem and which arises in communication networks. Note that if $k = n$, then the k-MST problem is simply the usual minimum spanning tree problem, which has efficient (polynomial-time) exact solutions; thus, the complexity of the k-MST problem arises from the fact that we must find *which* k vertices to connect with a minimum spanning tree.

In the *geometric* k-MST problem, the underlying graph is the complete graph induced by a set of points in the plane, with distances between pairs of points determined by the underlying metric space (typically, this will be Euclidean (L_2) or L_1). Specifically, we are given a set P of n points in the plane, and an integer $k \leq n$, and we are to find a subset of k points of P that has the shortest minimum spanning tree. The problem is NP-hard. Ravi et al. [21] give an approximation algorithm with ratio $O(k^{1/4})$, which was quickly improved to a factor of $O(\log k)$ by Garg and Hochbaum [13] and Mata and Mitchell [17]. Eppstein [10] has improved the approximation ratio to $O(\log k / \log \log n)$ and has given general techniques to improve the running times (as a function of n) of existing algorithms; further, he shows that the exact k-MST problem can be solved in time $2^{O(k \log k)} n + O(n \log n)$, which is simply $O(n \log n)$ for fixed k.

Note that, up to small constant factors in the approximation ratio, the k-MST problem is equivalent to the problem of finding a shortest path or shortest tour that visits k points (the k-TSP problem) or a shortest Steiner tree connecting k points.

Our contribution. Our result is a simple proof of an $O(1)$ approximation ratio for the geometric k-MST problem. We obtain a factor of 2 (resp., 3) (for the L_1 metric) or $2\sqrt{2}$ (resp., 3.266) (for the Euclidean metric) in the case that Steiner points are (resp., are not) allowed. Further, we expect that our guillotine subdivision results may yield similar improvements and simplifications to approximation algorithms for other geometric network design problems. In section 6, we mention a few applications of our method to some problems that are related to the k-MST.

This paper represents the contributions from two manuscripts: the work of Blum, Chalasani, and Vempala [6], based on the notion of a "division tree," and the improvement and simplification to it given by Mitchell [18], based on the notion of "guillotine subdivisions." We will briefly describe the original "division tree" method of Blum et al. but will only give details of the simpler method of Mitchell [18].

2. Division trees. In this section we define *division trees*, and describe a simple dynamic programming algorithm based on this notion that achieves a constant-factor approximation to the k-MST problem. The proof of the approximation guarantee appears in [6]. We do not present the proof here because in the next section we will describe a more powerful algorithm that achieves a better constant factor, and for which the proof is significantly simpler.

To define a division tree, we assume for convenience that no two points lie on the same horizontal or vertical line. In this case, we say that a spanning tree T for a set of points is a *division tree* if T satisfies the following recursive property.

> There exists some point r (the "root") such that either the vertical or the horizontal line through r splits T into two division trees. More precisely, we require both that (A) this line does not intersect any edges of T, and (B) the trees T_1 and T_2 induced by the points on either side of the line *including* r should be division trees. For the base case, if there are just two points, then the single edge is a division tree.

Given any set of n points, the following simple dynamic programming algorithm finds the subset of k points having a division tree of minimum weight.

The algorithm is most easily viewed in a recursive "memoizing" form. It returns both the desired set of k points and the *cost* of the associated division tree. The algorithm takes as input a set of points P, an integer k, and also up to four additional constraints. For each of the four sides of the bounding box of P the algorithm may be told that the point on that bounding side is "required" and must be in any set of k points the algorithm produces. At the outer loop there are no required points. Given these inputs, the algorithm considers each vertical and horizontal line that passes through some point in P and does not coincide with an edge of the bounding box. For a given such line—let p be the point in P that the line passes through—the algorithm constructs the bounding boxes B_1 and B_2 of the points on the two sides, considering p to be on both sides. It then calls itself recursively $k-1$ times for each of the two boxes B_i: in each call passing down the set of points in B_i, a new integer $k' \in [2, k]$, and the set of required points it was originally given (only considering those that lie in the box B_i) *including* the new point p. Once the algorithm receives its $k-1$ answers from each side, it simply compares to find the pair $\langle k', k-k'+1\rangle$ whose costs sum to the least amount (the reason for the "+1" is that point p lies on both sides). In the base case $k=2$, the algorithm just returns the cost of the single edge.

Because there are at most n^4 different bounding boxes, k different possibilities for the desired number of points, and 16 different settings for the "required points," the memoized procedure (or, equivalently, dynamic program) will run in polynomial time. Also, it is not too hard to see that this algorithm finds the set of k points with the lightest division tree. What is shown in [6] is that for any set of points, the division tree of minimum weight is only a constant factor more costly than the minimum spanning tree.

3. Guillotine subdivisions. We now turn to guillotine subdivisions, and our main theorem that any rectilinear subdivision of the plane can be approximated by one that is guillotine.

Consider a rectilinear polygonal subdivision S that is induced by a finite set of noncrossing horizontal and vertical (closed) line segments in the plane, whose union,

E, comprises the edges of S. We can assume (without loss of generality) that S is restricted to the unit square B (i.e., $E \subset \text{int}(B)$). Then each facet (2-face) is a bounded rectilinear polygon, possibly with holes. The *length* of S is the sum of the lengths of the edges of S.

A closed, axis-aligned rectangle W is a *window* if $W \subseteq B$. In the following definitions, we fix attention on a given window W.

A line ℓ is a *cut* for E with respect to W if $\ell \cap \text{int}(W) \neq \emptyset$. The intersection $\ell \cap (E \cap \text{int}(W))$ of a cut ℓ with $E \cap \text{int}(W)$, the *restriction* of E to the window W, consists of a discrete (possibly empty) set of subsegments of ℓ. (Some of these "segments" may be points, where ℓ crosses an edge.) The endpoints of these subsegments are called the *endpoints along* ℓ (with respect to W). (The two points where ℓ crosses the boundary of W are not considered to be endpoints along ℓ.) Let ξ be the number of endpoints along ℓ, and let the points be denoted by $p_1, \ldots, p_\xi$ in order along ℓ.

We define the *span* $\sigma(\ell)$ of ℓ (with respect to W) as follows. If $\xi = 0$, then $\sigma(\ell) = \emptyset$; otherwise, $\sigma(\ell)$ is defined to be the (possibly zero-length) line segment $p_1 p_\xi$.

A line ℓ is a *perfect cut with respect to* W if $\sigma(\ell) \subseteq E$ (which implies that $\xi = 2$, or $\xi = 1$ in case $\sigma(\ell)$ is a single point).

Finally, we say that S is a *guillotine subdivision with respect to window* W if either (1) $E \cap \text{int}(W) = \emptyset$; or (2) there exists a perfect cut ℓ, with respect to W, such that S is guillotine with respect to windows $W \cap H^+$ *and* $W \cap H^-$, where H^+, H^- are the closed halfplanes induced by ℓ. We say that S is a *guillotine subdivision* if S is guillotine with respect to the unit square B.

See Figure 3.1 for an example of a guillotine subdivision, where we illustrate the entire tree of perfect cuts. (Each perfect cut is indicated with a small arrow.)

Note that, in contrast with guillotine *rectangular* subdivisions (see [9, 17]), the guillotine subdivisions we study here are not restricted to have rectangular faces; rather, the faces of a guillotine subdivision are rectilinear polygons. In fact, it is precisely this distinction that permits us to get constant-factor approximations, while the previous method of [17] obtained logarithmic factors. For example, in order to transform a "staircase" (rectilinear) polygon into a guillotine rectangular subdivision, we must increase its total edge length by a factor of $\Omega(\log n)$; in contrast, a staircase polygon is already a guillotine subdivision according to our definition.

4. The main theorem. We now show that any rectilinear subdivision can be converted into a guillotine subdivision without increasing its length by much (at most doubling it). Our proof is inspired by the proof in [9] that any subdivision of a box (in $\Re^2$) into *rectangles* can be converted into a "guillotine" rectangular subdivision of at most twice the length by adding a new set of edges whose total length is small (charged off to the original edges of the subdivision).

THEOREM 4.1. *Let S be a rectilinear subdivision of length L with edge set E. Then there exists a guillotine subdivision S_G of length at most $2L$ whose edge set E_G contains E.*

Proof. We will convert S into a guillotine subdivision S_G by adding to E a new set of horizontal or vertical edges whose total length is at most L. The construction is recursive; at each stage, we show that there exists a cut ℓ with respect to the current window W (which initially is the box B) such that we can afford to add the span $\sigma(\ell)$ to E, while appropriately charging off the length of $\sigma(\ell)$. (Once we add $\sigma(\ell)$ to E, ℓ becomes a perfect cut with respect to W.)

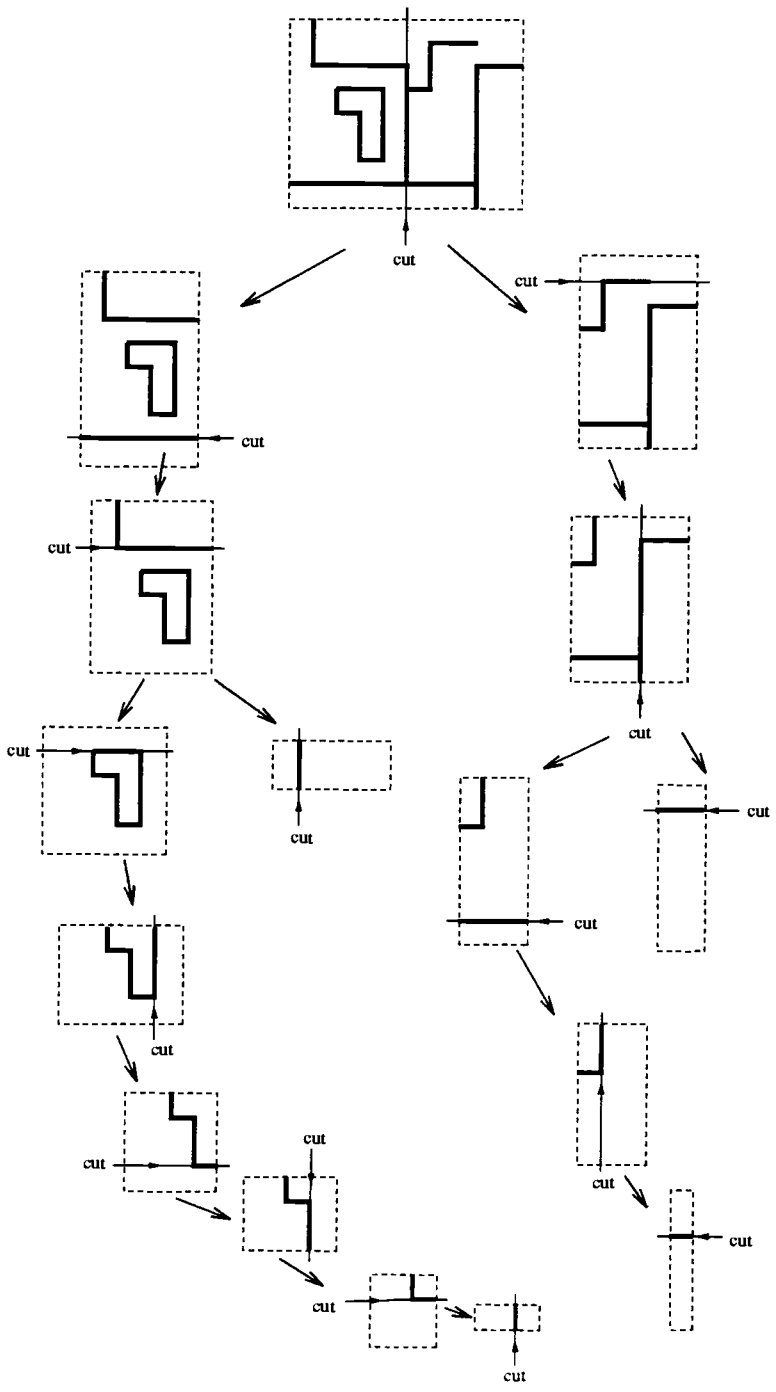


FIG. 3.1. *An example of a guillotine subdivision. Each perfect cut is indicated with a small arrow; bounding boxes (windows) are indicated with dashed rectangles.*

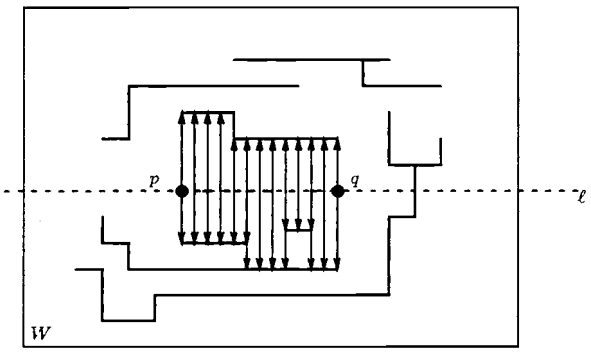


FIG. 4.1. *Subsegment $pq \subset \ell$ is dark; its length is charged to the subsegments of E that lie above or below.*

In fact, we will restrict ourselves to a special discrete set of horizontal or vertical cuts, namely, those determined by the x- or y-coordinates of original vertices V of the subdivision, or by the midpoints between consecutive x- or y-coordinates of V.

First, note that if a perfect cut (with respect to W) exists, then we can simply use it and proceed recursively on each side of the cut. Thus, we assume that no perfect cut exists with respect to a given window, W.

We say that a point p on a cut ℓ is *dark with respect to ℓ and W* if, along $\ell^\perp \cap \text{int}(W)$, there is at least one endpoint (strictly) on each side of p, where $\ell^\perp$ is the line through p that is perpendicular to ℓ.[1] We say that a subsegment of ℓ is *dark* (with respect to W) if all points of the segment are dark with respect to ℓ and W.

The important property of dark points along ℓ is the following: assume, without loss of generality, that ℓ is horizontal. Then if all points on subsegment pq of ℓ are dark, then we can charge the length of pq off to the bottoms of the subsegments $E^+ \subseteq E$ of edges that lie above pq and are vertically visible to pq, and the tops of the subsegments $E^- \subseteq E$ of edges that lie below pq and are vertically visible to pq (since we know that there is at least one edge "blocking" each point of pq from the top or bottom of W). We charge pq's length half to E^+ (charging E^+ *from below*, with $\frac{1}{2}$ units of charge) and half to E^- (charging E^- *from above*, with $\frac{1}{2}$ units of charge). In Figure 4.1 we illustrate how a dark subsegment pq has its length charged off.

We call a cut ℓ *favorable* if the dark portion of ℓ is at least as long as the span $\sigma(\ell)$. Lemma 4.2 below shows that a favorable cut always exists (even one in the special discrete set). For a favorable cut ℓ, we add its span to the edge set (charging off its length, as above) and recurse on each side of the cut in the two new windows. After a portion of E has been charged on one side, due to a cut ℓ, it will be vertically visible to the boundary of the windows on either side of ℓ and, hence, will be vertically visible to the boundary of any future windows, found deeper in the recursion, that contain the portion. Thus, *no portion of E will ever be charged more than once from each side* (top and bottom), so no portion of E will ever pay more than its total length in charge ($\frac{1}{2}$ from each side). Also, the new edges added (the spans $\sigma(\ell)$) are never themselves charged, since they lie on window boundaries and cannot therefore serve to make a portion of some future cut dark.

[1] We can think of the edges E as being "walls" that block light; then p on a line ℓ is dark if p is not illuminated when light is shone in from the boundary of W along the direction of $\ell^\perp$.

Note too that in order for a cut ℓ to be favorable, but not perfect, there must be at least one segment of E parallel to ℓ in each of the two open halfplanes induced by ℓ; thus, the recursion must terminate in a finite number of steps.

Since the total length of all spans for all favorable cuts is at most L, and the total length of all spans for all perfect cuts is at most L, we are done. □

We now prove our key lemma, which guarantees the existence of a favorable cut when there is no perfect cut. The proof of the lemma uses a particularly simple argument based on elementary calculus (reversing the order of integration).

LEMMA 4.2. *For any subdivision S, and any window W, there must be a favorable cut.*

Proof. We show that there must be a favorable cut that is either horizontal or vertical.

Let $f(x)$ denote the length of the span (with respect to W) of the vertical line through x. Then $\int_0^1 f(x)dx$ is simply the area A_x of the (x-monotone) region R_x of points of B that are dark with respect to horizontal cuts. Similarly, define $g(y)$ to be the length of the span of the horizontal line through y, and let $A_y = \int_0^1 g(y)dy$.

Assume without loss of generality that $A_x \geq A_y$. We claim that there exists a horizontal favorable cut; i.e., we claim that there exists a horizontal cut ℓ such that the length of its dark portion is at least as large as the length of its span $\sigma(\ell)$. To see this, note that A_x can be computed by switching the order of integration, "slicing" the region R_x horizontally, rather than vertically; i.e., $A_x = \int_0^1 h(y)dy$, where $h(y)$ is the length of the intersection of R_x with a horizontal line through y; i.e., $h(y)$ is the length of the dark portion of the horizontal line through y. Thus, since $A_x \geq A_y$, we get that $\int_0^1 h(y)dy \geq \int_0^1 g(y)dy \geq 0$. Thus, it cannot be that for all values of $y \in [0,1]$, $h(y) < g(y)$, so there exists a $y = y^*$ for which $h(y^*) \geq g(y^*)$. The horizontal line through this y^* is a cut satisfying the claim of the lemma. (If, instead, we had $A_x \leq A_y$, then we would get a *vertical* cut satisfying the claim.)

Finally, we note that, in the rectilinear case, f, g, and h are piecewise constant, with discontinuities corresponding to vertices V of S. Then we can always select y^* to be at a discontinuity or at a midpoint between two discontinuities. □

Remark. It is interesting to consider whether or not the factor of 2 in the theorem can be improved. We have not been able to find an example of a subdivision that cannot be made into a guillotine subdivision by the addition of edges whose total length is less than *half* that of the original subdivision. We conjecture that the factor 2 can be improved to a factor of 3/2.

5. An application to the k-MST. One application of our theorem is that it yields an algorithm along with a simple proof that it achieves a (small) constant-factor approximation of the geometric k-MST problem.

COROLLARY 5.1. *The geometric k-MST problem has a polynomial-time approximation algorithm with approximation factor 2 (allowing Steiner points) or 3 (not allowing Steiner points) in the L_1 metric. In the Euclidean metric, the approximation factors become $2\sqrt{2}$ (Steiner) or $4\sqrt{6}/3 = 3.266$ (non-Steiner).*

Proof. Let P be a set of n points in the plane. Assume that no two points of P lie on a common vertical or horizontal line (otherwise, we can perturb the points or slightly rotate the coordinate system). Let T_R be a minimum-length rectilinear Steiner k-MST for P, and let L_R^* denote its total (Euclidean) length. We can assume that T_R lies on the grid of horizontal or vertical lines through P, since T_R can easily be modified to lie on the grid, without increasing its overall length. Clearly, the length

L_R^* of T_R is the optimal length for the L_1 Steiner k-MST and is at most $\sqrt{2}$ times the length, L^*, of an optimal Euclidean Steiner k-MST.

By Theorem 4.1, there exists a guillotine subdivision S_G, with edge set E_G, of length at most $2L_R^*$ (at most $2\sqrt{2}L^*$), such that $T_R \subseteq E_G$. Thus, there exists a perfect cut ℓ (inducing closed halfplanes $\bar{H}^+$ and $\bar{H}^-$) for E_G such that $\ell \cap E_G$ is connected (a segment). When we select a favorable cut, whose existence is shown in Lemma 4.2, we can always select the cut either to pass through a point of P or to pass through, say, the midpoint of some x- or y-interval determined by consecutive coordinates of points of P; thus, the cuts will have coordinates from a *discrete* set determined by P. Proceeding recursively on each side of ℓ, we can build a tree $\mathcal{T}$ of perfect cuts.

The bounding boxes corresponding to each node of this tree have the special property that the intersection of E_G with each side of a box is *connected*; thus, we can partition the problem into subproblems, each having *constant*-size (discrete) specification, and can easily apply dynamic programming to search for an optimal guillotine subdivision that visits k of the points P and has a connected set of edges E. This yields a (rectilinear) Steiner tree connecting k points of total Euclidean length at most $2L_R^*$. If our goal is a Steiner k-MST approximation, then we are done. Otherwise, at the end we compute and output a minimum spanning tree for this subset of k points. The worst-case length of the final tree is obtained by multiplying by the Steiner ratio ($3/2$ for L_1, $2\sqrt{3}/3$ for L_2). Thus, we get an approximation factor of $2 \cdot (3/2) = 3$ (for L_1 metric) or of $2\sqrt{2} \cdot (2\sqrt{3}/3) = 4\sqrt{6}/3 = 3.266$ (for L_2 metric). We give details of the dynamic programming algorithm below. □

A dynamic programming algorithm. Let $x_1 < x_2 < \cdots < x_{2n-1}$ (resp., $y_1 < y_2 < \cdots < y_{2n-1}$) denote the sorted x (resp., y) coordinates of the n points P, as well as the $n-1$ midpoints of the intervals determined by these coordinates. We now give a dynamic programming algorithm, which is based on solving the following subproblems.

Input.

1. An integer $k' \leq k$;
2. a rectangle $R(i, i', j, j')$, defined by x_i, $x_{i'}$ $(x_{i'} > x_i)$, y_j, and $y_{j'}$ $(y_{j'} > y_j)$;
3. four "boundary segments," σ_l, σ_r, σ_b, and σ_t, which are (possibly empty or zero-length) subsegments of the four sides (left, right, bottom, top) of $R(i, i', j, j')$, each of which has endpoints determined by coordinates x_i, y_j; and,
4. a partition $\mathcal{P}$ of the set $\{\sigma_l, \sigma_r, \sigma_b, \sigma_t\}$ of boundary segments.

Objective. Compute a minimum-length guillotine subdivision S_G^* determined by some set E_G^* of horizontal or vertical line segments not lying on the boundary of rectangle $R(i, i', j, j')$, such that E_G^* covers at least k' points of P (interior to $R(i, i', j, j')$) and the edges E_G^* connect the boundary segments, according to the partition $\mathcal{P}$.

Note that there are $O(k \cdot n^4 \cdot (n^2)^4) = O(kn^{12})$ possible inputs (subproblems). Note too that an optimal solution S_G^* will necessarily be a forest, since any cycle that is formed can be broken without violating the connectivity requirements (given by $\mathcal{P}$).

The optimal value V of the above problem is 0 if $k' = 0$ and all connections among boundary segments specified by $\mathcal{P}$ are vacuously satisfied (i.e., the boundary segments that need to be connected are already connected at their endpoints (corners of the boundary box)). Otherwise, we can compute the value V recursively by adding

the values of the two subproblems obtained by splitting the problem and optimizing over all choices associated with a split:

1. $O(k)$ choices of how to partition k' among the two new subproblems;
2. $O(n)$ choices of a cut by a horizontal or vertical line (determined by some x_i or y_j);
3. $O(n^2)$ choices of new boundary segment σ on the cut; and
4. $O(1)$ choices of partitions for the two sets of boundary segments on the two new subrectangles determined by the cut subject to these partitions being consistent with the partition $\mathcal{P}$.

The polynomial time bound for solving the above recursions has a rather high exponent — $O(n^{15}k^2)$. One approach to improving this is to apply the following lemma of Eppstein [10], which is obtained by doing a simple nearest-neighbor clustering.

LEMMA 5.2 (see [10]). *If we have a time bound $T(n,k)$ for an exact or approximate geometric k-MST problem, we can solve the same problem in time $O(n\log n + nT(k^2,k)/k^2)$.*

A direct application of the above yields a time bound of $O(n\log n + nk^{30})$, which is an improvement when k is small compared with n.

Remark. Awerbuch et al. [2] discuss both "rooted" and "unrooted" versions of the k-MST problem. In the "rooted" version of the problem, we are given a specified point r, and we must use r as one of the k points in the MST. They prove that their $O(\log^2 k)$-approximation method applies to both problems, but the approximation ratio may increase by 1 for the rooted case. It is easy to see that our dynamic programming algorithm allows us also to obtain a rooted solution that is within the *same* factor of optimal as in the unrooted case, simply by specifying the appropriate constraint in the input to the dynamic program.

6. Other applications. Our methods also apply to three other problems that are related to the k-MST, and for which approximation algorithms for the graph versions have been given by Awerbuch et al. [2].

1. In the *quota-driven salesman* problem, each point of P has an associated integral value w_i, and a salesman has a given integer quota R. The objective is to find a shortest possible route (or tour) such that the sum of the values for the cities visited is at least R. It is immediate that our k-MST approximation gives an approximation for this problem too: simply replace each point of P by w_i copies of itself, at the same location in the plane. Now, simply compute an approximate solution to the k-MST, with $k = R$, and then double the tree to obtain a path or a tour.
   This algorithm runs in time polynomial in n and R. One open problem is to extend the algorithm to run in time polynomial in n and $\log R$, which holds for the algorithm of Awerbuch et al. [2].
2. In the *prize collecting salesman* problem (or, PCTSP), as studied by Balas [3] (see also [4]), the setup is the same as in the quota-driven salesman problem, except that, in addition to "values" w_i, there are nonnegative penalties associated with each point of P, and the objective function is now to minimize the sum of the distances traveled *plus* the sum of the penalties on the points *not* visited subject to satisfying the quota R. (Thus, if all penalties are 0, we simply get the quota-driven salesman problem.) As mentioned in [2], an approximation of PCTSP follows immediately from concatenating a tour obtained for the quota-driven salesman, with the 2-approximation tour given

by the algorithm of Goemans and Williamson [15] (which considers the effect of penalties but does not use the quota constraint).

3. In the *bank robber (orienteering)* problem, we are faced with essentially the dual of the quota-driven salesman problem: given a gas tank that allows one to travel a distance d, maximize the total value R of all points visited. As in [2], we can obtain an approximation for this problem, based on "guessing" the value of R, running the approximation for the quota-driven salesman for quota R, breaking the path into subpaths of length $d/2$, and then picking the subpath of highest value.
   This reduction holds only for the *unrooted* version of this problem. For the rooted version (one has a fixed root and a given distance d, and the goal is to visit as many points as possible and return to the root without running out of gas) it is unclear how to obtain any nontrivial approximation.
4. In the *minimum latency problem* (MLP), also known as the *deliveryman problem* and the *traveling repairman problem*, we are given a set of points and must find a tour that minimizes the sum of the "latencies" of all points, where the *latency* of a point p is the length of the tour from the starting point to the point p. (Thus, the latency of a point measures how long a job at that point must wait before being served by the repairman/deliveryman that is traveling along the tour.) Blum et al. [5] have given an approximation algorithm with a constant-factor bound of 128; this bound has recently been improved by Goemans and Kleinberg [14] to 29. By a direct application of Theorem 2 of [5], which states that a c-approximation for the k-MST implies an $8c$-approximation for the MLP, we see that our results immediately imply a 24-approximation algorithm for the L_1 metric MLP and a 26.13-approximation algorithm for the L_2 metric MLP for points in the plane.

In a recent application of our guillotine subdivision results, Mata and Mitchell [16] have obtained a constant-factor approximation algorithm for the following *red-blue separation* problem: given n points in the plane, each colored red or blue, find a shortest simple polygon separating red from blue. This problem is known to be NP-hard and previously had an $O(\log n)$ approximation algorithm [17].

7. Conclusion. In conclusion, we mention some of the exciting developments that have happened in the time since this paper was written. In the spring of 1996, Arora [1] announced a remarkable result—he had found a polynomial-time approximation scheme (PTAS) for the Euclidean traveling salesperson problem (TSP) as well as other geometric optimization problems such as the Steiner tree problem, the k-MST problem, etc.[2] Then, some weeks later, Mitchell [19] discovered that, in fact, a very minor variation of the method and proof given in his earlier work [18], and contained in this paper, also gives a particularly simple PTAS for geometric instances of the TSP, the k-MST, the Steiner tree problem, etc. All that must be modified is the definition of "span," from "span" to "m-span" (which links the mth endpoint with the mth-from-the-last endpoint along a cut ℓ), and, in the proof of Theorem 4.1, the notion of "darkness," from "darkness" to "m-darkness" (p is *m-dark* with respect to ℓ and W if, along $\ell^{\perp} \cap \text{int}(W)$, there are at least m endpoints (strictly) on each side of p). Here, we gave the case of $m = 1$. By allowing m to be any positive integer, the exact same proof goes through, resulting in the following extension to

[2] This means that for any fixed $\epsilon > 0$ there exists a polynomial-time algorithm that gets within a factor $(1 + \epsilon)$ of optimal.

Theorem 4.1: *for any rectilinear subdivision S with edge set E of length L and for any positive integer m, there exists an m-guillotine rectilinear subdivision S_G of length at most $(1+\frac{1}{m})L$ whose edge set E_G contains E.* (The same proof also applies to general (nonrectilinear) subdivisions, resulting in an m-guillotine subdivision of length at most $(1+\frac{\sqrt{2}}{m})L$.) See [19] for further details of the application of this extension.

Acknowledgments. We thank D. Eppstein, M. Held, S. Khuller, and C. Mata for comments and suggestions that improved this paper.

REFERENCES

[1] S. Arora, *Polynomial time approximation schemes for Euclidean TSP and other geometric problems*, in Proc. 37th Ann. IEEE Sympos. Found. Comput. Sci., 1996, pp. 2–12.

[2] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala, *Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen*, in Proc. 27th Ann. ACM Sympos. Theory Comput., 1995, pp. 277–283.

[3] E. Balas, *The prize collecting traveling salesman problem*, Networks, 19 (1989), pp. 621–636.

[4] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson, *A note on the prize collecting traveling salesman problem*, Math. Programming, 59 (1993), pp. 413–420.

[5] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, *The minimum latency problem*, in Proc. 26th Ann. ACM Sympos. Theory Comput., 1994, pp. 163–171.

[6] A. Blum, P. Chalasani, and S. Vempala, *A constant-factor approximation for the k-*MST *problem in the plane*, in Proc. 27th Ann. ACM Sympos. Theory Comput., 1995, pp. 294–302.

[7] A. Blum, R. Ravi, and S. Vempala, *A constant-factor approximation algorithm for the k-*MST *problem*, in Proc. 28th Ann. ACM Sympos. Theory Comput., 1996, pp. 442–448.

[8] S. Y. Cheung and A. Kumar, *Efficient quorumcast routing algorithms*, in Proc. IEEE INFOCOM '94 Conference on Computer Communications, 2 (1994), pp. 840–847.

[9] D.-Z. Du, L.-Q. Pan, and M.-T. Shing, *Minimum Edge Length Guillotine Rectangular Partition*, Report 02418-86, Math. Sci. Res. Inst., Univ. California, Berkeley, CA, 1986.

[10] D. Eppstein, *Faster geometric k-point* MST *approximation*, Comput. Geom., 8 (1997), pp. 231–240.

[11] M. Fischetti, H. W. Hamacher, K. Jørnsten, and F. Maffioli, *Weighted k-cardinality trees: Complexity and polyhedral structure*, Networks, 24 (1994), pp. 11–21.

[12] N. Garg, *A* 3*-approximation for the minimum tree spanning k vertices*, in Proc. 37th Ann. IEEE Sympos. Found. Comput. Sci., 1996, pp. 302–309.

[13] N. Garg and D. S. Hochbaum, *An* $O(\log k)$ *approximation algorithm for the k minimum spanning tree problem in the plane*, in Proc. 26th Ann. ACM Sympos. Theory Comput., 1994, pp. 432–438.

[14] M. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, in Proc. 7th ACM-SIAM Sympos. Discrete Algorithms, 1996, pp. 152–158.

[15] M. Goemans and D. Williamson, *General approximation technique for constrained forest problems*, in Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms, 1992, pp. 307–315.

[16] C. Mata and J. S. B. Mitchell, *A Constant Factor Approximation Algorithm for the Red-Blue Separation Problem*, Dept. of Computer Science, SUNY, Stony Brook, NY, 1995, manuscript.

[17] C. Mata and J. S. B. Mitchell, *Approximation algorithms for geometric tour and network design problems*, in Proc. 11th Ann. ACM Sympos. Comput. Geom., 1995, pp. 360–369.

[18] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-*MST *problem*, in Proc. 7th ACM-SIAM Sympos. Discrete Algorithms, 1996, pp. 402–408.

[19] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems*, SIAM J. Comput., to appear.

[20] S. Rajagopalan and V. Vazirani, *Logarithmic Approximation of Minimum Weight k Trees*, manuscript, 1995.

[21] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi, *Spanning trees short and small*, in Proc. 5th ACM-SIAM Sympos. Discrete Algorithms, 1994, pp. 546–555.

[22] A. Zelikovsky and D. Lozevanu, *Minimal and bounded trees*, in Tezele Cong. XVIII Acad. Romano-Americane, Kishinev, 1993, pp. 25–26.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 782–797

SOLVABILITY OF CONSENSUS: COMPOSITION BREAKS DOWN FOR NONDETERMINISTIC TYPES*

PRASAD JAYANTI†

Abstract. *Consensus*, which requires processes with different input values to eventually agree on one of these values, is a fundamental problem in fault-tolerant computing. We study this problem in the context of asynchronous shared-memory systems. Prior research on consensus focused on its solvability using shared objects of *specific* types. In this paper, we investigate the following general question: *Let T and T' be any two types. Consider the consensus problem among N processes. Suppose that this problem is unsolvable if processes may use only objects of any one type (T or T') for communication. Does it follow that the problem is unsolvable even if processes may use objects of both types?* Recent results imply that the answer is positive if T and T' are both deterministic types. We prove that the answer is negative even if one of T and T' is nondeterministic.

Key words. asynchronous distributed computation, consensus, wait-free algorithms, nondeterministic object type

AMS subject classifications. 68Q10, 68Q22

PII. S0097539795280081

1. Introduction.

1.1. Background. In an asynchronous system, processes progress at independent and arbitrarily varying speeds. Consequently, the view that a process holds of (any aspect of) the global state of the computation does not necessarily coincide either with the reality or with the view of another process. Thus it often becomes necessary for processes to reconcile their differences and arrive at a mutually acceptable common view. The desirable requirements of such a reconciliation are captured by the *consensus problem*, which may be stated as follows. Each process is initially given a binary input and each noncrashing process is required to eventually decide a value such that (i) no two processes decide different values, and (ii) the decision value is the input of some process.

In this paper we study the consensus problem in the shared-memory model, where asynchronous processes communicate via linearizable typed shared objects [10]. An object's type specifies the operations that may be invoked and its *sequential behavior*: one or more legal sequences of responses corresponding to each sequence of nonoverlapping operations. (The type is *nondeterministic* if there is more than one legal sequence of responses for some sequence of operations.) `register`, `queue`, `test&set`, and `compare&swap` are some examples of types. (We will use the typewriter font for types. Thus `queue` refers to the type and *queue* refers to an object of this type.)

In most systems, simple shared objects, such as registers and test&set objects, are supported in hardware, but more complex objects, such as queues and sets, must be implemented in software. Consequently there has been much research on implementing objects of one type from objects of other types. Early implementations were based

*Received by the editors January 18, 1995; accepted for publication (in revised form) March 14, 1996; published electronically September 14, 1998. The result in this paper appeared without proof in *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*, August 1993. This work was supported by NSF grants CCR-9102231 and CCR-9410421, and Dartmouth College Startup grant.

http://www.siam.org/journals/sicomp/28-3/28008.html

†6211 Sudikoff Laboratory for Computer Science, Dartmouth College, Hanover, NH 03755 (prasad@cs.dartmouth.edu).

on critical sections (for example, see [6]), which are poorly suited to asynchronous systems: if one process is delayed in the critical section of an implemented object, the other processes will simply have to wait; even worse, if a process crashes in a critical section, the other processes are permanently disabled from accessing the object. To overcome this problem, Lamport advocated *wait-free implementations* [13]. A wait-free implementation has the property that every process can complete every operation on the implemented object in a finite number of its own steps, regardless of the speeds of the remaining processes.

The importance of the consensus problem became explicit when Herlihy discovered the following fundamental connection between consensus and wait-free implementations: given (i) a wait-free protocol that (repeatedly) solves the consensus problem among N processes and (ii) an unbounded array of registers, it is possible to have a wait-free implementation of an object of *any* type, where the implemented object can be shared by up to N processes [9]. Thus a consensus protocol can be regarded as a "universal" primitive.

The solvability of the consensus problem has been extensively researched. To succinctly describe these results, we use the following notation: we say that a *set $\mathcal{S}$ of types solves consensus for N processes* if there is a wait-free protocol $\mathcal{P}$ that solves the consensus problem among N processes such that *all* objects used in $\mathcal{P}$ for communication between processes are of types in $\mathcal{S}$. (There is no limit on the number of objects that $\mathcal{P}$ may use. We only require that each object be of a type in $\mathcal{S}$.) If such a protocol does not exist, we say that $\mathcal{S}$ does not solve consensus for N processes. If $\mathcal{S}$ is a singleton set $\{T\}$, we write "T solves consensus" instead of "$\{T\}$ solves consensus."

Dolev, Dwork, and Stockmeyer [7]; Loui and Abu-Amara [14]; and Chor, Israeli, and Li [4] proved that `register` does not solve consensus even for two processes. (These and most other impossibility results relating to consensus are proved using the bivalency technique introduced by Fischer, Lynch, and Paterson [8].) Loui and Abu-Amara proved that neither $\{$`test&set`, `register`$\}$ nor `2-valued read-modify-write` solves consensus for three processes, but each solves consensus for two processes [14]. They also exhibited a simple protocol to show that `3-valued read-modify-write` solves consensus for N processes for all N. Finally, Herlihy considered a host of common types—`queue`, `stack`, `fetch&add`, `move`, `compare&swap`, etc.—and analyzed, for each type T, the maximum number of processes for which $\{T,$ `register`$\}$ can solve consensus [9].

1.2. The main result. As described above, the ability of *specific types* to solve consensus has been well studied. In this paper we ask a general question: can we deduce the ability of a set of types to solve consensus simply from knowing the abilities of the individual types in the set? More specifically, consider the following compositional property of types T and T'.

PROP$_N(T, T')$. If neither T nor T' solves consensus for N processes, then $\{T, T'\}$ does not solve consensus for N processes.

For all types T, T' studied in the literature, PROP$_N(T, T')$ holds (for all N). The natural question follows.

QUESTION. Does PROP$_N(T, T')$ hold for all types T, T'?

We prove that the answer is a strong no. Specifically, we exhibit a nondeterministic type called `DAD`(∞) with the following two properties:

1. `DAD`(∞) does not solve consensus for two processes;
2. $\{$`DAD`(∞), `register`$\}$ solves consensus for N processes for any N.

From this and the known result that `register` does not solve consensus for two processes [4, 7, 14], it follows that $\text{PROP}_N(\texttt{DAD}(\infty), \texttt{register})$ is false for all $N \geq 2$.

1.3. Related results. Research relating the ability of a set of types to the abilities of the individual types in the set began with our work [11] and the work of Kleinberg and Mullainathan [12]. In [11], it is proved that the maximum number of processes for which $\{T, \texttt{register}\}$ solves consensus is reduced in general if any limit is imposed on the number of type T objects that may be used in the solution. Both [11] and [12] prove that the maximum number of processes for which a type T solves consensus is reduced in general if any limit is imposed on the number of type T objects that may be used in the solution. [11] also presents the result in this paper without proof.

Borowsky, Gafni, and Afek [2]; Peterson, Bazzi, and Neiger [16]; Chandra et al. [3] study the following question posed in [11]: If neither $\{T, \texttt{register}\}$ nor $\{T', \texttt{register}\}$ solves consensus for N processes, does it follow that $\{T, T', \texttt{register}\}$ does not solve consensus for N processes? For deterministic types T and T', [2] and [16] prove that the answer is yes. If types are nondeterministic and each process may bind to at most one "port" of each object, [3] proves that the answer is no in general. [3] also proves that if $\{T, \texttt{register}\}$ does not solve consensus for N processes, then $\{T, \texttt{register}, (N-1)-\texttt{consensus}\}$ does not solve consensus for N processes. Bazzi, Neiger, and Peterson prove that if either T is deterministic or T solves consensus for two processes, then the following holds: if T does not solve consensus for N processes then $\{T, \texttt{register}\}$ does not solve consensus for N processes [1]. Cori and Moran [5] prove a result similar to the one in [12].

2. Model and definitions.

2.1. Type. A *type* is a tuple (OP, RES, Q, δ), where OP is a set of operations, RES is a set of responses, Q is a set of states, and $\delta \subseteq Q \times OP \times Q \times RES$ is a relation known as the *sequential specification* of the type. Intuitively, if $(\sigma, op, \sigma', res) \in \delta$ it means the following: applying the operation op to an object in state σ can cause the object to move to state σ' and return the response res. δ is required to satisfy the following two properties.

1. *Totality.* For all $\sigma \in Q$ and $op \in OP$, there is at least one pair (σ', res) such that $(\sigma, op, \sigma', res) \in \delta$. (This condition ensures that it is legitimate to apply any operation in any state.)

2. *Computability.* There is a Turing machine M such that M halts on all inputs, and on input (σ, op) ($\sigma \in Q$, $op \in OP$), M computes at least one pair (σ', res) such that $(\sigma, op, \sigma', res) \in \delta$. (This condition ensures that a sequential implementation, one that is accessed by a single process, is feasible.)

A type is *deterministic* if, for all $\sigma \in Q$ and $op \in OP$, there is exactly one pair (σ', res) such that $(\sigma, op, \sigma', res) \in \delta$. Thus for deterministic types δ can be regarded as a function $\delta : Q \times OP \to Q \times RES$. A type is *nondeterministic* if it is not deterministic.

2.2. Concurrent system. We define a concurrent system informally. A formal definition, using I/O automata [15], was given in [9].

A *concurrent system* is specified by a finite set of processes $\{P_1, P_2, \ldots, P_N\}$ and a finite/infinite set of objects $\{O_1, O_2, \ldots\}$, where the following hold.

- Processes and objects have distinct names. (All names are known to all processes.)

• Each object has two attributes: a type and an initial value. (The initial value is a state of the object's type.)

• Each process is specified by a deterministic program. Some internal variables are distinguished as *input variables* and some are distinguished as write-once *output variables* (output variables are initialized to $\perp$). Each instruction of the program specifies which object to access, what operation to apply on the object, and how the object's response should alter the values of the internal variables of the program. We denote such a concurrent system as $(P_1, \ldots, P_N; O_1, O_2, \ldots)$.

A *configuration* of a concurrent system is a tuple of process and object states. Notice that the initial configuration is uniquely fixed by an assignment of values to input variables of processes. An *execution* of a concurrent system is a sequence $C_0, C_1, C_2, \ldots$ of configurations such that C_0 is an initial configuration, and C_{i+1} is the configuration that results when some process P executes a single instruction of its program in configuration C_i. We refer to the change of configuration from C_i to C_{i+1} as a *step* and associate this step with process P.

2.3. Consensus protocol. A *(binary) consensus protocol for processes* $P_1, \ldots, P_N$ is a concurrent system $(P_1, \ldots, P_N; O_1, O_2, \ldots)$, where each P_i has a binary input variable $proposal_i$ and an output variable $decision_i$ such that every infinite execution Σ has the following properties.

• *Wait-freedom.* If a process P_i has infinitely many steps in $\Sigma = C_0, C_1, C_2, \ldots$ then P_i *decides* in Σ; that is, there is a configuration C_k such that $decision_i$ has a non-$\perp$ value in C_k. (We refer to this non-$\perp$ value as P_i's *decision value* in Σ, and refer to the value of the input variable $proposal_i$ in C_0 as P_i's *proposal* in Σ.)

• *Agreement.* If P_i and P_j decide in Σ, then their decision values are the same.

• *Validity.* If P_i decides in Σ, then its decision value is the proposal of some process.

DEFINITION 2.1. *Let $\mathcal{S}$ be a set of types. We say $\mathcal{S}$* solves consensus for N processes *if there is a consensus protocol $(P_1, \ldots, P_N; O_1, O_2, \ldots)$ such that the type of each object O_i is in $\mathcal{S}$.*

DEFINITION 2.2. *We say $\mathcal{S}$* solves consensus for ∞ processes *if, for all $N \geq 1$, $\mathcal{S}$ solves consensus for N processes.*

We let CONSENSUS$(P_i, v_i, \mathcal{P})$ denote process P_i's program in consensus protocol $\mathcal{P}$ when P_i's proposal is v_i.

3. Specification of the type DAD(k). In this section we specify a family of types DAD(k), $k \in \{2, 3, 4, \ldots\} \cup \{\infty\}$. In the next two sections, we prove the following properties.[1]

1. For $k < \infty$, {DAD(k), `register`} solves consensus for k processes but not for $k + 1$ processes. {DAD(∞), `register`} solves consensus for ∞ processes.

2. For all $k \in \{2, 3, 4, \ldots\} \cup \{\infty\}$, DAD$(k)$ does not solve consensus for two processes.

The type DAD(k) is specified in Figure 1. In this specification, *choose*(S) is assumed to pick an element from set S nondeterministically and return it. Notice that the variables *upset* and *ahead*$[i]$ are stable: once true, they remain true. Similarly, once the variable *decision* assumes a binary value, its value does not subsequently change. The following is an informal description of DAD(k). This description is in-

[1] `DAD` stands for "disciplined-access demanding," a name that captures the fact that an object of this type does not return useful responses to processes unless processes show certain discipline in how they access the object.

S1. OP, the set of operations, is $\{\texttt{op}(0), \texttt{op}(1)\} \cup \{\texttt{give-decision}(i, b) \mid i \in \{0,1\}, b \in \{true, false\}\}$.

S2. RES, the set of responses, is $\{0, 1, ack\}$.
The response for $\texttt{op}(0)$ or $\texttt{op}(1)$ is always ack. The response for $\texttt{give-decision}(-,-)$ is either 0 or 1.

S3. Q, the set of states, is represented by the variables n_0, n_1, n_{gd} : `integer`; $decision \in \{\perp, 0, 1\}$; $ahead[0..1]$, $upset$: `boolean`. The state corresponding to ($n_0 = n_1 = n_{gd} = 0$; $decision = \perp$; $ahead[0..1] = upset =$ false) is known as the *fresh state*. The states of $\texttt{DAD}(k)$ are *only* those that are reachable from the fresh state by the sequential specification given in **S4**.

Informally, n_0, n_1, n_{gd} count the number of applications of $\texttt{op}(0)$, $\texttt{op}(1)$, and `give-decision`, respectively. The variable $ahead[i]$ is set to $true$ if $n_i > 0$ and $n_{\bar{i}} = 0$ when $\texttt{give-decision}(i, -)$ is applied. The variable $upset$ is set to $true$ if one of the following happens: (i) $\texttt{op}(1)$ is applied more than once ($\texttt{op}(0)$ may be applied any number of times), (ii) the total number of times that all $\texttt{give-decision}(-,-)$ operations are applied is more than k, (iii) $\texttt{give-decision}(i, -)$ is applied with no prior application of $\texttt{op}(i)$, (iv) $\texttt{give-decision}(i, true)$ is applied with no prior application of $\texttt{op}(\bar{i})$, (v) $\texttt{give-decision}(i,$ false) is applied and $ahead\ [\bar{i}] = true$. If upset, a $\texttt{DAD}(k)$ object returns 0 or 1 nondeterministically to an invocation of `give-decision`. If not upset, it sets $decision$ irrevocably and nondeterministically (if not already set) to 0 or 1 such that $n_{decision} > 0$, and returns $decision$.

S4. δ, the sequential specification, is described as follows:

```
op(i)                                   /*  i ∈ {0,1}  */
    n_i := n_i + 1
    if n_1 > 1 then upset := true
    return(ack)

give-decision(i, other-is-ahead)        /*  i ∈ {0,1},  other-is-ahead: boolean  */
    n_gd := n_gd + 1
    if (n_i > 0 ∧ n_ī = 0) then ahead[i] := true
    if (n_gd > k) ∨ (n_i = 0) ∨ (ahead[ī] ∧ ¬other-is-ahead) ∨ (n_ī = 0 ∧ other-is-ahead) then
        upset := true
    if upset then
        return(choose({0,1}))
    else if decision = ⊥ then
        decision := choose({j | n_j > 0})
    return(decision)
```

FIG. 1. *The type* $\texttt{DAD}(k)$.

tended only as an intuitive guide to, and not as a substitute for, the formal specification in Figure 1.

A $\texttt{DAD}(k)$ object supports three types of operations: $\texttt{op}(0)$, $\texttt{op}(1)$, and $\texttt{give-decision}(-,-)$. (The first parameter of `give-decision` is a 0 or 1, and the second parameter is either *true* or *false*.) The response to $\texttt{op}(0)$ or $\texttt{op}(1)$ is always *ack*. The response to $\texttt{give-decision}(-,-)$ is either 0 or 1. A $\texttt{DAD}(k)$ object requires that certain rules be observed in accessing it. If these rules are violated, the object becomes "upset." Once upset, the object remains upset forever. Below we explain (i) the conditions under which an object becomes upset and (ii) how an object computes the response (0 or 1) to a `give-decision` operation.

We say that $\texttt{op}(0)$ *is ahead of* $\texttt{op}(1)$ if, before $\texttt{op}(1)$ is applied for the first time, both $\texttt{op}(0)$ and $\texttt{give-decision}(0, -)$ are applied and are applied in that order. (It is not necessary that $\texttt{give-decision}(0, -)$ be applied immediately after $\texttt{op}(0)$). The definition of "$\texttt{op}(1)$ *is ahead of* $\texttt{op}(0)$" is symmetric. Notice that once the proposition "$\texttt{op}(i)$ is ahead of $\texttt{op}(\bar{i})$" becomes true, it remains true forever.

A DAD(k) object becomes upset if *any* of the following conditions is met. Furthermore, once upset, an object will remain upset forever.

1. op(1) is invoked more than once.

2. The total number of times that all give-decision($-,-$) operations are invoked is more than k.

3. For $i \in \{0,1\}$, give-decision($i,-$) is invoked, but op(i) has never been previously invoked (by any process).

4. For $i \in \{0,1\}$, give-decision(i, *false*) is invoked and op($\bar{i}$) is ahead of op(i).

5. For $i \in \{0,1\}$, give-decision(i, *true*) is invoked and op($\bar{i}$) has never been previously invoked (by any process).

Conditions 1 and 2 are easy to comprehend. Condition 3 states that op(i) must be invoked before give-decision($i,-$) can be invoked. Condition 4 states that if op($\bar{i}$) is ahead of op(i), then in any invocation of give-decision(i, *flag*), *flag* must be *true*. By itself, this condition is trivial to meet: whenever a process invokes give-decision($-,-$), it can always supply *true* as the second parameter. Condition 5 prevents processes from adopting such a trivial strategy. It states that if give-decision(i, *true*) is invoked, then it must be the case that op($\bar{i}$) has been previously invoked.

The operations op(0) and op(1) always get *ack* as their response. The response to an invocation INV of the give-decision operation is computed using the following rules *in the order they are specified*.

1. If the object was already upset or the current invocation INV upsets the object, the object returns 0 or 1 nondeterministically.

2. If some give-decision($-,-$) operation has previously completed and the object returned a response *res* to that operation, then the object returns *res*.

3. If op(0) has been previously invoked and op(1) has not been previously invoked, the object returns 0. Similarly, if op(1) has been previously invoked and op(0) has not been previously invoked, the object returns 1.

4. If op(0) and op(1) have both been previously invoked (and, because of rule 2, a give-decision operation has never been previously invoked), the object returns 0 or 1 nondeterministically.

The above is a complete set of rules. In particular, we do not need a rule 5 for the case when neither op(0) nor op(1) was previously invoked. This is because if neither op(0) nor op(1) was previously invoked, then the current invocation INV will upset the object, making rule 1 applicable.

The motivation for this involved specification is as follows. Consider the *name-consensus problem* for two processes P_0 and P_1 [9]. Informally, this problem requires two processes P_0 and P_1 to agree which, between them, is the winner. The nontriviality condition is that the winner must have taken at least one step. Consider the following protocol for this problem. Process P_i ($i \in \{0,1\}$) applies op(i) on $\mathcal{O}$, a DAD(k) object. P_i then applies give-decision(i, $flag_i$) on $\mathcal{O}$ (for some $flag_i$) and, if the response is j, it decides P_j to be the winner. This protocol is correct only if each process ensures that the second parameter $flag_i$ supplied while invoking give-decision does not upset $\mathcal{O}$. Thus, intuitively, DAD(k) objects are useful in solving consensus only if processes ensure that they do not upset these objects while accessing them. We have carefully specified DAD(k) so that, while solving consensus, processes can usefully access DAD(k) objects, without upsetting them, only when registers are also available. This will help us show that DAD(k) does not solve consensus but {DAD(k), register} does.

O_{dad}: DAD(k) object, initialized to the fresh state
R[0..1]: binary registers (arbitrarily initialized)
R'[0..1]: boolean registers, initialized to *false*

internal variables of process P_i
$proposal_i \in \{0, 1\}$
$decision_i \in \{\perp, 0, 1\}$, initialized to $\perp$
$d_i, winner_i \in \{0, 1\}$ (arbitrarily initialized)
$other\text{-}ahead_i$: boolean (arbitrarily initialized)

CONSENSUS($P_i, proposal_i, \mathcal{P}_n^k$) (for $1 \le i \le n-1$)

1. $d_i :=$ CONSENSUS($P_i, proposal_i, \mathcal{P}_{n-1}^k$)
2. $R[0] := d_i$
3. Apply(P_i, op(0), O_{dad})
4. $R'[0] := true$
5. $other\text{-}ahead_i := R'[1]$
6. $winner_i :=$ Apply(P_i, give-dec(0, $other\text{-}ahead_i$), O_{dad})
7. $decision_i := R[winner_i]$

CONSENSUS($P_n, proposal_n, \mathcal{P}_n^k$)

$d_n := proposal_n$
$R[1] := d_n$
Apply(P_n, op(1), O_{dad})
$R'[1] := true$
$other\text{-}ahead_n := R'[0]$
$winner_n :=$ Apply(P_n, give-dec(1, $other\text{-}ahead_n$), O_{dad})
$decision_n := R[winner_n]$

FIG. 2. $\mathcal{P}_n^k$, *consensus protocol for processes* $P_1, \ldots, P_n$.

4. {DAD(k), register} **solves consensus for** k **processes.** In this section, we exhibit a consensus protocol for k processes using only DAD(k) objects and registers. Our protocol is recursive. Let $\mathcal{P}_n^k$ ($n \le k$) denote a consensus protocol for processes $P_1, P_2, \ldots, P_n$ that uses only DAD(k) objects and registers. The base case is to derive $\mathcal{P}_1^k$, a consensus protocol for a single process P_1, and is trivial. The recursive step of deriving $\mathcal{P}_n^k$ from $\mathcal{P}_{n-1}^k$ (for all $n > 1$) is presented in Figure 2.

The protocol $\mathcal{P}_n^k$ works as follows. Processes $P_1 \ldots P_n$ split into two groups: G_0 and G_1. Group G_0 has $P_1 \ldots P_{n-1}$, and group G_1 has just P_n. Processes $P_1 \ldots P_{n-1}$ do consensus among themselves (recursively) and announce the outcome in $R[0]$. Process P_n announces its input value in $R[1]$. The rest of the protocol resolves which of the two groups is the winner. If G_0 wins, every process decides the value in $R[0]$. Similarly, if G_1 wins, every process decides the value in $R[1]$. The object O_{dad} is used to determine the winner of the two groups. Processes $P_1 \ldots P_{n-1}$ perform the operation op(0) on O_{dad}. Then they set the register $R'[0]$ to inform process P_n that op(0) has been applied on O_{dad}. Process P_n, on the other hand, performs op(1) on O_{dad}, and then sets $R'[1]$ to inform processes in G_0 that op(1) has been applied. Processes then perform the give-decision operation. The return value determines the winning group. For this strategy to work correctly, the arguments of the give-decision operation must be such that the object O_{dad} does not get upset. We urge the reader to understand how the registers $R'[0..1]$ are used to ensure that O_{dad} does not get upset. (This will be clear when we prove below that the protocol is correct.) Finally, if O_{dad} returns v, a process assumes that the group G_v won and decides the value in $R[v]$.

LEMMA 4.1. *For* $2 \le n \le k$, *the protocol* $\mathcal{P}_n^k$ *in Figure* 2 *is a correct consensus protocol for processes* $P_1, P_2, \ldots, P_n$.

Proof. The proof is by induction. Assume that $\mathcal{P}_{n-1}^k$ is correct. Consider an execution E of the consensus protocol $\mathcal{P}_n^k$ in Figure 2. The key claim is that O_{dad} does not get upset in E. This claim follows from the following simple observations.

1. op(1) is executed only once.

2. For $v \in \{0,1\}$, op(v) is executed before executing give-decision($v,-$).

3. The total number of times that all give-decision($-,-$) operations are executed is no more than n. Since $n \leq k$, the total number of times that all give-decision($-,-$) operations are executed is no more than k.

4. Suppose op(v) is ahead of op($\overline{v}$). That is, the operations op(v) and then give-decision($v,-$) are completed before the first invocation of op($\overline{v}$). Then the use of the registers $R'[0..1]$ in the protocol guarantees that when a process invokes give-decision($\overline{v}$, *other-ahead*), the second parameter, namely, *other-ahead*, is *true*.

5. Suppose no process completes the operation op(v) before some process invokes give-decision($\overline{v}$, *other-ahead*). Then the use of the registers $R'[0..1]$ in the protocol guarantees that the second parameter, namely, *other-ahead*, is *false*.

So far we have argued that O_{dad} is not upset, in E. Since O_{dad} is not upset, by the specification of DAD(k) we have the following.

1. Every give-decision operation on O_{dad} returns the same binary response. Let *winner* $\in \{0,1\}$ denote this response.

2. Some process P_j invokes op(*winner*) before O_{dad} returns *winner* for the first time to a give-decision operation.

From the protocol, it is clear that P_j writes the value d_j in $R[winner]$ before invoking op(*winner*). Furthermore, once a value is written by a process into a register $R[0]$ or $R[1]$, the value of that register never subsequently changes. For $R[0]$ this follows from the agreement property of $\mathcal{P}^k_{n-1}$, and for $R[1]$ this follows from the fact that only P_n writes $R[1]$ and writes it only once.

The above implies that, for all i, $1 \leq i \leq n$, CONSENSUS($P_i, proposal_i, \mathcal{P}^k_n$) returns d_j. Thus the protocol $\mathcal{P}^k_n$ satisfies agreement. If $j = n$, then $d_j = d_n = proposal_n$. If $j \neq n$, since $\mathcal{P}^k_{n-1}$ satisfies validity, $d_j \in \{proposal_1, proposal_2, \ldots, proposal_{n-1}\}$. Thus $\mathcal{P}^k_n$ satisfies validity. It is obvious that $\mathcal{P}^k_n$ is wait-free. This concludes the proof of correctness. □

COROLLARY 4.2. *For $k \in \{2,3,\ldots\} \cup \{\infty\}$, {DAD($k$), register} solves consensus for k processes.*

In fact, as the next lemma states, it is impossible to solve consensus for $k+1$ processes using only DAD(k) objects and registers. This lemma is however not essential for establishing our main result. The proof is therefore presented in the appendix.

LEMMA 4.3. *For $k < \infty$, {DAD(k), register} does not solve consensus for $k+1$ processes.*

Proof. It is proved by a standard bivalency argument. See the appendix. □

4.1. DAD(k) does not solve consensus for even two processes. In this section, we prove that it is impossible to solve consensus even between two processes if we may *only* use DAD(k) objects for process communication.

Impossibility results such as these are typically proved using bivalency arguments [8]. However, bivalency arguments did not appear helpful in proving this result. This is due to the fact that DAD(k) objects are not entirely powerless: after all, as we showed in the previous section, it is possible to solve consensus using DAD(k) objects if registers are also available. Our proof is based on exhibiting two indistinguishable scenarios in which processes are required to act differently. The nondeterminism of DAD(k) is exploited in keeping the scenarios indistinguishable. The details are rather involved since arguments concerning nondeterministic objects are subtle and warrant a careful treatment.

We begin with a simple lemma that will be useful later. The lemma states that it

op(i)	give-decision(i, b)

return(ack)	**if** $\sigma[decision] \in \{0,1\}$ **then**
	  return($\sigma[decision]$)
	else if $(\sigma[upset] \vee \sigma[n_0] > 0)$ **then**
	  return(0)
	else return(1)

FIG. 3. *Simulating the responses of a* DAD(k) *object whose initial state σ is a nonfresh state.*

is trivial to simulate the responses of a DAD(k) object whose initial state is not fresh. More specifically, if the initial state of a DAD(k) object O is not fresh, every process can determine the response from O to every operation locally, without invoking any operation on O or any other shared object. In the following, let $\sigma[v]$ denote the value of state variable v in state σ.

LEMMA 4.4. *Let σ be any state of* DAD(k) *different from the fresh state. Let $\mathcal{O}$ be a* DAD(k) *object whose initial state is σ. The responses from $\mathcal{O}$ to operations can be (trivially) simulated using the code in Figure* 3.

Proof. If σ is different from the fresh state, then it is easy to verify that $(\sigma[decision] \in \{0,1\}) \vee (\sigma[n_0] > 0) \vee (\sigma[n_1] > 0) \vee \sigma[upset]$. From this and the specification of DAD(k), it is obvious that the responses are correct. □

COROLLARY 4.5. *Suppose there is a consensus protocol $\mathcal{P} = (P_1, P_2; O_1, O_2, \ldots)$ where each O_i is a* DAD(k) *object. Then there is a consensus protocol $\mathcal{P}' = (P_1, P_2; O_1', O_2', \ldots)$ where each O_i' is a* DAD(k) *object initialized to the fresh state.*

Proof. The protocol $\mathcal{P}'$ is the same as $\mathcal{P}$ with one exception: if $\mathcal{P}$ employs O_i and O_i's initial state is not fresh, then $\mathcal{P}'$ does not employ O_i; instead, in $\mathcal{P}'$, each process determines the response of O_i locally, as described in the proof of Lemma 4.4. □

The next lemma states that it is impossible to solve consensus between two processes using only DAD(k) objects. The proof exploits the fact that DAD(k) objects are so weak that a process cannot use these objects to leave its "footprints" behind. Thus if a process P_0 runs to completion and decides a value before the second process P_1 even begins to run, P_1 cannot figure out that P_0 had already run and decided upon a value. This can cause P_1 to decide upon a value that is not consistent with the decision of P_0. The formal proof below elaborates this argument. However, the details are complex. We therefore present the proof in a series of easily verifiable claims.

At some places in the proof, we refer to the type det-DAD(k). The specification of det-DAD(k) is obtained by replacing every occurrence of "choose(S)" in Figure 1 by the function "min(S)." (min(S) is the minimum element in set S.) Notice that this modification makes det-DAD(k) a deterministic type.

LEMMA 4.6. *For all $k \in \{2, 3, \ldots\} \cup \{\infty\}$,* DAD($k$) *does not solve consensus for two processes.*

Proof. The proof is by contradiction. Let $(P_0, P_1; O_1, O_2, \ldots)$ be a consensus protocol $\mathcal{P}$, where each O_i is of type DAD(k). By Corollary 4.5, we can assume, without loss of generality, that the initial state of each O_i is fresh.

Recall our notation that CONSENSUS($P_i, v_i, \mathcal{P}$) denotes process P_i's program in protocol $\mathcal{P}$ when P_i's proposal is v_i. In the following, we present two scenarios, $\mathtt{S}_0$

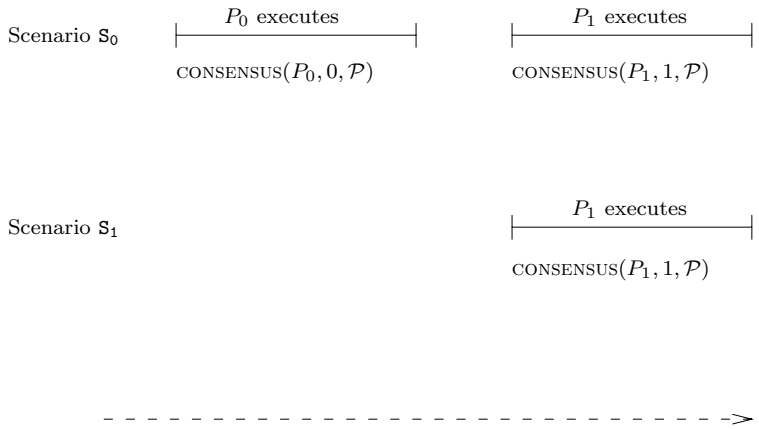


FIG. 4. *Scenarios* $\mathtt{S}_0$ *and* $\mathtt{S}_1$.

and $\mathtt{S}_1$, which are indistinguishable to P_1 but require P_1 to take different actions.

In Scenario $\mathtt{S}_0$, P_0 executes CONSENSUS$(P_0, 0, \mathcal{P})$ to completion. Assume that, during this execution of CONSENSUS$(P_0, 0, \mathcal{P})$, every object O_i $(i \geq 1)$ behaves like a $\mathtt{det\text{-}DAD}(k)$ object. We will refer to this as Assumption A1. Since $\mathcal{P}$ satisfies validity, at the completion of CONSENSUS$(P_0, 0, \mathcal{P})$, P_0 decides 0. Of $O_1, O_2, \ldots$ let $\mathcal{S}$ be the set of objects that are in the fresh state in scenario $\mathtt{S}_0$ at the completion of CONSENSUS$(P_0, 0, \mathcal{P})$.

We now do two things: (i) we continue scenario $\mathtt{S}_0$ by letting P_1 execute CONSENSUS$(P_1, 1, \mathcal{P})$, and (ii) we start a new scenario, call it scenario $\mathtt{S}_1$, by letting P_1 execute CONSENSUS$(P_1, 1, \mathcal{P})$. (See Figure 4 for a depiction of scenarios $\mathtt{S}_0$ and $\mathtt{S}_1$.) Assume that, in both scenarios, each object in $\mathcal{S}$ behaves deterministically, consistent with the type $\mathtt{det\text{-}DAD}(k)$. We will refer to this as Assumption A2.

We prove the following statement by induction on i: the objects in $\{O_1, O_2, \ldots\} - \mathcal{S}$ can choose among the nondeterministic alternatives (when applicable) such that for all $i \geq 0$, P_1 cannot distinguish scenario $\mathtt{S}_0$ from scenario $\mathtt{S}_1$ in its first i steps. The base case for $i = 0$ is trivial. To prove the induction step, assume the hypothesis for $i \leq m$.

Consider the $(m+1)$th step. Let *oper* be the operation that P_1 performs in this step in scenario $\mathtt{S}_0$, and let O be the object on which it performs *oper*. From the induction hypothesis and the fact that the protocol $\mathcal{P}$ is deterministic, it follows that P_1 performs *oper* on O in its $(m+1)$th step in scenario $\mathtt{S}_1$ also.

Suppose *oper* $\in \{\mathtt{op}(0), \mathtt{op}(1)\}$. Then the response is *ack* in either scenario. Thus $\mathtt{S}_0$ and $\mathtt{S}_1$ remain indistinguishable to P_1 after $m+1$ steps. Hence the induction step is proved.

Suppose instead that *oper* is $\mathtt{give\text{-}decision}(-,-)$. We make a case analysis to prove the induction step.

Case 0. $O \in \mathcal{S}$.

We claim that O is in the fresh state in both $\mathtt{S}_0$ and $\mathtt{S}_1$ just before P_1 begins executing CONSENSUS$(P_1, 1, \mathcal{P})$. For $\mathtt{S}_0$, this follows from the definition of $\mathcal{S}$; and for

$\mathtt{S}_1$, this follows from the fact that every object is initialized to the fresh state. By Assumption A2, O behaves deterministically (consistent with the type $\mathtt{det\text{-}DAD}(k)$) in both scenarios. The above facts, together with induction hypothesis, guarantee that O is in the same state in both scenarios at the end of m steps of P_1 and, therefore, O returns the same response to *oper* in both scenarios. Thus $\mathtt{S}_0$ and $\mathtt{S}_1$ remain indistinguishable to P_1 after $m+1$ steps. Hence the induction step is proved.

Case 1. Case 0 does not apply and the following holds: in at least one of $\mathtt{S}_0$ and $\mathtt{S}_1$, O is upset in the first $m+1$ steps of P_1.

Let $\mathtt{S}_i$ be a scenario in which O is upset in the first $m+1$ steps of P_1. By the specification of $\mathtt{DAD}(k)$, O is free to return 0 or 1 to *oper* in scenario $\mathtt{S}_i$. Suppose that O uses this freedom and returns the same response to *oper* in $\mathtt{S}_i$ as it does in $\mathtt{S}_{\overline{i}}$. Then $\mathtt{S}_0$ and $\mathtt{S}_1$ remain indistinguishable to P_1 after $m+1$ steps. Hence the induction step is proved.

Case 2. Neither Case 0 nor Case 1 applies. In other words, O is not in the fresh state in scenario $\mathtt{S}_0$ just before P_1 calls CONSENSUS$(P_1, 1, \mathcal{P})$ and, in both $\mathtt{S}_0$ and $\mathtt{S}_1$, O is not upset at the end of $m+1$ steps of P_1.

Our proof that the induction step holds for this case proceeds as follows. First, we make claims (C1 to C11 below) and show that the induction step holds if *any* of the claims is false. Then we show that if all of claims C1 to C11 are true, the induction step must hold.

Let σ_0^k and σ_1^k denote the state of O at the end of k steps of P_1 in scenarios $\mathtt{S}_0$ and $\mathtt{S}_1$, respectively.

C1. $\sigma_1^m[n_{gd}] = 0$.[2] In other words, P_1 does not apply a `give-decision` operation on O in its first m steps in scenario $\mathtt{S}_1$.

Suppose that the claim is false. Let $k \leq m$ be the smallest integer such that $\sigma_1^k[n_{gd}] = 1$. That is, `give-decision` is applied on O for the first time by P_1 in its kth step in scenario $\mathtt{S}_1$. Since O is not upset in $\mathtt{S}_1$, it follows that $\sigma_1^k[\mathit{decision}] \in \{0,1\}$, and this value is the response from O in the kth step of P_1 in scenario $\mathtt{S}_1$. Let $d = \sigma_1^k[\mathit{decision}]$. By inductive hypothesis, the response from O in the kth step of P_1 in scenario $\mathtt{S}_0$ is also d. Since O is not upset in scenario $\mathtt{S}_0$, this implies that $\sigma_0^k[\mathit{decision}] = d$. From the specification of $\mathtt{DAD}(k)$, it is clear that once the state variable *decision* assumes a non-$\perp$ value, its value does not subsequently change. Thus we have $\sigma_0^m[\mathit{decision}] = d$ and $\sigma_1^m[\mathit{decision}] = d$. From this and the fact that O is not upset in either scenario, we conclude that the response from O to *oper* is d in both scenarios. Thus $\mathtt{S}_0$ and $\mathtt{S}_1$ remain indistinguishable to P_1 after $m+1$ steps. Hence the induction step is proved.

C2. There is a $v \in \{0,1\}$ such that $\sigma_1^m[n_v] > 0$ and $\sigma_1^m[n_{\overline{v}}] = 0$. In other words, P_1 applies $\mathtt{op}(v)$ but not $\mathtt{op}(\overline{v})$ in its first m steps in $\mathtt{S}_1$. (This v is fixed in the remainder of this proof.)

Assume that the claim is false. Then either $\sigma_1^m[n_0] = \sigma_1^m[n_1] = 0$ or $\sigma_1^m[n_0] > 0$ and $\sigma_1^m[n_1] > 0$. Suppose $\sigma_1^m[n_0] = \sigma_1^m[n_1] = 0$. Then by the specification of $\mathtt{DAD}(k)$, when P_1 applies $\mathit{oper} \equiv \mathtt{give\text{-}decision}(-,-)$ in the $(m+1)$th step in $\mathtt{S}_1$, it upsets O. This contradicts the case we are considering.

Suppose $\sigma_1^m[n_0] > 0$ and $\sigma_1^m[n_1] > 0$. Since $\sigma_1^m[n_{gd}] = 0$ (by C1), by the specification of $\mathtt{DAD}(k)$, O is free to return either 0 or 1 in $\mathtt{S}_1$. Suppose that O uses this freedom and returns the same response to *oper* in $\mathtt{S}_1$ as it does in $\mathtt{S}_0$. Then $\mathtt{S}_0$ and $\mathtt{S}_1$ remain indistinguishable to P_1 after $m+1$ steps. Thus the induction step holds.

[2]Recall our notation that $\sigma[v]$ denotes the value of the state variable v in state σ.

C3. P_1 applies $\texttt{op}(v)$ on O at least once in its first m steps in $\texttt{S}_0$.

This claim follows from C2 and the induction hypothesis.

C4. $oper \equiv \texttt{give-decision}(v, false)$.

Suppose $oper \equiv \texttt{give-decision}(\overline{v}, -)$ or $oper \equiv \texttt{give-decision}(v, true)$. Since $\sigma_1^m[n_{\overline{v}}] = 0$ (by C2), O will be upset in $\texttt{S}_1$ when *oper* is invoked in the $(m+1)$th step. This contradicts the case we are considering.

So far we made the following claims for the case in consideration (Case 2): there is some $v \in \{0, 1\}$ such that, in both $\texttt{S}_0$ and $\texttt{S}_1$, P_1 applies $\texttt{op}(v)$ on O but not $\texttt{op}(\overline{v})$ in its first m steps and applies $\texttt{give-decision}(v, \text{false})$ in its $(m+1)$th step.

C5. $\sigma_0^m[ahead[\overline{v}]] = false$.

Suppose $\sigma_0^m[ahead[\overline{v}]] = true$. Then when P_1 applies $oper \equiv \texttt{give-decision}(v, \text{false})$ (guaranteed by C4) in its $(m+1)$th step in $\texttt{S}_0$, it upsets O. This contradicts the case we are considering.

C6. $v = 1$ implies $\sigma_0^0[n_{gd}] = 0$. In other words, if $v = 1$, then P_0 never applied a `give-decision` operation on O in $\texttt{S}_0$.

Suppose $v = 1$ and P_0 applied $\texttt{give-decision}(1, -)$ on O in $\texttt{S}_0$. Since O is not upset in $\texttt{S}_0$, it follows that P_0 applied $\texttt{op}(1)$ on O before applying $\texttt{give-decision}(1, -)$. By C3 and the assumption that $v = 1$, P_1 applied $\texttt{op}(1)$ in $\texttt{S}_0$. Thus $\texttt{op}(1)$ was applied at least twice on O in $\texttt{S}_0$. By the specification of $\texttt{DAD}(k)$, O would be upset in $\texttt{S}_0$. This contradicts the case we are considering.

Suppose $v = 1$ and P_0 applied $\texttt{give-decision}(0, -)$ on O in $\texttt{S}_0$. Since O is not upset in $\texttt{S}_0$, it follows that P_0 applied $\texttt{op}(0)$ on O before applying $\texttt{give-decision}(0, -)$. By C5 and the assumption that $v = 1$, $\sigma_0^m[ahead[0]] =$ false. This implies that P_0 applied $\texttt{op}(1)$ on O before applying $\texttt{give-decision}(0, -)$. By C3 and the assumption that $v = 1$, P_1 applied $\texttt{op}(1)$ in $\texttt{S}_0$. Thus $\texttt{op}(1)$ was applied at least twice on O in $\texttt{S}_0$. By the specification of $\texttt{DAD}(k)$, O would be upset in $\texttt{S}_0$. This contradicts the case we are considering.

C7. $v = 0$.

Suppose $v = 1$. Then we can infer the following: (1) $\sigma_1^m[n_{gd}] = 0$ (by C1); (2) $\sigma_0^m[n_{gd}] = 0$ (by C1, induction hypothesis, and C6); (3) $\sigma_1^m[n_1] > 0$ (by C2); (4) $\sigma_0^m[n_1] > 0$ (by C3). These four facts, together with the specification of $\texttt{DAD}(k)$, imply that O is free to return 1 to *oper* in both $\texttt{S}_0$ and $\texttt{S}_1$. Suppose that O does this. Then $\texttt{S}_0$ and $\texttt{S}_1$ remain indistinguishable to P_1 after $m+1$ steps. Thus the induction step holds.

The claims made so far assert that, in both $\texttt{S}_0$ and $\texttt{S}_1$, P_1 applies $\texttt{op}(0)$ on O, but not $\texttt{op}(1)$, in its first m steps, and applies $\texttt{give-decision}(0, \text{false})$ in its $(m+1)$th step. Furthermore, $\texttt{op}(1)$ is not ahead of $\texttt{op}(0)$ in scenario $\texttt{S}_0$.

C8. O returns 0 to *oper* (in the $(m+1)$th step of P_1) in scenario $\texttt{S}_1$.

Claims C2 and C7 imply that $\sigma_1^m[n_0] > 0$ and $\sigma_1^m[n_1] = 0$. Further, by the case we are considering, O is not upset in the first $m+1$ steps of P_1 in scenario $\texttt{S}_1$. The above facts imply that the only legal response that O can return to *oper* is 0.

C9. If P_0 applied $\texttt{give-decision}(1, -)$ on O (in $\texttt{S}_0$), it did so only after applying $\texttt{op}(0)$ on O.

Suppose P_0 applied $\texttt{give-decision}(1, -)$ on O (in scenario $\texttt{S}_0$). Since O is not upset in $\texttt{S}_0$, this implies that P_0 applied $\texttt{op}(1)$ on O before applying $\texttt{give-decision}(1, -)$. If P_0 did not apply $\texttt{op}(0)$ before applying `give-`

`decision`$(1,-)$, then this application of `give-decision`$(1,-)$ would set $ahead[1]$ to *true*. This, together with the fact that $ahead[1]$ is stable, implies that $\sigma_0^m[ahead[1]] = true$. This contradicts the conjunction of C5 and C7.

C10. Every application of the operation `give-decision`$(-,-)$ on O by P_0 in scenario $\mathtt{S}_0$ gets the response 0.

Consider the earliest application e of `give-decision`$(w,-)$ on O by P_0 in $\mathtt{S}_0$. If $w = 1$, C9 implies that P_0 applies `op`(0) before e. If $w = 0$, the fact that O is not upset in $\mathtt{S}_0$ implies that P_0 applies `op`(0) before e. Thus we conclude that P_0 applies `op`(0) before e. This, together with Assumption A1, implies that e returns 0. From this and the fact that O is not upset in $\mathtt{S}_0$, it follows that every application of `give-decision`$(-,-)$ on O in $\mathtt{S}_0$ returns the response 0.

C11. P_0 never applies `give-decision`$(-,-)$ on O (in $\mathtt{S}_0$).

Suppose that the claim is false. Then, from C10 and the fact that O is not upset in $\mathtt{S}_0$, it follows that O returns 0 to *oper* in the $(m+1)$th step of P_1 in scenario $\mathtt{S}_0$. Thus, by C8, $\mathtt{S}_0$ and $\mathtt{S}_1$ remain indistinguishable to P_1 after $m+1$ steps. Hence the induction step is proved.

From the above claims we deduce the following facts: (F1) $\sigma_1^m[n_0] > 0$. This follows from C2 and C7. (F2) $\sigma_0^m[n_0] > 0$. This follows from F1 and induction hypothesis. (F3) $\sigma_0^m[n_{gd}] = 0$. This follows from C1, induction hypothesis, and C11. From F2, F3, and the specification of `DAD`(k), it is clear that O is free to return 0 to *oper* (in the $(m+1)$th step of P_1) in scenario $\mathtt{S}_0$. Suppose that it does. Then, by C8, $\mathtt{S}_0$ and $\mathtt{S}_1$ remain indistinguishable to P_1 after $m+1$ steps. Thus the induction step holds. This completes the proof of the induction step for Case 2.

We have proved the induction step for all cases. We therefore conclude that no matter how many steps P_1 takes, scenarios $\mathtt{S}_0$ and $\mathtt{S}_1$ can remain indistinguishable to P_1.

Since $\mathcal{P}$ is a wait-free protocol, CONSENSUS$(P_1, 1, \mathcal{P})$ terminates in $\mathtt{S}_0$ after a finite number of steps, with P_1 deciding some value $val \in \{0,1\}$. Since $\mathtt{S}_0$ and $\mathtt{S}_1$ are indistinguishable to P_1, CONSENSUS$(P_1, 1, \mathcal{P})$ terminates in $\mathtt{S}_1$ after exactly the same number of steps, with P_1 deciding val. If $val = 0$, the protocol does not satisfy the validity property in $\mathtt{S}_1$. If $val = 1$, the protocol does not satisfy the agreement property in $\mathtt{S}_0$. Thus $\mathcal{P}$ is not a correct consensus protocol. This completes the proof of the lemma. □

We conclude this section with some observations on Lemma 4.6 and its proof above. Nondeterminism occurs at two different places in the specification of `DAD`(k)(see the *choose* statement on lines 6 and 8 of the `give-decision` procedure in Figure 1). The proof of Lemma 4.6 exploits both occurrences, the first occurrence in proving the induction step for Case 1 and the second occurrence in proving the induction step for Case 2 (see the proofs of claims C2 and C7 and the conclusion of Case 2).

Is it possible to eliminate *either* occurrence of nondeterminism from the specification of `DAD`(k) and still prove Lemma 4.6? The answer appears to be no. Specifically, if either occurrence of the *choose* function is replaced with the deterministic function min (or max), it can be shown that Lemma 4.6 does not hold.

Lemma 4.6 states that the binary consensus problem for two processes is not solvable using `DAD`(k) objects. It is easy to show that binary consensus for N processes is solvable using objects of type T if and only if name-consensus for N processes is solvable using objects of type T. (The name-consensus problem is described earlier in section 3.) It follows that the name-consensus problem for two processes is also not solvable using `DAD`(k) objects alone.

5. The main theorem and conclusion. Recall from section 1.2 the property PROP$_N$ defined for pairs of types. From Lemma 4.6, Corollary 4.2, and the fact that `register` does not solve consensus for two processes [4, 7, 14], we conclude that, for $N \geq 2$, PROP$_N$(`DAD`(∞), `register`) is false. Thus we have Theorem 5.1.

THEOREM 5.1. *For $N \geq 2$,* PROP$_N$ *is not a property of all types.*

The type `DAD`(∞), which we have used to prove the above theorem, is nondeterministic. It is natural to ask if there are deterministic types T and T' for which PROP$_N(T, T')$ is false (for some N). Interestingly, the answer is negative. This follows from two recent results, as we explain below.

For all deterministic types T, T' and for all $N \geq 2$, if neither $\{T,$ `register`$\}$ nor $\{T',$ `register`$\}$ solves consensus for N processes, then $\{T, T',$ `register`$\}$ does not solve consensus for N processes [2, 16]. For all types T, if T is deterministic or T solves consensus for two processes, the following is true: if T does not solve consensus for N processes (for some N), then $\{T,$ `register`$\}$ does not solve consensus for N processes [1]. Together, these two results have the following straightforward implications.

1. If PROP$_N(T, T')$ is false, then at least one of T and T' is nondeterministic.
2. If PROP$_N(T, T')$ is false and one of T and T' is `register`, then the other type does not solve consensus for two processes.

Thus it is not surprising that the type `DAD`(∞) identified in this paper both is nondeterministic and does not solve consensus for two processes.

Appendix A. We prove that it is impossible to solve consensus for $k+1$ processes using only `DAD`(k) objects and registers. This impossibility result follows from a straightforward bivalency argument. The intuition behind why consensus is impossible for $k+1$ processes, but not for k processes, is as follows. As we have seen, a `DAD`(k) object supports two kinds of operations: `op` and `give-decision`. The operation `op`(i) does not return any useful information to the invoking process. This is due to the fact that the response of `op`(i) is always *ack*. The operation `give-decision` does return useful information but only to the first k invocations of the operation. Thereafter, its response is nondeterministic and hence is not helpful. Thus k processes may gain useful information from a `DAD`(k) object but $k+1$ processes cannot. We now proceed to give a formal proof of impossibility.

Let `det-DAD`(k) be the type obtained by replacing every occurrence of "choose(S)" in Figure 1 by the function "min(S)." (min(S) is the minimum element in set S.) Thus `det-DAD`(k) is a deterministic restriction of `DAD`(k). We prove below that `det-DAD`(k) objects and registers do not suffice to solve consensus for $k+1$ processes. This trivially implies that `DAD`(k) objects and registers do not suffice to solve consensus for $k+1$ processes.

As mentioned, the proof uses a simple bivalency argument. Since bivalency arguments are standard, our definitions and the proof are informal. A configuration C of a consensus protocol is *v-valent* (for $v \in \{0, 1\}$) if there is no execution from C in which $\overline{v}$ is decided upon by some process. In other words, once the protocol is in configuration C, no matter how processes are scheduled, no process decides $\overline{v}$. A configuration is *monovalent* if it is either 0-valent or 1-valent. A configuration is *bivalent* if it is not monovalent. If E is a finite execution of a consensus protocol $\mathcal{P}$ started in configuration C, $E(C)$ denotes the configuration at the end of the execution E.

LEMMA A.1. *For all $k \in \{2, 3, \ldots\}$,* $\{$`det-DAD`(k), `register`$\}$ *does not solve consensus for $k+1$ processes.*

Proof. Suppose that there is a consensus protocol $\mathcal{P} = (P_1, \ldots, P_{k+1}; O_1, O_2, \ldots)$, where each O_i is either a `det-DAD`(k) object or a register. Let C_0 be an initial configuration of $\mathcal{P}$ such that, for some processes P_l and P_m, $proposal_l = 0$ and $proposal_m = 1$.

When P_l runs by itself from C_0, the validity and the wait-freedom properties of $\mathcal{P}$ require that P_l decides $proposal_l = 0$. Similarly, when P_m runs by itself from C_0, it decides $proposal_m = 1$. Thus C_0 is bivalent. Let E be a finite execution from C_0 such that (1) $C_{crit} = E(C_0)$ is bivalent; and (2) for all P_i, if P_i takes a step from C_{crit}, the resulting configuration is monovalent. (If such E does not exist, it is easy to see that there is an infinite execution E' in which no process decides. Thus some process takes infinitely many steps in E' without deciding, contradicting the fact that $\mathcal{P}$ is a wait-free protocol.) Let S_v be the set of processes whose step from C_{crit} results in a v-valent configuration. Since C_{crit} is bivalent, neither S_0 nor S_1 is empty. Furthermore, $S_0 \cap S_1 = \emptyset$ and $|S_0 \cup S_1| = k + 1 \geq 3$ (since $k \geq 2$). Without loss of generality, assume that $|S_0| \geq 2$ and $|S_1| \geq 1$. In particular, let $S_0 = \{P_1^0, P_2^0, \ldots, P_r^0\}$ and $S_1 = \{P_1^1, P_2^1, \ldots, P_s^1\}$, where $r \geq 2$ and $s \geq 1$.

By a standard argument, the enabled step of every process in configuration C_{crit} must be on the same object O. Furthermore, again by a standard argument, O is not a register. Thus the enabled step of every process in configuration C_{crit} is on O, an object of type `det-DAD`(k). Let s_2^0 and s_1^1 denote the enabled steps of P_2^0 and P_1^1, respectively, in configuration C_{crit}. Consider the following scenarios $\mathtt{S_0}$ and $\mathtt{S_1}$, each starting from the configuration C_{crit}.

- In scenario $\mathtt{S_0}$, P_2^0 takes the step s_2^0. Then P_1^1 takes a step. Let D_0 be the resulting configuration. Clearly D_0 is a 0-valent configuration.
- In scenario $\mathtt{S_1}$, P_1^1 takes the step s_1^1. Then P_2^0 takes a step. Let D_1 be the resulting configuration. Clearly D_1 is a 1-valent configuration.

Processes P_2^0 and P_1^1 have to distinguish scenario $\mathtt{S_0}$ from scenario $\mathtt{S_1}$, since they must decide 0 in (every extension of) $\mathtt{S_0}$, and decide 1 in (every extension of) $\mathtt{S_1}$. Observe that unless the operation applied by P_2^0 (resp., P_1^1) in step s_2^0 (resp., s_1^1) is a `give-decision` operation, it must eventually apply a `give-decision` operation on O in order to distinguish $\mathtt{S_0}$ from $\mathtt{S_1}$. Thus we extend scenarios $\mathtt{S_0}$ and $\mathtt{S_1}$, currently in configurations D_0 and D_1, respectively, as follows.

1. First, if the operation applied by P_2^0 on O in step s_2^0 is not a `give-decision` operation, run P_2^0 (in both scenarios) exactly until P_2^0 completes a step in which it applies a `give-decision` operation on O.
2. Then if the operation applied by P_1^1 on O in step s_1^1 is not a `give-decision` operation, run P_1^1 (in both scenarios) exactly until P_1^1 completes a step in which it applies a `give-decision` operation on O.

A process $P \in \{P_1, \ldots, P_{k+1}\} - \{P_1^0, P_2^0, P_1^1\}$ has to distinguish scenario $\mathtt{S_0}$ from scenario $\mathtt{S_1}$, since P must decide 0 in (every extension of) $\mathtt{S_0}$, and decide 1 in (every extension of) $\mathtt{S_1}$. Observe, however, that P cannot distinguish $\mathtt{S_0}$ from $\mathtt{S_1}$ until it applies a `give-decision` operation on O. Thus we extend scenarios $\mathtt{S_0}$ and $\mathtt{S_1}$ as follows.

- For each $P \in \{P_1, \ldots, P_{k+1}\} - \{P_1^0, P_2^0, P_1^1\}$, run P (in both scenarios) exactly until P completes a step in which it applies a `give-decision` operation on O.

We make the following observations: (1) the process P_1^0 is in the same state in scenarios $\mathtt{S_0}$ and $\mathtt{S_1}$; (2) every object except O is in the same state in $\mathtt{S_0}$ and $\mathtt{S_1}$; (3) in both $\mathtt{S_0}$ and $\mathtt{S_1}$, a `give-decision` operation is applied on O at least k times (once by each process in $\{P_1, \ldots, P_{k+1}\} - \{P_1^0\}$). The third observation, together with the

specification of det-DAD(k), implies that every subsequent give-decision operation on O returns 0 in either scenario. Extend scenarios $\mathtt{S}_0$ and $\mathtt{S}_1$ by letting P_1^0 run by itself. By the above observations, P_1^0 cannot distinguish $\mathtt{S}_0$ from $\mathtt{S}_1$, no matter how many steps it takes. Yet it must decide 0 in $\mathtt{S}_0$, and 1 in $\mathtt{S}_1$. This is impossible. Hence the lemma is proved. □

Acknowledgment. I am grateful to an anonymous referee for providing a long list of comments that helped improve the presentation.

REFERENCES

[1] R. Bazzi, G. Neiger, and G. Peterson, *On the use of registers in achieving wait-free consensus*, in Proc. 13th Annual ACM Symposium on Principles of Distributed Computing, 1994, pp. 354–362.

[2] E. Borowsky, E. Gafni, and Y. Afek, *Consensus power makes (some) sense*, in Proc. 13th Annual ACM Symposium on Principles of Distributed Computing, August 1994, pp. 363–372.

[3] T. Chandra, V. Hadzilacos, P. Jayanti, and S. Toueg, *Wait-freedom vs. t-resiliency and the robustness of wait-free hierarchies*, in Proc. 13th Annual ACM Symposium on Principles of Distributed Computing, 1994, pp. 334–343.

[4] B. Chor, A. Israeli, and M. Li, *Wait-free consensus using asynchronous hardware*, SIAM J. Comput., 23 (1994), pp. 701–712.

[5] R. Cori and S. Moran, *Exotic behaviour of consensus numbers*, in Proc. 8th Workshop on Distributed Algorithms, Terschelling, The Netherlands, September–October 1994, pp. 101–115; Lecture Notes in Computer Science 857, Springer-Verlag, New York.

[6] P. Courtois, F. Heymans, and D. Parnas, *Concurrent control with readers and writers*, Comm. Assoc. Comput. Mach., 14 (1971), pp. 667–668.

[7] D. Dolev, C. Dwork, and L. Stockmeyer, *On the minimal synchronism needed for distributed consensus*, J. Assoc. Comput. Mach., 34 (1987), pp. 77–97.

[8] M. Fischer, N. Lynch, and M. Paterson, *Impossibility of distributed consensus with one faulty process*, J. Assoc. Comput. Mach., 32 (1985), pp. 374–382.

[9] M. Herlihy, *Wait-free synchronization*, ACM Trans. Programming Languages and Systems, 13 (1991), pp. 124–149.

[10] M. Herlihy and J. Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Trans. Programming Languages and Systems, 12 (1990), pp. 463–492.

[11] P. Jayanti, *Robust wait-free hierarchies*, J. Assoc. Comput. Mach., 44 (1997), pp. 592–614.

[12] J. Kleinberg and S. Mullainathan, *Resource bounds and combinations of consensus objects*, in Proc. 12th Annual ACM Symposium on Principles of Distributed Computing, August 1993, pp. 133–143.

[13] L. Lamport, *Concurrent reading and writing*, Comm. Assoc. Comput. Mach., 20 (1977), pp. 806–811.

[14] M. Loui and H. Abu-Amara, *Memory requirements for agreement among unreliable asynchronous processes*, Adv. Comput. Res., 4 (1987), pp. 163–183.

[15] N. Lynch and M. Tuttle, *An Introduction to Input/Output Automata*, Tech. Report MIT/LCS/TM-373, MIT Laboratory for Computer Science, MIT, Cambridge, MA, 1988.

[16] G. Peterson, R. Bazzi, and G. Neiger, *A gap theorem for consensus types*, in Proc. 13th Annual ACM Symposium on Principles of Distributed Computing, August 1994, pp. 344–353.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 798–815

A LOWER BOUND FOR INTEGER MULTIPLICATION WITH READ-ONCE BRANCHING PROGRAMS*

STEPHEN PONZIO†

Abstract. We prove that read-once branching programs computing integer multiplication require size $2^{\Omega(\sqrt{n})}$. This is the first nontrivial lower bound for multiplication on branching programs that are not oblivious. By the appropriate problem reductions, we obtain the same lower bound for other arithmetic functions.

Key words. multiplication, read-once, branching programs, BDD, verification

AMS subject classifications. 68Q05, 68Q25, 68M15

PII. S0097539795290349

1. Introduction and background.

It is well known that many functions, some of them very simple, cannot be computed by read-once branching programs of polynomial size [We88, Za84, Du85, We87, BHST87, Ju88, Kr88]. Interest in whether integer multiplication can be so computed has been created by recent developments in the field of digital design and hardware verification.

1.1. Hardware verification and branching programs.

The central problem of verification is to check whether a combinational hardware circuit has been correctly designed. One approach to verification often employed today is to independently convert both the circuit description and the function specification to a common intermediate representation and then test whether the two representations are equivalent (e.g., [We94]). The use of restricted branching programs as the intermediate representation has made this approach feasible and very popular—several software packages are available for implementing this very strategy [Kr94, Br92].

DEFINITION 1. *A* branching program *is a directed acyclic graph with a distinguished root node and two sink nodes. The sink nodes are labeled* 1 *and* 0 *and each nonsink node is labeled with an input variable* x_i, $i \in \{1, \ldots, n\}$, *and has two outgoing edges labeled* 0 *and* 1.

A branching program computes a Boolean function $f : \{0,1\}^n \to \{0,1\}$ in the natural manner: each assignment of Boolean values to the variables x_i defines a unique path through the graph from the root to one of the sinks; the label of that sink defines the value of the function on that input. The *size* of a branching program is its number of nodes.

The hardware circuit to be verified is assumed to be an ordinary combinational single-output circuit, built up from a standard basis of Boolean functions such as $\{\wedge, \vee, \neg\}$. The typical algorithm for constructing the intermediate representation from the circuit is to work bottom-up through the circuit, from the inputs to the output, combining the representations appropriately at each gate. Thus, the algorithm need only compute a representation for $f \wedge g$, $f \vee g$, and $\neg f$, when given representations for f and g. In the literature, these are called the "synthesis operations."

*Received by the editors August 14, 1995; accepted for publication (in revised form) October 23, 1996; published electronically September 14, 1998.

http://www.siam.org/journals/sicomp/28-3/29034.html

†Institute of Computer Science, Hebrew University, Jerusalem 91904, Israel. The work of this author was done at the MIT Laboratory for Computer Science.

1.2. Ordered binary decision diagrams (OBDDs) and multiplication. Because it is NP-complete to determine whether two general branching programs are equivalent, the intermediate representation is chosen to be a restricted class of branching programs for which both the synthesis operations and the equivalence test are tractable.[1] The chosen class, which has led to the success of this verification technique, is *oblivious read-once* branching programs also known as *ordered binary decision diagrams*, or *OBDDs*.

DEFINITION 2. *A branching program is* read-once *if on every path from the source to a sink, each variable appears at most once as the label of a vertex.*

DEFINITION 3. *A branching program is* oblivious *if on every path from the source to a sink, the variables appear in the same order.*

Thus, any oblivious branching program may be leveled so that all nodes at a given level are labelled with the same variable. An OBDD will have n levels, with a variable ordering that is a permutation of the variables.

It is easy to verify that two OBDDs obeying the same ordering of the variables are easily tested for equivalence: Construct an OBDD to compute their exclusive-or using standard finite automata product constructions for conjunction and disjunction; then test for a path from source to sink. (In general, there do not exist polynomial-size constructions for conjunction and disjunction if the two OBDDs do not obey the same ordering—see section 1.4.)

Although OBDDs are easily manipulated, they are clearly a very weak model of computation. For the purposes of hardware verification, however, they are usually powerful enough: OBDDs can compute in polynomial size integer addition, symmetric Boolean functions, and many of the benchmark functions [BF85] used by the verification community. But multiplication is a very important exception—for this function Byrant [Br91] proved that exponential size is required.

DEFINITION 4. Integer multiplication *is the Boolean function* $\mathbf{MULT} : \{0,1\}^{2n} \to \{0,1\}$ *that computes the middle bit in the product of two* n*-bit integers. That is,* $\mathbf{MULT}(x,y) = z_{n-1}$ *where* $x = x_{n-1}\cdots x_0$ *and* $y = y_{n-1}\cdots y_0$ *and* $xy = z = z_{2n-1}\cdots z_0$.

(The middle bit is the "hardest" bit, in the sense that if it can be computed by read-once branching programs (or circuits, etc.) of size $s(n)$, then any other bit can be computed with size at most $s(2n)$.) This is a serious limitation to the usefulness of OBDDs, since the hardware to be tested typically contains circuits that perform multiplication. Today, the largest multipliers that can be checked using this method have 12-bit inputs; ideally, circuit designers would like to check multipliers of 32 or even 64 bits.

1.3. Other oblivious models: Extensions to OBDDs. Thus, despite the success of this approach, there has also been great effort expended to find another model that is likewise manipulated but with greater computational power [SDG94, SW95, e.g.]. For example, the various extensions to OBDDs that have been considered include

- "k-OBDDs," where the variable ordering is a single permutation repeated k times consecutively [BSSW93, BHR95, Kr91];

[1] Of course efficient (polynomial time) algorithms for the individual synthesis operations do not imply that the resulting bottom-up algorithm for computing a representation is efficient. Despite this problem, researchers have been content with the bottom-up algorithm as long as each synthesis operation can be performed efficiently.

- "k-IBDDs," where the variable ordering is k consecutive permutations, possibly different [JABFA92, BSSW95];
- OBDDs with various kinds of nondeterministic branching nodes ($\vee$-nodes, $\wedge$-nodes, $\oplus$-nodes) [Me89, SDG94, Ge94, and others].

Recently proposed alternative models include "graph-driven BDDs" [SW95] and "binary moment diagrams" [BC94]. The latter are not branching programs and do not *compute* a function, but they do allow polynomial-size representation of multiplication. Also, in [AM88] lower bounds are proved for any oblivious programs of linear length, regardless of the order in which variables are read.

From the proof of Bryant's lower bound for OBDDs [Br91], it follows by a simple communication complexity argument that **Mult** cannot be computed in polynomial-size by k-OBDDs [Kr91, BSSW93] or the various nondeterministic OBDDs [Ge94]. Incorporating results from [AM88], [Ge94] extends the lower bound to arbitrary linear-length oblivious programs. Indeed, all of these oblivious models have been found too weak to compute **Mult** in polynomial size. It is therefore natural to consider *nonoblivious* programs, the simplest of these being read-once programs.

1.4. Nonoblivious programs. Unfortunately, (nonoblivious) read-once programs are not as easily manipulated as OBDDs. It is not known how to test equivalence in polynomial time, though there is a randomized (co-RP) algorithm [BCW80]. (There is also a deterministic algorithm to test the equivalence of an OBDD and a read-once program [FHS78].) Moreover, the synthesis operations are *provably intractable*—there exist functions f and g that each have polynomial-size read-once programs but whose conjunction $f \wedge g$ requires exponential-size read-once programs. For example, determining whether a $0,1$-matrix is a permutation matrix requires exponential-size read-once programs (even nondeterministic) [KMW91], whereas the rowwise and columnwise criteria (that each has one 1) are each computable with small OBDDs. Despite their relative recalcitrance, read-once programs have been considered by some researchers for possible use in hardware verification [GM94]. Very little was known about the complexity of multiplication.

There has been great success in proving lower bounds on the size of read-once programs, even for some very simple functions [Ma76, Du85, Za84, We87, SS93]. For example, it was proved in [Za84] (see also [We87]) that determining whether a graph on n nodes is an $n/2$-clique (with no further edges) requires size $2^{\Omega(n)}$. Until the time of this writing, the only asymptotically optimal lower bound was found in [BHST87], which proves a bound of $2^{\Omega(n^2)}$ for computing the parity of the number of triangles in a graph on n nodes. Very recently, Savicky and Zak [SZ96] proved a lower bound of $2^{n-3\sqrt{n}}$, the best to date. Exponential lower bounds for explicit functions have also been proved for nondeterministic read-once branching programs [Ju89, KMW91, BRS93]. Lower bounds for read-k-times programs (where each variable appears at most k times on each path)[2] are proved in [Ok91, BRS93, Ju92].

1.5. The decision problem. Although it is not directly related to the issue of verification, another Boolean function that has been considered is the decision problem $\textbf{Dmult}(x, y, z) = 1$ if $xy = z$. Note that it is not readily apparent which problem is "harder", **Mult** or **Dmult**: On the one hand, **Dmult** seems to require

[2]These are often called "syntactic" read-k-times, an apparently (though not proven) more severe restriction than "semantic" read-k-times where the limitation applies only to paths from source to sink that are traversed by some input—that is, paths that contain no contradictory literals along them. No superpolynomial bounds are known for semantic read-k-times programs for any $k \geq 2$.

practically computing all the bits of xy; however, an algorithm for **DMULT** has the advantage of inspecting all the bits of z, the putative product. (An easy reduction in [FSS84] shows **MULT** $\notin$ AC^0, but is was not until [Bu92] that **DMULT** $\notin$ AC^0 was proved.)

A simple argument [We94] shows that computing **DMULT** with read-once programs is as hard as factoring: To factor a given integer n, instantiate it as z in the read-once branching program and construct a factor by instantiating the bits of x one at a time, maintaining the satisfiability (source-to-sink connectivity) of the branching program. Jukna [Ju94] proves a lower bound of $2^{n^{1/4}/k^{2k}}$ for **DMULT** on nondeterministic read-k-times branching programs, but this does not yield results for **MULT**.

1.6. Our results. In this paper, we prove that any read-once branching programs for **MULT** have size $2^{\Omega(\sqrt{n})}$. This is the first superpolynomial lower bound for multiplication on nonoblivious branching programs.

We begin by describing in section 2 a paradigm for read-once lower bounds, recognized and distilled by Simon and Szegedy [SS93], in Lemma 1. For ease of presentation we first prove a lower bound of $2^{\Omega(\sqrt[3]{n})}$ in section 3 and then extend the proof to achieve $2^{\Omega(\sqrt{n})}$ in section 4. In section 5, we define *read-once reductions*[3] in order to deduce similar lower bounds for other arithmetic functions.

2. A paradigm for read-once lower bounds. Let f be a Boolean function, $f : \{0,1\}^n \to \{0,1\}$, and let $X = \{x_0, \ldots, x_{n-1}\}$ be its n binary input variables. Let $\mathcal{F}$ be a filter on X. (That is, $\mathcal{F} \subseteq 2^X$ and $\mathcal{F}$ is closed upward—if $S \in \mathcal{F}$, then all supersets of S are in $\mathcal{F}$.)[4] The filter $\mathcal{F}$ gives us a way of partitioning 2^X into "large" sets (in $\mathcal{F}$) and "small" sets (not in $\mathcal{F}$). A subset $B \subset X$ is said to be in the *boundary* of $\mathcal{F}$ if $B \notin \mathcal{F}$ but $B \cup \{x_i\} \in \mathcal{F}$ for some x_i. By setting the values of $\overline{B} = X \setminus B$, we naturally induce a function on B, the unset bits of X. The lemma is stated below in the form we will need it; it appears in [SS93] in slightly more generalized form.

LEMMA 1 (see [SS93]). *If for any B in the boundary of $\mathcal{F}$ at most $2^{|\overline{B}|}/L$ settings to $\overline{B}$ induce the same subfunction on B, then any read-once branching program computing f has size at least L.*

For completeness, we now provide a proof of this lemma.

Proof. The idea is that $\mathcal{F}$ defines a "frontier" of edges in the branching program—a cut containing exactly one edge from each source-to-sink path—in which every edge allows only a fraction $1/L$ of the inputs in $\{0,1\}^n$ to pass through it. Since the path of every input passes through some frontier edge, there must be at least L such edges. Having fan-out 2 and only one root, the program also has at least L nodes (the frontier edges lead to the leaves of an embedded binary tree which has $L-1$ distinct internal nodes, not counting the two sinks of the program).

In order to define the frontier, we first associate with each node of the program the set of variables appearing in the subprogram rooted there—that is, those variables appearing on nodes that are reachable from the given node. Clearly, moving down any path, the variable-sets of later nodes are subsets of the variable-sets of earlier nodes. The frontier is defined to be those edges leading from nodes with "large" sets of variables to nodes with "small" sets. "Large" sets are those that are in the filter $\mathcal{F}$ and "small" are not. Clearly there is exactly one frontier edge on each source-to-sink path, since the root has the variable-set $X \in \mathcal{F}$ (assuming f depends on all variables)

[3] Similar reductions have recently been considered also in [BW95].

[4] This differs from the usual definition of *filter* for infinite sets because we do not require that $(S \in \mathcal{F}) \wedge (T \in \mathcal{F}) \Longrightarrow (S \cap T) \in \mathcal{F}$.

and the sinks have the variable-set $\emptyset \notin F$ (for proper filters $\mathcal{F}$). Each frontier edge is thus naturally associated with a set $B \subset X$ in the boundary of $\mathcal{F}$.

Suppose boundary set B is associated with a given frontier edge. Because the program is read-once, these variables do not appear on any path from the root to this edge. In fact, the inputs $x \in \{0,1\}^n$ that reach this edge are characterized exactly by their settings to $\overline{B}$. Each setting to $\overline{B}$ that reaches this edge clearly induces the same subfunction on B, as defined by the subprogram rooted there. Since at most $2^{|\overline{B}|}/L$ settings to $\overline{B}$ give the same subfunction on B, at most $(2^{|\overline{B}|}/L) \cdot 2^{|B|} = 2^n/L$ inputs in $\{0,1\}^n$ may pass through this frontier edge. The lower bound follows. □

3. A lower bound of $2^{\Omega(\sqrt[3]{n})}$. In this section, we prove a weaker bound with an easier proof that includes the main ideas.

THEOREM 1. *Read-once branching programs for* **MULT** *require size* $2^{\Omega(\sqrt[3]{n})}$.

Proof. Let $m = \sqrt[3]{n}/4$ and let X and Y denote the sets of variables $X = \{x_0, \ldots, x_{n-1}\}$ and $Y = \{y_0, \ldots, y_{n-1}\}$. Define the filter

$$\mathcal{F} = \{V \subset (X \cup Y) : |V \cap X| > n - m \text{ and } |V \cap Y| > n - m\}.$$

The resulting "frontier" of the branching program (as defined in Lemma 1) roughly speaking marks the threshold where at most m bits of X and at most m bits of Y have been read. (In order for this notion to be strictly correct, "have been read" must be interpreted to mean "appear on any path from the root.")

We will show that for any B in the boundary of $\mathcal{F}$, at most $2^{|\overline{B}|-m}$ settings to $\overline{B}$ give the same subfunction on B. By Lemma 1, this gives the desired lower bound of 2^m. Fix any B in the boundary of $\mathcal{F}$ and let $S = \overline{B}$. Think of S as being the variables already read by the branching program. Since B is in the boundary of $\mathcal{F}$, either $|S \cap X| = m$ or $|S \cap Y| = m$ (but not both). We will show that there is a subset $S' \subset S$ of size at least m such that if two settings to S differ on S' then they induce different subfunctions on $\overline{S} = B$. Thus at most $2^{|S|-m}$ settings to $S = \overline{B}$ induce the same subfunction on $\overline{S} = B$, as desired. We will show that the two subfunctions are different by explicitly demonstrating a single setting to the bits of $\overline{S}$ where the induced subfunctions of **MULT** differ.

Suppose without loss of generality that $|S \cap X| = m$ (and therefore $|S \cap Y| < m$). Let $i \in \{0, \ldots, n-1\}$ be the smallest index such that $y_i \notin S$. Let

$$S' = \{y_0, \ldots, y_{i-1}\} \cup \left(S \cap \{x_0, \ldots, x_{n-1-i}\}\right).$$

Note that because $\{y_0, \ldots, y_{i-1}\} \subseteq S$ and $|S \cap X| = m$, we have $|S'| \geq m$.

Let us adopt the following notation for the integers obtained from partial settings to the variables. For a setting α to $W \subseteq X \cup Y$ (i.e., $\alpha : W \to \{0,1\}$), let x_α denote the integer that is represented in binary when the variables of $X \cap W$ have the value given by α and the variables of $X \cap \overline{W}$ are each 0. Define y_α similarly. For a single variable $z \notin W$, let "$\alpha + z$" denote the setting to $W \cup \{z\}$ that further sets $z = 1$. For two settings α and τ to disjoint subsets W and V, let "$\alpha \cup \tau$" denote the setting equal to α on W and to τ on V. Finally, let $(x)_i$ denote the ith bit in the binary representation of integer x, so $x = \sum_{i=0}^{n-1} (x)_i \, 2^i$.

Let α and β be two settings to S that differ on some bit in S'. Our goal is thus to find a setting τ to the bits of $\overline{S}$ so that $(x_{\alpha\cup\tau} y_{\alpha\cup\tau})_{n-1} \neq (x_{\beta\cup\tau} y_{\beta\cup\tau})_{n-1}$.

We proceed in two stages, according to Lemmas 2 and 3. First we ensure, by setting to 1 (if necessary) a single variable z of $\overline{S}$, that the two products $x_{\alpha+z} y_{\alpha+z}$

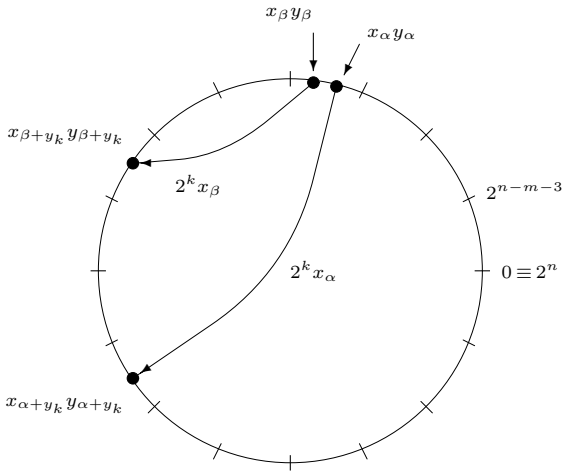


FIG. 1. *The integers modulo 2^n. In order for $x_{\beta+y_k} y_{\beta+y_k}$ and $x_{\alpha+y_k} y_{\alpha+y_k}$ to fall into different segments, we must choose k so that $2^k(x_\alpha - x_\beta)$ has large magnitude.*

and $x_{\beta+z}y_{\beta+z}$ differ in a "high-order" bit—a bit position in the range $[n-m-3, n-1]$ (we aren't concerned with bit positions higher than the middle). In the second stage, we set to 1 a pair of variables of $\overline{S}$, one in X and one in Y, so that the resulting product differs in a *higher* high-order bit position. We iterate this second stage, repeatedly setting a pair of variables until the resulting products differ in bit position $n-1$. It follows that α and β induce different subfunctions on $\overline{S}$—the subfunctions differ when $\overline{S}$ has z and the pairs from the second stage all set to 1 and the remaining bits of $\overline{S}$ set to 0.

LEMMA 2. *If for all $i \in [n-m-3, n-1]$ we have $(x_\alpha y_\alpha)_i = (x_\beta y_\beta)_i$, then there is a single variable $z \in \overline{S}$ such that*

$$(x_{\alpha+z}y_{\alpha+z})_i \neq (x_{\beta+z}y_{\beta+z})_i$$

for some $i \in [n-m-3, n-1]$.

LEMMA 3. *Let $T \subset X \cup Y$, and α and β be two settings to T. Let d be the greatest index in $[0, n-2]$ such that $(x_\alpha y_\alpha)_d \neq (x_\beta y_\beta)_d$. If $d \geq n-m-3$ and $\max(|T \cap X|, |T \cap Y|) = t \leq 3m$, then there are two variables, $x_u \in X \cap \overline{T}$ and $y_v \in Y \cap \overline{T}$, such that*

$$(x_{\alpha'}y_{\alpha'})_{d+1} \neq (x_{\beta'}y_{\beta'})_{d+1}$$

where $\alpha' = \alpha + x_u + y_v$ and $\beta' = \beta + x_u + y_v$.

Theorem 1 now follows from these lemmas as outlined above. Notice that Lemma 3 is first applied with $t \leq m+1$, and since we must apply Lemma 3 at most $m+3$ times, each time setting one more variable of X and Y, we maintain $t \leq 2m+4 \leq 3m$ as required. □

We now give the proofs of Lemmas 2 and 3.

Proof of Lemma 2. The settings α and β differ on $S' \subseteq S$; suppose first that they differ in a bit of $S' \cap X$.

The proof is most easily explained by picturing the integers modulo 2^n on a circle. Partition the circle into 2^{m+3} equal-sized segments according to the values

of the $m+3$ highest bits, so each segment contains 2^{n-m-3} consecutive integers, as depicted in Figure 1. The hypothesis of the lemma is that $x_\alpha y_\alpha$ and $x_\beta y_\beta$ fall into the same segment. If we set bit $y_k \in \overline{S} \cap Y$ to 1, we obtain the products $x_{\alpha+y_k} y_{\alpha+y_k} = x_\alpha y_\alpha + x_\alpha 2^k$ and $x_{\beta+y_k} y_{\beta+y_k} = x_\beta y_\beta + x_\beta 2^k$. The product $x_{\alpha+y_k} y_{\alpha+y_k}$ is obtained by a translation of $2^k x_\alpha$ along the circle from $x_\alpha y_\alpha$, and $x_{\beta+y_k} y_{\beta+y_k}$ is obtained by a translation of $2^k x_\beta$ from $x_\beta y_\beta$. If, modulo 2^n, their difference $2^k(x_\alpha - x_\beta)$ is at least 2^{n-m-2}, or two segments long, and at most $2^n - 2^{n-m-2}$, or "negative two" segments long, then it is clear that the translates $x_{\alpha+y_k} y_{\alpha+y_k}$ and $x_{\beta+y_k} y_{\beta+y_k}$ fall into different segments. It follows that the products $x_{\alpha+y_k} y_{\alpha+y_k}$ and $x_{\beta+y_k} y_{\beta+y_k}$ differ in a high-order bit position.

It only remains to show how to choose $y_k \in \overline{S} \cap Y$ so that $2^{n-m-2} \leq 2^k(x_\alpha - x_\beta) \leq 2^n - 2^{n-m-2}$ modulo 2^n. Let $\overline{x} = x_\alpha - x_\beta$. It is useful now to think in terms of the table generated by the usual grade-school algorithm for multiplying $\overline{x}$ by y, as shown in Figure 2.

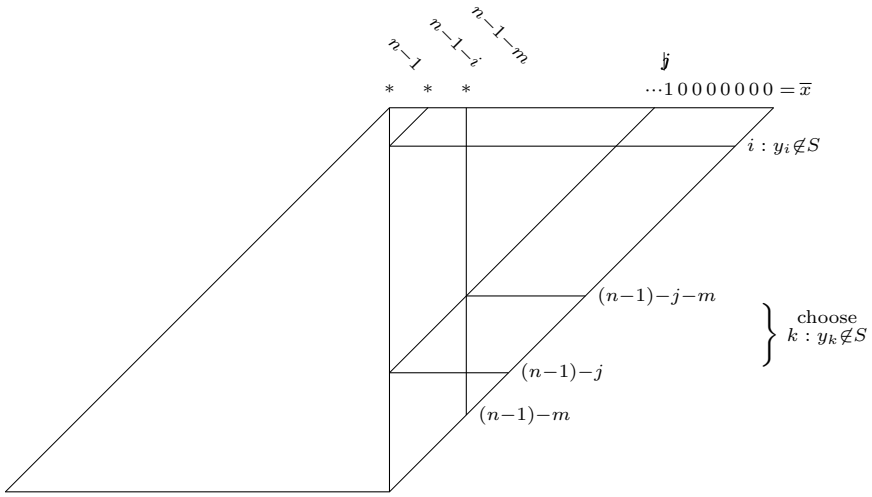


FIG. 2. *The table generated by the grade-school algorithm for multiplying $\overline{x} = x_\alpha - x_\beta$ by y. We choose a bit y_k to set to 1 so that the least significant 1 in $\overline{x}$ is shifted into a "high-order" bit position.*

In this table, the rows are the partial products, indexed by $y_0, \ldots, y_{n-1}$. The diagonals are indexed by $\overline{x}_{n-1}, \ldots, \overline{x}_0$. Since α and β differ in a bit of $S' \cap X \subseteq \{x_0, \ldots, x_{n-1-i}\}$, the difference $\overline{x} = x_\alpha - x_\beta$ must have a 1 somewhere in the range of bit positions $[0, n-1-i]$. Let j be the position of the least significant 1 in $\overline{x}$, so that either there is a 0 in position $j-1$, or $j = 0$. We now choose any variable of $\overline{S} \cap Y$ with index k in the range $[(n-1)-j-m, (n-1)-j]$. This range must contain a variable $y_k \in \overline{S} \cap Y$ because if $(n-1)-j-m \geq 0$, the range has at least $m+1$ elements while $|S \cap Y| < m$; if, however, $(n-1)-j-m < 0$, we may choose $k = i$ (by definition $y_i \notin S$), which lies in the range $[0, n-1-j]$ since $j \leq n-i-1$. This ensures that $2^k \overline{x}$ has a 1 in position $j+k$ and a 0 in position $j+k-1$, where $n-1-m \leq j+k \leq n-1$. It follows that modulo 2^n, we have $2^{n-m-1} \leq 2^k \overline{x} \leq 2^n - 1 - 2^{n-m-2}$ (the upper bound attained if all bits except bit $j+k-1$ are 1's and $j+k = n-1-m$). This satisfies the desired bounds.

If α and β differ in a bit of $S' \cap Y \subseteq \{y_0, \ldots, y_{i-1}\}$ the proof is essentially the same. We have to choose $x_k \in \overline{S} \cap X$ so that $2^{n-m-2} \leq 2^k(y_\alpha - y_\beta) \leq 2^n - 2^{n-m-2}$ modulo 2^n. In this case, we know $\overline{y} = y_\alpha - y_\beta$ has a 1 in the range $[0, i-1]$. Again

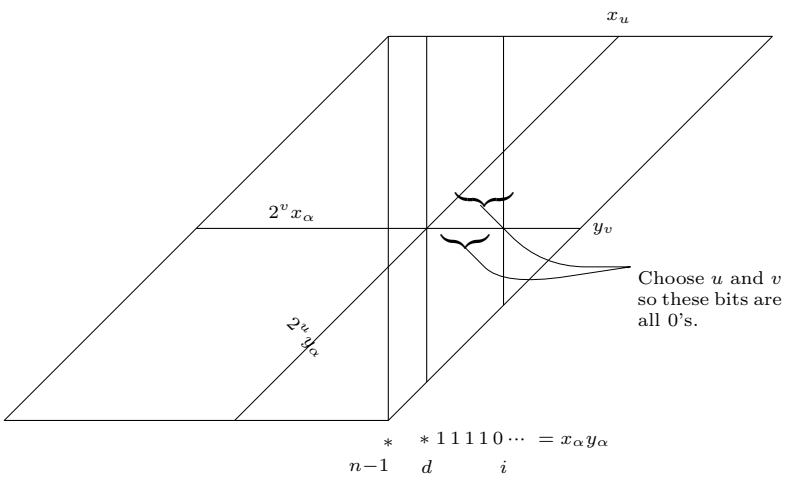


FIG. 3. *In Lemma* 3*, we choose* x_u *and* y_v *to set to* 1 *so that* $u+v=d$ *and also so that the products* $2^u y_\alpha$ *and* $2^v x_\alpha$ *have* 0*'s in bit positions* $d-1,\ldots,i-1$ *so that when added to* $x_\alpha y_\alpha + 2^d$*, they do not cause a carry to propagate into position* $d+1$*.*

letting j be the least significant 1 of $\overline{y}$ in this range, we simply choose k anywhere in the range $[(n-1)-j-m, n-1-j]$. Now for sure $(n-1)-j-m \geq 0$ and the range always has $m+1$ elements, since $j \leq i-1 \leq m = \sqrt[3]{n}/4$. It follows as before that $2^k\overline{y}$ satisfies the desired inequality. This completes the proof. □

We restate Lemma 3 for convenience.

LEMMA 3. *Let* $T \subset X \cup Y$*, and* α *and* β *be two settings to* T*. Let* d *be the greatest index in* $[0, n-2]$ *such that* $(x_\alpha y_\alpha)_d \neq (x_\beta y_\beta)_d$*. If* $d \geq n-m-3$ *and* $\max(|T\cap X|, |T\cap Y|) = t \leq 3m$*, then there are two variables* $x_u \in X \cap \overline{T}$ *and* $y_v \in Y \cap \overline{T}$ *such that*

$$(x_{\alpha'} y_{\alpha'})_{d+1} \neq (x_{\beta'} y_{\beta'})_{d+1}$$

where $\alpha' = \alpha + x_u + y_v$ *and* $\beta' = \beta + x_u + y_v$*.*

Proof of Lemma 3. We will consider all pairs of variables (x_u, y_v) such that $u+v=d$. We want $(x_{\alpha'} y_{\alpha'})_{d+1} \neq (x_{\beta'} y_{\beta'})_{d+1}$, where

$$\begin{aligned} x_{\alpha'} y_{\alpha'} &= (x_\alpha + 2^u)(y_\alpha + 2^v) \\ &= \left(x_\alpha y_\alpha + 2^d\right) + \left(2^v x_\alpha + 2^u y_\alpha\right), \\ \text{and } \; x_{\beta'} y_{\beta'} &= (x_\beta + 2^u)(y_\beta + 2^v) \\ &= \left(x_\beta y_\beta + 2^d\right) + \left(2^v x_\beta + 2^u y_\beta\right). \end{aligned}$$

Since d is the highest bit in which $x_\alpha y_\alpha$ and $x_\beta y_\beta$ differ, clearly $\left(x_\alpha y_\alpha + 2^d\right)_{d+1} \neq \left(x_\beta y_\beta + 2^d\right)_{d+1}$. We will choose u and v so that the addition of the cross terms $2^v x_\alpha + 2^u y_\alpha$ to $x_\alpha y_\alpha + 2^d$ does not affect bits d or $d+1$ of $x_\alpha y_\alpha + 2^d$ (and similarly for β). In order to do this, we choose u and v so that the cross terms have 0's in bit positions d and $d+1$ and, furthermore, in the addition of the two integers, there is no carry bit into position d.

To accomplish this, we first find the largest bit position i less than d where $x_\alpha y_\alpha$ has a 0 (so positions $i+1$ through $d-1$ are all 1's). We will choose u and v so that $2^v x_\alpha$ and $2^u y_\alpha$ each has 0's in positions $i-1$ through $d+1$. It follows that their sum

then has 0's in positions i through $d+1$, and so, when added to $x_\alpha y_\alpha + 2^d$ which has a 0 in position i, causes no carry into any position $i+1$ through d (see Figure 3). We will choose u and v so that the same conditions hold for β as well.

A simple counting argument now shows that there exist u and v as desired. First, we claim that $x_\alpha y_\alpha$ (and $x_\beta y_\beta$) has 1's in at most t^2 bit positions, so that $i \geq (d-1) - t^2$. In general, if the binary representations of integers p and q have $w(p)$ and $w(q)$ 1's in them, respectively, then clearly $p+q$ has at most $w(p)+w(q)$ 1's in it. Recall α sets at most t bits in X or Y. We may therefore view $x_\alpha y_\alpha$ as the addition of at most t shifts of x_α, and the claim follows.

We require $(2^v x_\alpha)_j = (2^v x_\beta)_j = 0$ in at most t^2+4 positions j: $j = d+1,\ d,\ d-1,\ \ldots,\ i,\ i-1$. There are at most t bit positions in which either x_α or x_β has a 1, and for each such 1, there are at most t^2+4 "bad" values of $v \in [0, n-1]$ that shift the 1 to a position we require to be 0. Thus, x_α and x_β rule out at most $t(t^2+4)$ values of v. Furthermore, there are up to t variables of Y that are in T, making a total of $t(t^2+4)+t$ values of v that we may not choose. Similarly, a total of at most $t(t^2+4)+t$ values of u are ruled out by y_α, y_β, and T. The number of pairs (x_u, y_v) in which either x_u or y_v has been ruled out is thus at most

$$2(t^3+5t) \leq 2\left(27m^3+15m\right) \leq 2\left(\frac{27n}{64} + \frac{15\sqrt[3]{n}}{4}\right)$$

since $t \leq 3m$ and $m = \sqrt[3]{n}/4$. There are at least $d+1 \geq n-m-2$ pairs (x_u, y_v) such that $u+v=d$. Thus we retain at least

$$n - \frac{\sqrt[3]{n}}{4} - 2 - \left(\frac{54}{64}n + \frac{30}{4}\sqrt[3]{n}\right) = \Omega(n)$$

good pairs satisfying the desired requirements for x_u and y_v. □

4. Improving the bound to $2^{\Omega(\sqrt{n})}$. We can improve the lower bound to $2^{\Omega(\sqrt{n})}$ by analyzing more closely how we iterate Lemma 3 in the proof of the theorem. There we needed $m = O(\sqrt[3]{n})$, because in Lemma 3 we used $t^2 = O(m^2)$ as an upper bound on the number of consecutive 1's to the right of position d in $x_\alpha y_\alpha$ or $x_\beta y_\beta$. We then required 0's in these $O(m^2)$ positions in the cross terms $2^v x_\alpha + 2^u y_\alpha$ and $2^v x_\beta + 2^u y_\beta$. Since each of the $O(m)$ 1's in x_α may then rule out $O(m^2)$ values of v, we needed $O(m^3) < n$ in order not to rule out all values of v. In order to allow $m = O(\sqrt{n})$, we will reduce to $O(m)$ the number of positions in which we require 0's in the cross terms. For the rest of this section, we let $m = \sqrt{n}/3$.

Depending on exactly what $x_\alpha y_\alpha$ and $x_\beta y_\beta$ look like, we may not need to require more than a few 0's in the cross terms. For example, if $x_\alpha y_\alpha$ and $x_\beta y_\beta$ look like[5]

$$\begin{aligned} x_\alpha y_\alpha &= \cdots \underset{d}{1}0 \cdots \\ x_\beta y_\beta &= \cdots \underset{d}{0}0 \cdots \end{aligned},$$

then we need to require 0's in the cross terms in only four positions: $d+1$ through $d-2$. This is sufficient to ensure that the addition of $2^v x_\alpha + 2^u y_\alpha$ to $x_\alpha y_\alpha + 2^d$ does not generate a carry into position d and does not affect bits d or $d+1$ of $x_\alpha y_\alpha + 2^d$. The same holds for β and we get $(x_{\alpha'} y_{\alpha'})_{d+1} \neq (x_{\beta'} y_{\beta'})_{d+1}$. With only these four positions

[5]Here and henceforth, "$\cdots$" denotes an arbitrary string of 0's and 1's; thus $x_\alpha y_\alpha = \cdots \underset{d}{1}0 \cdots$ has a 1 in bit d, a 0 in bit $d-1$, and may have any values in other bit positions.

required to be 0's, the total number of v's ruled out by x_β and x_α is proportional to the number of 1's they contain, which is $O(m)$. Similarly, the cases

$$\begin{array}{l} x_\alpha y_\alpha = \cdots \underset{d}{1} 1 \cdots \\ x_\beta y_\beta = \cdots \underset{d}{0} 0 \cdots \end{array} \quad \text{and} \quad \begin{array}{l} x_\alpha y_\alpha = \cdots \underset{d}{1} 1 \cdots \\ x_\beta y_\beta = \cdots \underset{d}{0} 1 \cdots \end{array}$$

can be handled with only a few constraints by choosing $u + v = d - 1$ (this will be proved in Lemma 6). In fact, there is really only one case in which we need to require $(2^v x_\beta + 2^u y_\beta)$ or $(2^v x_\alpha + 2^u y_\alpha)$ to have many 0's, which follows.

DEFINITION 5. *Let d be the greatest index less than n in which $(x_\alpha y_\alpha)_d \neq (x_\beta y_\beta)_d$. We say that $x_\alpha y_\alpha$ and $x_\beta y_\beta$ are* k-bad *if $d \geq n - m - 4$ and the products look like*

$$\begin{array}{rcl} x_\alpha y_\alpha & = & \cdots \underset{d}{1} 0 \cdots\cdots \\ x_\beta y_\beta & = & \cdots \underset{d}{0} 1\,1\,1\,1\,\underset{\substack{\uparrow \\ n-m-6}}{1}\underbrace{1\,1\,1\,1\,1}_{k}\cdots \end{array}$$

or vice versa (exchanging α and β).

In this case, say $x_\beta y_\beta = \cdots \underset{d}{0} 1\,1\,1\,1\,\underset{\substack{\uparrow \\ n-m-6}}{1}\underbrace{1\,1\,1\,1\,1}_{k}\cdots$, we must require $2^v x_\beta + 2^u y_\beta$ to be 0 in the positions of each of these 1's in order to prevent a carry into position $d+1$ when we add it to $x_\beta y_\beta + 2^d$. In order to allow $m = O(\sqrt{n})$, we will ensure that the products are not k-bad for $k > m + 4$. Then the number of v's ruled out by each 1 of x_α and x_β is $2m + 10$, and as long as the number of 1's in x_α or x_β is $O(m)$, the total number of v's ruled out is $O(m^2)$.

We will first show that we may begin with products that differ in a high-order bit but are not 1-bad and then prove a version of Lemma 3 in which each application allows the "badness" to grow by at most 1.

LEMMA 5. *For any two settings α and β to S that differ on a bit of S', there are three (or fewer) variables $x_u, y_v, z \in \overline{S}$ ($x_u \in X$ and $y_v \in Y$) such that for $\alpha' = \alpha + x_u + y_v + z$ and $\beta' = \beta + x_u + y_v + z$, the products $x_{\alpha'} y_{\alpha'}$ and $x_{\beta'} y_{\beta'}$ differ in a high-order bit (in the range $[n - m - 4, n - 1]$) and, moreover, are not* 1*-bad.*

LEMMA 6. *Let $T \subset X \cup Y$, and let α and β be two settings to T. Let d be the greatest index in $[0, n-2]$ such that $(x_\alpha y_\alpha)_d \neq (x_\beta y_\beta)_d$. Suppose $d \geq n - m - 4$ and $\max(|T \cap X|, |T \cap Y|) = t \leq 2m + 5$ and also that $x_\alpha y_\alpha$ and $x_\beta y_\beta$ are not k-bad for some $k \leq m + 4$. Then there are two variables, $x_u, y_v \in \overline{T}$ ($x_u \in X$ and $y_v \in Y$), such that*

$$(x_{\alpha'} y_{\alpha'})_{d+1} \neq (x_{\beta'} y_{\beta'})_{d+1}$$

for $\alpha' = \alpha + x_u + y_v$ and $\beta' = \beta + x_u + y_v$, and, moreover, $x_{\alpha'} y_{\alpha'}$ and $x_{\beta'} y_{\beta'}$ are not $(k+1)$-bad.

We now have Theorem 2.

THEOREM 2. *Read-once branching programs for* **MULT** *require size $2^{\Omega(\sqrt{n})}$.*

Proof. The proof is exactly the same as the proof of Theorem 1 except for the lemmas. We start with products that differ in a high-order bit but are not 1-bad, as provided by Lemma 5. The number of variables in X or Y set in these products is at most $m + 2$. We obtain a difference in bit $n - 1$ by iterating Lemma 5 at most $m + 3$ times, each time setting at most one variable in X and in Y. This maintains $t \leq (m + 2) + (m + 3)$ and $k \leq 1 + (m + 3)$ as required. □

We now give the proofs of Lemmas 5 and 6.

Proof of Lemma 5. Either $x_\alpha y_\alpha$ and $x_\beta y_\beta$ differ (modulo 2^n) by at least 2^{n-m-3} or they do not. If they do, then they must differ in a high-order bit (in the range $[n-m-4, n-1]$). If not, we proceed just as in Lemma 2 to find a variable z such that $x_{\alpha+z}y_{\alpha+z}$ and $x_{\beta+z}y_{\beta+z}$ differ by at least 2^{n-m-3}. In order to avoid overly cumbersome notation, let us abuse notation slightly by calling these products $x_\alpha y_\alpha$ and $x_\beta y_\beta$, even though they should possibly be called $x_{\alpha+z}y_{\alpha+z}$ and $x_{\beta+z}y_{\beta+z}$.

Now that we know the products differ in a high-order bit, it remains to ensure that they are not 1-bad. Assume they are. Let d be the greatest index less than n of a bit position in which $x_\alpha y_\alpha$ and $x_\beta y_\beta$ differ.

First, we claim that if the products are 1-bad, then in fact $d \geq n-m-2$. Because if, say, $d = n-m-3$, then the products look like[6]

$$x_\alpha y_\alpha = \cdots 1\,0 \cdots$$
$$x_\beta y_\beta = \cdots \underset{d}{0}\,1\,1\,1\,\underset{n-m-6}{1} \cdots$$

and therefore they differ modulo 2^n by at most $2^{n-m-4}+2^{n-m-7}-1$ (since they agree in bits $d+1$ through $n-1$), but we know they differ by at least 2^{n-m-3}. Furthermore, by the same reasoning, not only is $d \geq n-m-2$, but $x_\alpha y_\alpha$ must have a 1 in some position between $d-2$ and $n-m-4$ inclusive (note that $(x_\alpha y_\alpha)_{d-1} = 0$ or else the products are not 1-bad). For otherwise, the products look like

$$x_\alpha y_\alpha = \cdots \underset{d}{1}\,0\,0\,0\,0\,0 \cdots$$
$$x_\beta y_\beta = \cdots \underset{d}{0}\,1\,1\,1\,1\,1\,1\,\underset{n-m-6}{1} \cdots$$

and again they differ modulo 2^n by at most $2^{n-m-4}+2^{n-m-7}-1$, a contradiction.

So we are reduced to the case that the products are 1-bad, differ in position $d \geq n-m-2$, and $x_\alpha y_\alpha$ has a 1 in some position between $d-2$ and $n-m-4$. Let ℓ be the highest index of a 1 in this range: $x_\alpha y_\alpha = \cdots \underset{d}{1}\,0\,0\,0\,\underset{\ell}{1} \cdots$. We will find a pair of variables (x_u, y_v) with $u+v = n-m-6$ so that the cross terms $2^v x_\alpha, 2^v x_\beta, 2^u y_\alpha, 2^u y_\beta$ all have 0's in positions $n-m-8$ through $n-1$. Then $(2^{u+v}+2^v x_\alpha + 2^u y_\alpha)$ and $(2^{u+v}+2^v x_\beta + 2^u y_\beta)$ both look like $\cdots \underset{n-1}{0}\,0\,0\,0\,0\,0\,0\,0\,\underset{n-m-6}{1}\,0 \cdots$. We see that $x_{\alpha'}y_{\alpha'}$ looks like either $\cdots \underset{d}{1}\,0\,0\,0\,\underset{\ell}{1} \cdots$ or $\cdots \underset{d}{1}\,0\,0\,1\,\underset{\ell}{0} \cdots$, depending on whether there is a carry into position ℓ when $2^{u+v}+2^v x_\alpha + 2^u y_\alpha$ is added to $x_\alpha y_\alpha$. Meanwhile,

$$x_{\beta'}y_{\beta'} = \begin{array}{ll} & \cdots \underset{d}{0}\,1\,1\,1\,1\,1\,1\,1\,1 \cdots \\ + & \cdots 0\,0\,0\,0\,0\,0\,0\,1\,0 \cdots \\ \hline \end{array}$$

looks like $\cdots \underset{d}{1}\,0\,0\,0\,0\,0\,0\,0\,\underset{n-m-6}{0} \cdots$ or $\cdots \underset{d}{1}\,0\,0\,0\,0\,0\,0\,0\,\underset{n-m-6}{1} \cdots$ depending on whether there is a carry into position $n-m-6$ in this addition.

Since $x_{\beta'}y_{\beta'}$ has 0's in positions ℓ ($\leq d-2$) and $\ell-1$ ($\geq n-m-5$), we see that $x_{\alpha'}y_{\alpha'}$ and $x_{\beta'}y_{\beta'}$ look like

$$x_{\alpha'}y_{\alpha'} = \cdots \underset{d}{1}\,0\,0\,0\,0\,\underset{d'}{1} \cdots$$
$$x_{\beta'}y_{\beta'} = \cdots \underset{d}{1}\,0\,0\,0\,0\,\underset{d'}{0}\,0 \cdots,$$

[6]Without loss of generality, let us assume that in position d, $x_\alpha y_\alpha$ has a 1 and $x_\beta y_\beta$ has a 0.

where d' is either ℓ or $\ell+1$. Furthermore, the products agree in all higher bits up to $n-1$ because by the definition of d, $x_\alpha y_\alpha$ and $x_\beta y_\beta$ agree in bits $d+1$ through $n-1$, and we chose x_u and y_v so that the cross terms have 0's in these positions. Since $\ell \geq n-m-4$, it follows that $x_{\alpha'}y_{\alpha'}$ and $x_{\beta'}y_{\beta'}$ differ in a high-order bit and are not even 1-bad.

A counting argument like that for Lemma 3 shows that we may choose x_u and y_v as needed. We require the cross terms to have 0's in at most $m+8$ positions. Since at most $m+1$ bits are set to 1 in x_α or x_β, the total number of values v that we may not choose is $(m+1)(m+8)+(m+1)$. The same number of values u are ruled out, making a total of at most $2(m+1)(m+9) = 2\frac{n}{9} + O(\sqrt{n})$ pairs (x_u, y_v) that are ruled out. Since there are $n-m-5$ pairs to choose from initially, we retain $\Omega(n)$ pairs. □

Proof of Lemma 6. We have four possible cases (up to switching α and β):

$$\begin{array}{lllll} x_\alpha y_\alpha = & (1)\ \cdots \underset{d}{1}0\cdots & (2)\ \cdots \underset{d}{1}1\cdots & (3)\ \cdots \underset{d}{1}1\cdots & (4)\ \cdots \underset{d}{1}0\cdots \\ x_\beta y_\beta = & \cdots \underset{d}{0}0\cdots & \cdots \underset{d}{0}0\cdots & \cdots \underset{d}{0}1\cdots & \cdots \underset{d}{0}1111110\cdots \end{array}$$

Case 1. $x_\alpha y_\alpha = \cdots \underset{d}{1}0\cdots$
$x_\beta y_\beta = \cdots \underset{d}{0}0\cdots.$

It is sufficient to choose (x_u, y_v) so that $u+v=d$ and each of the cross terms $2^v x_\beta$, $2^u y_\beta$, $2^v x_\alpha$, and $2^u y_\alpha$ has 0's in positions $d-3$ through $d+1$. Then the sums $2^v x_\beta + 2^u y_\beta$ and $2^v x_\alpha + 2^u y_\alpha$ have 0's in positions $d-2$ through $d+1$. Adding these to $x_\alpha y_\alpha$ and $x_\beta y_\beta$, respectively, therefore, causes no carry into position d and thus the addition of $2^{u+v} = 2^d$ causes a carry into bit $d+1$ for α but not for β. Since $x_\alpha y_\alpha$ and $x_\beta y_\beta$ agree in bits $d+1$ through $n-1$, this carry bit causes them to differ in bit $d+1$ and possibly higher bits as well.

We now verify that $x_{\alpha'}y_{\alpha'}$ and $x_{\beta'}y_{\beta'}$ are not 1-bad. We know that $2^{u+v} + 2^v x_\beta + 2^u y_\beta$ looks like $\cdots 0\underset{d}{1}00\cdots$. Thus

$$x_{\beta'}y_{\beta'} = \begin{array}{ll} & \cdots \underset{d}{0}0\cdots \\ + & \cdots 0100\cdots \\ \hline \end{array}$$

looks like either $\cdots \underset{d}{1}0\cdots$ or $\cdots \underset{d}{1}10\cdots$, depending on whether there is a carry into position $d-1$. Thus $x_{\beta'}y_{\beta'}$ does not have a string of 1's extending past position $d-1 \geq n-m-5$ and cannot make the products even 1-bad. Since the products differ in position $d+1$ or higher and $x_{\alpha'}y_{\alpha'}$ has a 0 in position d, the products cannot be 1-bad due to a string of 1's in $x_{\alpha'}y_{\alpha'}$.

To prove that we can choose (x_u, y_v) as desired, we argue as in the proof of Lemma 3. The number of positions required to be 0 is 5, ruling out $5t$ values of v. Of the $d+1 = n - O(\sqrt{n})$ pairs (x_u, y_v) such that $u+v=d$, the number of pairs ruled out is at most $2(5t+t) = 12t \leq 12(2m+5) = O(\sqrt{n})$, so there are $n-o(n)$ remaining pairs to choose from.

Case 2. $x_\alpha y_\alpha = \cdots \underset{d}{1}1\cdots$
$x_\beta y_\beta = \cdots \underset{d}{0}0\cdots$

Case 3. $x_\alpha y_\alpha = \cdots \underset{d}{1}1\cdots$
$x_\beta y_\beta = \cdots \underset{d}{0}1\cdots$

It is sufficient to choose (x_u, y_v) as in Case 1 except that $u+v=d-1$. Adding 2^{d-1} will cause a carry to propagate into position $d+1$ for α but not for β, causing them to differ in bit $d+1$ and possibly higher bits as well. The counting argument for choosing (x_u, y_v) is exactly the same as in Case 1 except that there is one fewer pair (x_u, y_v) with $u+v=d-1$.

It only remains to show that in fact $x_{\alpha'}y_{\alpha'}$ and $x_{\beta'}y_{\beta'}$ are not 1-bad. Now $2^{u+v}+2^v x_\alpha + 2^u y_\alpha$ looks like $\cdots \underset{d}{0}\,0\,1\,0\cdots$ and so does $2^{u+v}+2^v x_\beta + 2^u y_\beta$. Thus

$$x_{\alpha'}y_{\alpha'} = \begin{array}{r} \cdots \underset{d}{1}\,1\cdots \\ +\quad \cdots 0\,0\,1\,0\cdots \\ \hline \end{array},$$

and we see that it has a 0 in bit d.

Looking now at $x_{\beta'}y_{\beta'}$, we see that in Case 2,

$$x_{\beta'}y_{\beta'} = \begin{array}{r} \cdots \underset{d}{0}\,0\cdots \\ +\quad \cdots 0\,0\,1\,0\cdots \\ \hline \end{array}$$

looks like either $\cdots \underset{d}{0}\,1\cdots$ or $\cdots \underset{d}{1}\,0\cdots$, depending on whether there is a carry into position $d-1$. In Case 3,

$$x_{\beta'}y_{\beta'} = \begin{array}{r} \cdots \underset{d}{0}\,1\cdots \\ +\quad \cdots 0\,0\,1\,0\cdots \\ \hline \end{array}$$

looks like either $\cdots \underset{d}{1}\,0\cdots$ or $\cdots \underset{d}{1}\,1\,0\cdots$, depending on whether there is a carry into position $d-1$. In any case, $x_{\beta'}y_{\beta'}$ does not have a string of 1's extending past $d-2 \geq n-m-6$, and so $x_{\alpha'}y_{\alpha'}$ and $x_{\beta'}y_{\beta'}$ are not even 1-bad.

Case 4. $x_\alpha y_\alpha = \cdots \underset{d}{1}\,0\cdots,$

$$x_\beta y_\beta = \cdots \underset{d}{0}\,1\,1\,1\,1\overbrace{1\,1\,1\,1}^{k-1}\,0\cdots.$$
$$\qquad\qquad\qquad\qquad\quad \underset{n-m-6}{\uparrow}$$

Without loss of generality, let us say that $x_\beta y_\beta$ contains the maximum number, $k-1$, of consecutive 1's extending past position $n-m-6$. We choose (x_u, y_v) so that $u+v=d$ and the cross terms $2^v x_\alpha$, $2^u y_\alpha$, $2^v x_\beta$, and $2^u y_\beta$ have 0's in positions $(n-m-6)-(k+2)$ through $n-1$. This will ensure that from 2^d we get a carry into position $d+1$ for α' but not for β', causing the products to differ in bit $d+1$ and possibly higher bits as well.

The sum $2^v x_\beta + 2^u y_\beta$ has 0's in positions $(n-m-6)-(k+1)$ through $n-1$, so

$$x_{\beta'}y_{\beta'} = \begin{array}{r} \cdots \underset{d}{0}\,1\,1\,1\,1\overbrace{1\,1\,1\,1}^{k-1}\,0\cdots \\ +\quad \cdots 0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\cdots \\ \hline \end{array}$$

looks like either

$$\cdots \underset{d}{1}\,1\,1\,1\,1\overbrace{1\,1\,1\,1}^{k-1}\,0\cdots \quad \text{or} \quad \cdots \underset{d}{1}\,1\,1\,1\,1\overbrace{1\,1\,1\,1}^{k-1}\,1\,0\cdots,$$

depending on whether there is a carry into position $(n-m-6)-k$. So $x_{\beta'}y_{\beta'}$ has at most k 1's extending past position $n-m-6$. The pair of products cannot be worse than k-bad due to a longer string of 1's in $x_{\alpha'}y_{\alpha'}$ since the products differ in position $d+1$ or higher and $x_{\alpha'}y_{\alpha'}$ has a 0 in position d. Thus $x_{\alpha'}y_{\alpha'}$ and $x_{\beta'}y_{\beta'}$ are at worst k-bad.

The number of positions in which we require $2^v x_\alpha$ or $2^u x_\beta$ to be 0 is $m+6+k+2 \leq 2m+12$. Together, x_α and x_β may rule out $t(2m+12)$ values v in addition to the t variables y_v already in T. Taking into account the same number of values u ruled out by y_α and y_β, there are at most $2(t(2m+12)+t)$ pairs (x_u, y_v) that could be ruled out. Of the $d+1 = n - O(\sqrt{n})$ possible pairs (x_u, y_v) with $u+v=d$, a total of at most

$$2(2m+5)(2m+13) = 8\frac{n}{9} + O(\sqrt{n})$$

pairs are ruled out, leaving $\frac{n}{9} - O(\sqrt{n}) = \Omega(n)$ pairs to choose from. □

5. Problem reductions. With the suitable reductions, we may deduce similar lower bounds for other Boolean functions. Clearly a read-once program for a function g will yield a read-once program for f if there is an appropriate one-to-one substitution of the variables of f for the variables of g. This substitution need not be onto the entire set of g's variables—some of them may be fixed to 0 or 1. This is exactly a one-to-one *projection reduction*, which we shall call a *read-once reduction* as discussed in the following definitions.

DEFINITION 6 (see [SV81]). *A function f is* projection reducible *to a function g, written $f \leq_{\text{proj}} g$, if for all n there is a polynomially bounded function $p(n)$ and a mapping $\sigma_n : \{y_1, \ldots, y_{p(n)}\} \to \{0, 1, x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$ such that*

$$f_n(x_1, \ldots, x_n) = g_{p(n)}(\sigma(y_1), \ldots, \sigma(y_{p(n)})).$$

In other words, $f \leq_{\text{proj}} g$ if any algorithm (circuit or branching program) for $g(y_1, \ldots, y_{p(n)})$ can be used as a black box for f simply by substituting the inputs to f (and $0, 1$) for the inputs to g so that the output of the algorithm is that of f. These reductions were used in the study of constant-depth reducibility [CSV84]—clearly, given that $f \leq_{\text{proj}} g$, if $g \in \text{AC}^0$ then $f \in \text{AC}^0$.

DEFINITION 7. *A function f is* read-once reducible *to a function g, written $f \leq_{r\text{-}o} g$, if there is a projection reduction σ from f to g such that for all k and $i \neq j$,*

$$\sigma(y_i) \in \{x_k, \overline{x_k}\} \Longrightarrow \sigma(y_j) \notin \{x_k, \overline{x_k}\}.$$

That is, each of f's inputs x_i is substituted for no more than one input y_j of g. It follows that a polynomial-size read-once branching program for $f(x_1, \ldots, x_n)$ is obtained by relabeling and reducing the nodes of a polynomial-size read-once program for $g(y_1, \ldots, y_{p(n)})$. We remark that the same holds for read-once *formulas* (e.g., [KLNSW93, Gu77]).

5.1. Reductions to other arithmetic functions. Projection reductions have been used to deduce tight lower bounds on the depth of polynomial-size threshold circuits. It was originally proved in [HMPST93] that **INNER-PRODUCT-MODULO-2** cannot be computed in polynomial-size by threshold circuits of depth 2. It was also noted there that the projection reduction to multiplication (first given in [FSS84], from **PARITY** to **MULT**) shows that **MULT** obeys the same lower bound. Wegener [We93] gives projection reductions from **MULT** to squaring and inversion in order

to show that these functions also require depth 3 polynomial-size threshold circuits. Our lower bound for the middle bit of multiplication implies a lower bound for the appropriate bit of these two functions. We will phrase the reductions of [We93] in terms of the following Boolean functions:

- **SQUARING** : $\{0,1\}^n \to \{0,1\}$; computes "the" middle bit (here, bit n rather than bit $n-1$, which we chose for **MULT**) in the square of an n-bit integer:

$$\textbf{SQUARING}(z) = (z^2)_n.$$

- **INVERSION** : $\{0,1\}^n \to \{0,1\}$; computes the ones' bit in the reciprocal of an n-bit number between 0 and 1:

$$\textbf{INVERSION}(x) = y_0,$$

  where x represents the number $0\,.\,x_1x_2\cdots x_n = \sum_i x_i 2^{-i}$ and $y = y_n \cdots y_0 = \sum_i y_i 2^i$ is the integral part of $1/x$. (Note that $1 < y \le 2^n$.) Define the function to be 0 if all x_i are 0.

Wegener actually shows that

$$\textbf{MULT} \le_{\text{proj}} \textbf{SQUARING} \le_{\text{proj}} \textbf{INVERSION},$$

except that the reductions are given for all bits of multiplication, squaring, and inversion. We provide the reductions here in order to verify that each reduction is in fact read-once and also because we are working instead with Boolean versions of these functions. The polynomial $p(n)$ of the reduction is linear in both cases, implying that if each bit of the function is computable with a read-once program of size $f(n)$, then **MULT** is computable with a read-once program of size $f(cn)$ for some constant c. This gives the following corollaries to Theorem 2.

COROLLARY 1. *Read-once branching programs for* **SQUARING** *require size* $2^{\Omega(\sqrt{n})}$.

Proof. The reduction $\textbf{MULT} \le_{\text{r-o}} \textbf{SQUARING}$ is given by mapping the n-bit inputs x, y (of **MULT**) to the $(3n+2)$-bit input $z = x2^{2(n+1)} + y$ (of **SQUARING**), so that $z^2 = x^2 2^{4(n+1)} + xy2^{2(n+1)+1} + y^2$. The middle bit of the product xy is found in the middle bit of z^2: $(xy)_{n-1} = (z^2)_{3n+2}$. It is clear that the mapping σ is injective since

$$\sigma(z_i) = \begin{cases} y_i & \text{if } 0 \le i < n, \\ 0 & \text{if } n \le i < 2(n+1), \\ x_{i-2(n+1)} & \text{if } 2(n+1) \le i < 2(n+1)+n. \end{cases} \qquad \square$$

COROLLARY 2. *Read-once branching programs for* **INVERSION** *require size* $2^{\Omega(\sqrt{n})}$.

Proof. The reduction $\textbf{SQUARING} \le_{\text{r-o}} \textbf{INVERSION}$ essentially reduces the problem of computing the square of an n-bit integer m to the problem of computing $1/(1-x) = 1 + x + x^2 + x^3 + \cdots$ where

$$1 - x = 1 - m\,2^{-4n} - 2^{-10n},$$

which is a $10n$-bit number slightly less than 1. The proof in [We93] shows that the product m^2 lies in bit positions $-6n-1$ through $-8n$ in $1/(1-x)$, its middle bit being in position $-7n$. By instead computing the inverse of $2^{-7n}(1-x)$, a $17n$-bit number, we find the middle bit of m^2 in position 0.

For example, working in decimal, we may compute 5^2 (so $n = 1$) by calculating

$$10^{-7}(1 - 5 \cdot 10^{-4} - 10^{-10})^{-1} = 10005002.50225\cdots$$

from which we may recover the middle digit, 2, of 25 in position in position 0.

To see that the mapping σ is injective, simply notice that $1-x = 1-2^{-10n}-m\,2^{-4n}$ has 1's in all positions -1 through $-10n$, except in positions $-3n-1$ through $-4n$, where it has exactly the complements of the bits of m. The number $2^{-7n}(1-x)$ is similar, with extra 0's on the left. □

6. Further work. We doubt that $2^{\Theta(\sqrt{n})}$ is the true read-once complexity of **Mult** (Bryant's lower bound for OBDDs is $2^{n/8}$ [Br91]), but the simple counting technique used in our proof seems limited to this lower bound. It is curious that many of the lower bounds for read-once programs achieve only $2^{\Omega(\sqrt{n})}$ if n is the number of input bits—only the lower bounds of [BHST87] and [SZ96] achieve a fully exponential lower bound of $2^{\Omega(n)}$. This limitation is most likely an artifact of the proofs, but it is not well understood. In addition to improving the bound, it may also be possible to extend the argument using the framework of [BRS93] to show that a similar bound holds for nondeterministic read-once programs or for read-k-times programs.

Acknowledgments. Thanks to Mikael Goldmann for pointing out the reductions in [We93], to Mauricio Karchmer and Ravi Sundaram for discussions, and to Allan Borodin for his seminar in which I learned of this problem. Thanks also to the referees for catching several small errors in the first version.

REFERENCES

[AGD91] P. Ashar, A. Ghosh, and S. Devadas, *Boolean satisfiability and equivalence checking using general binary decision diagrams*, in Proc. Int'l. Conference on Computer Design, Boston, 1991, IEEE, pp. 259–264.

[AM88] N. Alon and W. Maass, *Meanders and their applications in lower bounds arguments*, J. Comput. System Sci., 37 (1988), pp. 118–129.

[BC94] R. Bryant and Y. Chen, *Verification of Arithmetic Functions with Binary Moment Diagrams*, Tech. report CMU-CS-94-160, Carnegie Mellon University, Pittsburgh, PA, 1994.

[BCW80] M. Blum, A. Chandra, and M. Wegman, *Equivalence of free boolean graphs can be decided probabilistically in polynomial time*, Inform. Process. Lett., 10 (1980), pp. 80–82.

[BF85] F. Brglez and H. Fujiwara, *A neutral netlist of* 10 *combinational circuits*, in Proc. 1985 IEEE Int'l. Symposium on Circuits and Systems.

[BHR95] Y. Breitbart, H. B. Hunt, III, and D. Rosenkrantz, *On the size of binary decision diagrams representing Boolean functions*, Theoret. Comput. Sci., 145 (1995), pp. 45–69.

[BHST87] L. Babai, A. Hajnal, E. Szemeredi, and G. Turan, *A lower bound for read-once branching programs*, J. Comput. System Sci., 37 (1988), pp. 153–162.

[Br91] R. Bryant, *On the complexity of VLSI implementations and graph representations of Boolean functions with applications to integer multiplication*, IEEE Trans. Comput., 40 (1991), pp. 205–213.

[Br92] R. Bryant, *Symbolic boolean manipulation with ordered binary decision diagrams*, ACM Computing Surveys, 24 (1992), pp. 293–318.

[BRS93] A. Borodin, A. Razborov, and R. Smolensky, *On lower bounds for read-k-times branching programs*, Comput. Complexity, 3 (1993), pp. 1–18.

[BSSW93] B. Bollig, M. Sauerhoff, D. Sieling, and I. Wegener, *Read-k-times Ordered Binary Decision Diagrams—Efficient Algorithms in the Presence of Null Chains*, Tech. report 474, Univ. Dortmund, 1993.

[BSSW95] B. Bollig, M. Sauerhoff, D. Sieling, and I. Wegener, *Hierarchy theorems for kOBDDs and kIBDDs*, Theoret. Comput. Sci., submitted; also available via `http://www.eccc.uni-trier.de/eccc/` as TR94-026 (1994).

[BW95] B. Bollig and I. Wegener, *Read-Once Projections and formal Circuit Verification with Binary Decision Diagrams*, Electronic Colloquium on Computational Complexity, TR95-042, 1995, `http://www.eccc.uni-trier.de/eccc/`.

[Bu92] S. Buss, *The graph of multiplication is equivalent to counting*, Inform. Process. Lett., 41 (1992), pp. 199–201.

[CSV84] A. Chandra, L. Stockmeyer, and U. Vishkin, *Constant depth reducibility*, SIAM J. Comput., 13 (1984), pp. 423–439.

[Du85] P. E. Dunne, *Lower bounds on the complexity of* 1*-time only branching programs*, Lecture Notes in Comput. Sci. 199, Springer-Verlag, New York, 1985, pp. 90–99.

[FHS78] S. Fortune, J. Hopcroft, and E. M. Schmidt, *The complexity of equivalence and containment for free single variable program schemes*, Lecture Notes in Comput. Sci. 62, Springer-Verlag, New York, 1978, pp. 227–240.

[FSS84] M. Furst, J. B. Saxe, and M. Sipser, *Parity, circuits, and the polynomial-time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.

[Ge94] J. Gergov, *Time-space tradeoffs for integer multiplication on various types of input-oblivious sequential machines*, Inform. Process. Lett., 51 (1994), pp. 265–269.

[GM94] J. Gergov and C. Meinel, *Efficient Boolean manipulations with OBDD's can be extended to FBDD's*, IEEE Trans. Comput., 43 (1994), pp. 1197–1209.

[Gu77] V. A. Gurvich, *On the normal form of positional games*, Uspekhi Mat. Nauk, 32 (1977), pp. 183–184 (in Russian).

[HMPST93] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán, *Threshold circuits of bounded depth*, J. Comput. System Sci., 46 (1993), pp. 129–154.

[JABFA92] J. Jain, M. Abadir, J. Bitner, D. Fussell, and J. Abraham, *IBDD's: An efficient functional representation for digital circuits*, in Proceedings of the European Conference on Design Automation, 1992, pp. 440–446.

[Ju88] S. Jukna, *Entropy of contact circuits and lower bounds on their complexity*, Theoret. Comput. Sci., 47 (1988), pp. 113–129.

[Ju89] S. Jukna, *The effect of null-chains on the complexity of contact schemes*, in Proc. FCT, Lecture Notes in Comput. Sci. 380, Springer-Verlag, New York, 1989, pp. 246–256.

[Ju92] S. Jukna, *A note on read-k-times branching programs*, RAIRO Theoret. Inform. Appl., 29 (1995), pp. 75–83.

[Ju94] S. Jukna, *The Graph of Multiplication is Hard for Read-k-Times Networks*, Tech. report 95-10, University of Trier, Trier, Germany, 1995.

[KLNSW93] M. Karchmer, N. Linial, I. Newman, M. Saks, and A. Wigderson, *Combinatorial characterization of read-once formulae*, Discrete Math., 114 (1993), pp. 275–282.

[KMW91] M. Krause, C. Meinel, and S. Waack, *Separating the eraser Turing machine classes* L_e*,* NL_e*,* co-NL$_e$*, and* P_e, Theoret. Comput. Sci., 86 (1991), pp. 267–275.

[Kr88] M. Krause, *Exponential lower bounds on the complexity of real time and local branching programs*, J. Inform. Processing and Cybernetics (EIK), 24 (1988), pp. 99–110.

[Kr91] M. Krause, *Lower bounds for depth-restricted branching programs*, Inform. and Comput., 91 (1991), pp. 1–14.

[Kr94] S. Krischer, *FANCY, version* 1.1, `http://www.informatik.uni-trier.de/~krischer/`. Universität Trier, Germany, November 1994.

[KW91] M. Krause and S. Waack, *On oblivious branching programs of linear length*, Inform. Comput., 94 (1991), pp. 232–249.

[Ma76] W. Masek, *A Fast Algorithm for the String-Editing Problem and Decision Graph Complexity*, SM Thesis, MIT, Cambridge, MA, 1976.

[Me89] C. Meinel, *Modified branching programs and their computational power*, Lecture Notes in Comput. Sci. 370, Springer-Verlag, New York, 1989.

[Ok91] E. A. Okolnishnikova, *Lower bounds for branching programs computing characteristic functions of binary codes*, Metody Diskret. Anal., 51 (1991), pp. 61–83 (in Russian).

[Po95] S. Ponzio, *Restricted Branching Programs and Hardware Verification*, Ph.D. thesis, MI Technology, Cambridge, MA, August 1995; Tech. report MIT/LCS/TR-663, `http://www.lcs.mit.edu/` and `http://www.eccc.uni-trier.de/eccc/`.

[SDG94] A. Shen, S. Devadas, and A. Ghosh, *Probabilistic manipulation of boolean functions using free boolean diagrams*, IEEE Trans. Computer-Aided Design, 14 (1995), pp. 87–95.

[SS93] J. Simon and M. Szegedy, *A new lower bound theorem for read-only-once branching programs and its applications*, in Advances in Computational Complexity Theory, J. Cai, ed., DIMACS Series, Vol. 13, AMS, Providence, RI, 1993, pp. 183–193.

[SV81] S. Skyum and L. G. Valiant, *A complexity theory based on Boolean algebra*, J. Assoc. Comput. Mach., 32 (1985), pp. 484–502.

[SW95] D. Sieling and I. Wegener, *Graph driven BDD's—A new data structure for boolean functions*, Theoret. Comput. Sci., 141 (1995), pp. 283–310.

[SZ96] P. Savicky and S. Zak, *A large lower bound for* 1*-branching programs*, Electronic Colloquium on Computational Complexity, TR96-036, 1996, `http://www.eccc.-uni-trier.de/eccc/`.

[We87] I. Wegener, *The Complexity of Boolean Functions*, Wiley-Teubner Series in Computer Science, New York, Stutggart, 1987.

[We88] I. Wegener, *On the complexity of branching programs and decision trees for clique functions*, J. Assoc. Comput. Mach., 35 (1988), pp. 461–471.

[We93] I. Wegener, *Optimal lower bounds on the depth of polynomial-size threshold circuits for some arithmetic functions*, Inform. Process. Lett., 46 (1993), pp. 85–87.

[We94] I. Wegener, *Efficient data structures for boolean functions*, Discrete Math., 136 (1994), pp. 347–372.

[Za84] S. Zak, *An exponential lower bound for one-time-only branching programs*, in Proc. 11th MFCT, Lecture Notes in Comput. Sci. 176, Springer-Verlag, New York, 1984, pp. 562–566.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 816–821

TOTAL COLORING WITH $\Delta + \text{poly}(\log \Delta)$ COLORS*

HUGH HIND†, MICHAEL MOLLOY‡, AND BRUCE REED§

Abstract. We provide a polynomial time algorithm which finds a total coloring of any graph with maximum degree Δ, Δ sufficiently large, using at most $\Delta + 8\log^8 \Delta$ colors. This improves the best previous upper bound on the total chromatic number of $\Delta + 18\Delta^{1/3}\log(3\Delta)$.

Key words. total coloring, algorithms, probabilistic method

AMS subject classifications. 05C15, 05C70, 0C85

PII. S0097539795294578

1. Introduction. A *total coloring* of a graph G is an assignment of colors to its vertices and edges so that no two adjacent vertices have the same color, no two adjacent edges have the same color, and no edge has the same color as one of its endpoints. A k total coloring is a total coloring which uses at most k colors. The *total chromatic number,* $\chi''(G)$, is the least number of colors required for a total coloring of G.

This concept was introduced independently by Behzad [3] and Vizing [15], who each conjectured that any graph with maximum degree Δ has a $\Delta + 2$ total coloring. Note that if it is true, this conjecture is tight because every such graph requires at least $\Delta + 1$ colors and there are some graphs such as $K_{\Delta+1}$, Δ odd, which require $\Delta + 2$ colors. Kilakos and Reed [11] have shown that the fractional total chromatic number of such a graph is at most $\Delta + 2$.

The first $\Delta + o(\Delta)$ bound on the total chromatic number of such a graph was $\Delta + 2\sqrt{\Delta}$, due to Hind [7]. More recently, Häggkvist and Chetwynd (see [10]) have reported a bound of $\Delta + 18\Delta^{1/3}\log(3\Delta)$. In this paper we tighten the bound to $\Delta + 8\log^8 \Delta$ (all logarithms have base e).

THEOREM 1. *For sufficiently large Δ, if G has maximum degree Δ then $\chi''(G) \leq \Delta + 8\log^8 \Delta$.*

Our proof is probabilistic and makes use of the Lovász local lemma. The proof can be made constructive, providing an $O(n^3 \log^{O(1)} n)$ randomized algorithm and a polytime deterministic algorithm to find such a total coloring.

The total chromatic number conjecture is reminiscent of Vizing's theorem which states that if G has maximum degree Δ then the edge chromatic number of G, $\chi'(G)$ is either Δ or $\Delta + 1$. It is also reminiscent of the list coloring conjecture. In fact, a slightly weaker form of the total coloring conjecture follows from the list coloring conjecture.

The *list edge chromatic number* of a graph G, $\chi'_\ell(G)$, is the minimum number r with the following property: for any mapping $f : E(G) \to \mathcal{S}$ where $\mathcal{S}$ is a collection of sets of colors each of size r, G has a proper edge coloring where for each edge e,

*Received by the editors November 3, 1995; accepted for publication (in revised form) January 3, 1997; published electronically September 14, 1998. The second and third authors were supported by NATO Collaborative Research grant CRG950235.

http://www.siam.org/journals/sicomp/28-3/29457.html

†Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada.

‡Department of Computer Science, University of Toronto, Toronto, Canada (molloy@cs.toronto.edu).

§Equipe Combinatoire, CNRS, Université Pierre et Marie Curie, Paris, France (reed@lug.ecp6.jussieu.fr).

the color of e lies in $f(e)$. The list coloring conjecture is that $\chi_\ell^{'}(G) = \chi^{'}(G)$.

Recall that for any graph G, $\chi(G) \leq \Delta + 1$. Now consider any $\Delta + 1$ coloring $c : V(G) \to \{1, \ldots, \Delta + 1\}$. For each edge $e = (u, v)$ define $f(e)$ to be the set $\{1, \ldots, \Delta + 3\} - \{c(u), c(v)\}$. Now the size of $f(e)$ is $\Delta + 1$ for each e, and so if the list coloring conjecture holds, then we can use such a coloring to provide a $\Delta + 3$ total coloring of G. Therefore, the list coloring conjecture implies $\chi^{''}(G) \leq \Delta + 3$.

Inspired by this implication, we say that a proper vertex coloring is *extendible* to a t total coloring if there is a total coloring of size t whose restriction to $V(G)$ is that vertex coloring. Thus, we have seen that the list coloring conjecture implies that every $\Delta + 1$ vertex coloring of G is extendible to a $\Delta + 3$ total coloring of G. Hind [8] has shown that there exist graphs that have a $\Delta + 1$ vertex coloring which is not extendible to a $\Delta + 2$ total coloring.

In [9] we define a proper vertex coloring to be *β-frugal* if no vertex has more than β members of any color class in its neighbourhood. We prove the following theorem.

THEOREM 2. *Every graph G with maximum degree $\Delta \geq \Delta_0 = \mathrm{e}^{10^7}$ has a $\log^5 \Delta$-frugal $(\Delta + 1)$ vertex coloring.*

In this paper, we show that every $\log^5 \Delta$-frugal $(\Delta+1)$ vertex coloring is extendible to a $\Delta + 8 \log^8 \Delta$ total coloring, thus proving Theorem 1.

The idea behind our proof is simple. We begin by presenting the basic ideas. For ease of exposition, let us assume for now that G is Δ-regular. Consider any $\log^5 \Delta$-frugal $(\Delta + 1)$ vertex coloring of G with color classes $S_1, \ldots, S_{\Delta+1}$. If we could find an edge disjoint sequence of matchings $M_1, \ldots, M_{\Delta+1}$ such that M_i misses all of S_i and covers all of $V(G) - S_i$ (i.e., M_i is a perfect matching of $G - S_i$), then this would give us a $\Delta + 1$ total coloring. (Note that since every vertex is missed by exactly one matching here, then $\cup M_i = E(G)$.) Of course, this is not always possible as there are some graphs with $\chi^{''} \geq \Delta + 2$. For example, we will fail if $|V(G) - S_i|$ is odd for any i. Thus we will have to allow our matchings to miss a few more vertices. Essentially, we will show that we can find sets $X_1, \ldots, X_{\Delta+1}$ with the following two properties:

1. each vertex lies in at most $\log^8 \Delta$ of these sets; and
2. for each $1 \leq i \leq \Delta + 1$, we can find a matching M_i in $G_i = G - \cup_{1 \leq j \leq i-1} M_i$ which misses all of S_i and meets all of $V(G) - S_i - X_i$.

Therefore, the color classes $C_i = S_i \cup M_i$ will provide a total coloring of all but $E(G_{\Delta+2})$. By condition 1, $G_{\Delta+2}$ has maximum degree at most $\log^8 \Delta$ and so it can be edge colored with at most $\log^8 \Delta + 1$ colors thus providing a $\Delta + \log^8 \Delta + 2$ total coloring of G.

We have oversimplified things here. In fact, our argument is more intricate. In the next section we will fill in the details, including the manner in which we choose our sets X_i. For now, we will simply say that we choose them randomly and make use of the following two tools. The first is due to Lovász and appears in [4]. The second can be found in [6].

The local lemma. *Suppose $\mathcal{A} = A_1, \ldots, A_n$ is a list of random events such that for each i, $\mathbf{Pr}(A_i) \leq p$ and A_i is mutually independent of all but at most d other events in $\mathcal{A}$. If $\mathrm{e}p(d + 1) < 1$ then $\mathbf{Pr}(\wedge_{i=1}^n \bar{A}_i) > 0$.*

The Chernoff bounds. *Suppose $B(n, p)$ is the sum of n independent Bernoulli variables each equal to 1 with probability p. Then for any $0 < a < \frac{1}{6}np$ we have the following:*

$$\mathbf{Pr}(B(n, p) - np > a) < \mathrm{e}^{-a^2/3np}$$

and

$$\mathbf{Pr}(B(n,p) - np < -a) < \mathrm{e}^{-a^2/2np}.$$

For the remainder of this paper, we assume $\Delta \geq \mathrm{e}^{10^7}$. For each vertex v, $N(v)$ denotes the neighbourhood of v. We usually omit all $\lfloor, \rfloor$ and $\lceil, \rceil$ signs.

2. The details. Our main task will be to prove the following.

LEMMA 3. *Suppose G is a graph with maximum degree at most $D \geq 8\log^8 \Delta$. Suppose further that we are given $S_1, S_2, \ldots, S_{\frac{D}{2}} \subseteq V(G)$ such that for all $v \in V(G), 1 \leq i \leq \frac{D}{2}$, $|N(v) \cap S_i| \leq \log^5 \Delta$. Then there exists a sequence of edge-disjoint matchings in G, $M_1, \ldots, M_{\frac{D}{2}}$ such that*

1. *M_i misses S_i;*
2. *$G' = G - \cup_{i=1}^{\frac{D}{2}} M_i$ has maximum degree at most $\frac{D}{2} + 2\log^7 \Delta$.*

Repeated iterations of Lemma 3 will prove Theorem 1.

Proof of Theorem 1. Take $S_1, \ldots, S_{\Delta+1}$ to be the color classes of any $\log^5 \Delta$-frugal $\Delta + 1$ coloring of G, as guaranteed by Theorem 2. Set $G_0 = G$, $\Delta_0 = \Delta$, and repeatedly apply Lemma 1 until $\Delta_j < 8\log^8 \Delta$, setting $G_{j+1} = G'$, $\Delta_{j+1} = \Delta_j/2 + 2\log^7 \Delta \leq \frac{\Delta}{2^j} + 4\log^7 \Delta$, and choosing $S_1^{(j)}, \ldots, S_{\Delta_j/2}^{(j)}$ from previously unused members of $\{S_1, \ldots, S_{\Delta+1}\}$, all the while forming color classes from the pairs $S_i \cup M_i$. As there are at most $\log \Delta$ iterations, $\sum \Delta_i/2 \leq \Delta - 4\log^8 \Delta + \log \Delta(4\log^7 \Delta) < \Delta + 1$, and so we will have produced fewer than $\Delta + 1$ color classes. Therefore, an $8\log^8 \Delta$ edge coloring of the final G' will provide our $\Delta + 8\log^8 \Delta$ total coloring. □

We prove Lemma 3 by choosing random sets X_i, which we allow M_i to miss as described in the introduction. As mentioned earlier, it is important that no vertex appears in very many sets X_i, as that will cause its degree in G' to be too high. In order to ensure this, we will divide $\{1, \ldots, D/2\}$ into $\log^7 \Delta$ subsequences, and we will insist that no vertex falls into two sets from the same subsequence.

Specifically, we set $\alpha_j = \lfloor j \times \frac{D}{\log^7 \Delta} \rfloor$, $j = 0, \ldots, \lceil \frac{1}{2}\log^7 \Delta \rceil$, and we set $A_j = \{\alpha_{j-1}+1, \ldots, \alpha_j\}$. We will choose $X_1, \ldots, X_{D/2}$ such that for $i_1, i_2 \in A_j$, $X_{i_1} \cap X_{i_2} = \emptyset$.

For $i \in A_j$, we define $D_i = D - (i-1) + 2(j-1)$. We set $G_1 = G$, and for $1 \leq i \leq D/2$ we will find a matching M_i in $G_i = G - \cup_{j=1}^{i-1} M_i$ such that

1. M_i misses S_i;
2. M_i meets every vertex of degree at least D_i in $V(G) - S_i - X_i$.

We will see how to find X_i and M_i later, but first note that this will be enough to prove Lemma 3.

CLAIM 1. *For each $i \geq 2$, if M_{i-1} exists, then G_i has maximum degree at most $D_i + 2$.*

Proof. In what follows, we only discuss $k \leq i$ and j such that $\alpha_{j-1} \leq i$. Consider any vertex v. We denote by $\deg_k(v)$ the degree of v in G_k. We will see by induction on j that $\deg_{\alpha_{j-1}+1}(v) \leq D_{\alpha_{j-1}+1}$ and that for all k with $\alpha_{j-1} + 1 \leq k \leq \alpha_j$, $\deg_k(v) \leq D_k + 2$. The first condition holds for $j = 1$. To see that for each j, the first condition implies the second condition as well as the first condition for $j+1$, consider the first (if any) $k \in A_j$ such that $\deg_k(v) = D_k$, and note that $\deg_{k'}(v) \geq D_{k'}$ for all $k \leq k' \leq \alpha_j$, and so v can be missed by at most two matchings M_{k_1}, M_{k_2} before $\alpha_j + 1$, corresponding to $v \in X_{k_1}$ and $v \in S_{k_2}$. Therefore, $\deg_{k'}(v) \leq D_{k'} + 2$ for each $k \leq k' \leq \alpha_j$, and $\deg_{\alpha_j+1}(v) \leq D_{a_j+1}$ as $D_{a_j+1} = D_{a_j} + 1$. □

Therefore, $G_{D/2}$ has maximum degree at most $D_{D/2} + 2 \leq D/2 + 2\log^7 \Delta$ as required.

It only remains to choose X_i and M_i. This is done via the following two lemmas.

LEMMA 4. *Suppose G_i has maximum degree at most $D_i + 2$ where $D_i \geq \log^8 \Delta$, and suppose further that there exists a set $R \subseteq V(G)$ such that for any $v \in V(G)$, $|N(v) \cap R| \leq \frac{3D}{\log \Delta}$. Then there exists $X_i \subseteq V(G) - R$ such that for all $v \in V(G)$,*

1. $|N(v) - X_i| \leq D_i - \log^6 \Delta$;
2. $|N(v) \cap X_i| \leq 3 \log^6 \Delta$.

LEMMA 5. *Suppose G_i has maximum degree at most $D_i + 2$ and suppose further that we have $S_i, X_i \subseteq V(G)$ such that for each $v \in V(G)$,*

1. $|N(v) \cap S_i| \leq \log^5 \Delta$;
2. $|N(v) - X_i| \leq D_i - \log^6 \Delta$;
3. $|N(v) \cap X_i| \leq 3 \log^6 \Delta$.

Then there exists a matching M_i such that

1. *M_i misses S_i;*
2. *M_i meets every vertex of degree at least D_i in $V(G) - S_i - X_i$.*

Using these lemmas, along with Claim 1, it is now straightforward to prove Lemma 3 in the manner discussed earlier.

Proof of Lemma 3. For $i = 1, \ldots, D/2$, we choose X_i via Lemma 4 by setting $R = \cup_{k \in A_j, k<i} X_k$ where $i \in A_j$, noting that $|N(v) \cap R| \leq |A_j| \times 3 \log^6 \Delta \leq \frac{3D}{\log \Delta}$, and we choose M_i via Lemma 3. The result now follows as in the earlier discussion. ∎

We now complete the proof of Theorem 1 by proving Lemmas 4 and 5.

Proof of Lemma 4. We will choose X_i randomly. For each $v \in V(G) - R$, we place v in X_i with probability $p_i = \frac{2 \log^6 \Delta}{D_i + 2}$. For each $v \in V(G)$ define E_v to be the event that v violates one of the required conditions. Note that by the Chernoff bounds,

$$\begin{aligned}
\mathbf{Pr}(E_v) &\leq \mathbf{Pr}(|B(D_i + 2, p_i) - 2 \log^6 \Delta| > \tfrac{1}{3} \log^6 \Delta) \\
&\qquad + \mathbf{Pr}(|B(D_i - \tfrac{3D}{\log \Delta}, p_i) - p_i \times (D_i - \tfrac{3D}{\log \Delta})| > \tfrac{1}{6} \log^6 \Delta) \\
&\leq 2\mathrm{e}^{-\log^6 \Delta/54} + 2\mathrm{e}^{-\log^6 \Delta/108} \\
&< \Delta^{-3}.
\end{aligned}$$

Furthermore, each event E_v is mutually independent of all but at most Δ^2 other events. Therefore, our result follows from the local lemma as $\mathrm{e}\Delta^{-3}(\Delta^2 + 1) < 1$. ∎

Proof of Lemma 5. Suppose that such a matching does not exist. Then by a well-known extension of Tutte's theorem, there exist disjoint $T, Q \subset V(G_i) - S_i$ with $T \cap X_i = \emptyset$ such that each $v \in T$ has degree at least D_i in G; the subgraph induced by T has at least $|Q| + 1$ odd components $C_1, C_2, \ldots, C_{|Q|+1}$; and there are no edges from T to $G - (Q \cup S_i)$. (One way to see this is form G_i' by deleting S_i from G_i and then adding an edge between every pair of nonadjacent vertices that each have degree less than D_i in G_i. If $|G_i'|$ is odd, then add a vertex that is adjacent to every vertex of degree less than D_i in G_i. Now apply Tutte's theorem (see, e.g., [12]) to G_i'.)

For any disjoint $A, B \subseteq V(G_i)$, denote by $E(A, B)$ the set of edges with one endpoint in each of A, B.

CLAIM 2. *For each $1 \leq i \leq |Q| + 1$, $|E(C_i, Q)| \geq D_i - \log^5 \Delta$.*

Proof.

Case 1. $|C_i| \leq D_i/3$.

$|E(C_i, Q \cup S_i)| \geq D_i |C_i| - \binom{|C_i|}{2}$, and $|E(C_i, S_i)| \leq |C_i| \log^5 \Delta$. Therefore, $|E(C_i, Q)| \geq |C_i|(D_i - \log^5 \Delta) - \binom{|C_i|}{2} \geq D_i - \log^5 \Delta$.

Case 2. $|C_i| > D_i/3$.

For each $v \in C_i$, $|E(\{v\}, X_i)| \geq \log^6 \Delta$ and $|E(\{v\}, S_i)| \leq \log^5 \Delta$. Therefore, $|E(C_i, Q)| \geq |C_i|(\log^6 \Delta - \log^5 \Delta) \geq D_i - \log^5 \Delta$. □

CLAIM 3. *For each* $v \in Q$, $|E(\{v\}, T)| \leq D_i - \log^6 \Delta$.

Proof. This follows from the fact that $T \cap X_i = \emptyset$. □

Therefore, we have $(|Q| + 1)(D_i - \log^5 \Delta) \leq |E(T, Q)| \leq |Q|(D_i - \log^6 \Delta)$. Thus $D_i \leq \log^5 \Delta$ which contradicts $D \geq 8 \log^8 \Delta$. □

3. An efficient algorithm. We note here that our proof can be made algorithmic using the techniques developed by Beck [2], at the price of increasing our lower bound on Δ. Set $n = |V(G)|$.

In [9] the present authors provide an $O(n^3 \log^{O(1)} n)$ randomized algorithm and a polytime deterministic algorithm to find a $\log^5 \Delta$-frugal $(\Delta + 1)$-coloring of G. After doing this, we must find the set X_i guaranteed by Lemma 2 and the matching M_i guaranteed by Lemma 3 fewer than Δ times, and finally we must find the $8 \log^8 \Delta$ edge coloring used in the proof of Theorem 1. The latter step can be done in $O(n^4)$ steps, or we can find a $16 \log^8 \Delta$ edge coloring in $O(n^2)$ steps. Each M_i can be found in $O(n^{2.5})$ steps, as in [5] or [13]. It only remains to find X_i.

This can be done in $O(n^2 \log^{O(1)} n)$ steps using an algorithm essentially the same as that in section 4 of [2]. The only modification required is to allow for sampling with probability p_i here rather than with probability $\frac{1}{2}$ as in [2]. This can be done in a straightforward manner as described in [9].

Thus we have a $O(n^{2.5} \log^{O(1)} n)$-time randomized algorithm and a polytime deterministic algorithm for finding a $\Delta + 16 \log^8 \Delta$ total coloring of G.

4. Remarks. It is worth noting that by being more careful with our calculations, both here and in [9], and raising our lower bound on Δ, we can find a $\log^3 \Delta$-frugal $(\Delta + 1)$-coloring for any graph with maximum degree Δ sufficiently large, and we can find a $\Delta + \log^4 \Delta$ total coloring. However, these techniques do not appear to be sufficient to get a bound any lower than $\Delta + \text{poly}(\log \Delta)$.

Alon [1] has shown how to modify the technique of [2] to produce parallel algorithms. This does not seem to apply here.

Recently, Molloy and Reed [14] have improved the upper bound on the total chromatic number to $\Delta + C$ for a large constant C. The proof uses a different, much more complicated technique and does not appear to yield an efficient algorithm for finding such a coloring.

Acknowledgments. We would like to thank two anonymous referees for several improvements.

REFERENCES

[1] N. ALON, *A parallel algorithmic version of the local lemma*, Random Structures Algorithms, 2 (1991), pp. 367–378.

[2] J. BECK, *An algorithmic approach to the Lovász local lemma*, Random Structures Algorithms, 2 (1991), pp. 343–365.

[3] M. BEHZAD, *Graphs and Their Chromatic Numbers*, Ph.D. thesis, Michigan State University, East Lansing, 1965.

[4] P. ERDŐS AND L. LOVÁSZ, *Problems and results on 3-chromatic hypergraphs and some related questions*, in Infinite and Finite Sets Colloq. Math. Soc. János Bolyai 11, A. Hajnal et al., eds., North–Holland, Amsterdam, 1975, pp. 609–627.

[5] S. Even and O. Kariv, *An $O(n^{2.5})$ algorithm for maximum matching in general graphs*, Proceedings of the 16th Annual Symposium on the Foundations of Computer Science, 1975, pp. 100–112.
[6] W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 1, Wiley, New York, 1966.
[7] H. Hind, *An improved bound for the total chromatic number of a graph*, Graphs Combin., 6 (1990), pp. 153–159.
[8] H. Hind, *Recent developments in total colouring*, Discrete Math., 125 (1994), pp. 211–218.
[9] H. Hind, M. Molloy, and B. Reed, *Colouring graphs frugally*, Combinatorica, to appear.
[10] T. Jensen and B. Toft, *Graph Colouring Problems*, Wiley, New York, 1995.
[11] K. Kilakos and B. Reed, *Fractionally colouring total graphs*, Combinatorica, 13 (1993), pp. 435–440.
[12] L. Lovász and M. Plummer, *Matching Theory*, Ann. Discrete Math. 29, North–Holland, Amsterdam, 1986.
[13] S. Micali and V. Vazirani, *An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs*, Proceedings of the 21st Annual Symposium on the Foundations of Computer Science, 1980, pp. 17–27.
[14] M. Molloy and B. Reed, *A bound on the total chromatic number*, Combinatorica, to appear.
[15] V. Vizing, *Some unsolved problems in graph theory*, Russian Math. Surveys, 23 (1968), pp. 125–141.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 822–840

COMPUTING COMPONENTS AND PROJECTIONS OF CURVES OVER FINITE FIELDS*

JOACHIM VON ZUR GATHEN† AND IGOR SHPARLINSKI‡

Abstract. This paper provides an algorithmic approach to some basic algebraic and combinatorial properties of algebraic curves over finite fields: the number of points on a curve or a projection, its number of absolutely irreducible components, and the property of being "exceptional."

Key words. curves over finite fields, computational algebraic geometry, approximation algorithms

AMS subject classifications. 11G20, 14H25, 68Q40

PII. S009753979427741X

1. Introduction. Let $\mathbb{F}_q$ be a finite field with q elements, $f \in \mathbb{F}_q[x,y]$ a bivariate polynomial of total degree n over $\mathbb{F}_q$, and $\mathcal{C} = \{f = 0\} = \{(u,v) \in \mathbb{F}_q^2 \colon f(u,v) = 0\} \subseteq \mathbb{F}_q^2$ the plane curve defined by f over $\mathbb{F}_q$. In this paper we present some algorithms to compute approximations to the curve size $\#\mathcal{C}$ and to the number r_i^* of points with exactly i preimages under the projection to a coordinate axis. Since this task generalizes Weil's estimate of $\#\mathcal{C}$, it might be called a "computational Weil estimate."

In [7], a "strip-counting" method was introduced. It is based on the general principle that the behavior of a curve can be deduced from its behavior over a wide enough vertical strip.

To be specific, let $S \subseteq \mathbb{F}_q$, $i \in \mathbb{N}$, and $\mathcal{C}(S)$ be the set of $(u,v) \in S \times \mathbb{F}_q$ with $f(u,v) = 0$. Furthermore, $R_i(S)$ is the set of $u \in S$ for which there are exactly i values $v \in \mathbb{F}_q$ with $f(u,v) = 0$, $r_i(S) = \#R_i(S)$, $M(S)$ is the number of triples $(u,v,w) \in S^2 \times \mathbb{F}_q$ with $f(u-v,w) = 0$, and $t_i(S)$ is the number of pairs $(u,v) \in S^2$ for which there are exactly i values $w \in \mathbb{F}_q$ satisfying $f(u-v,w) = 0$.

The basic idea now is that for some properties of curves, we can find reasonably small sets S such that the above parameters are not too hard to compute, and give information about some of the global parameters we are interested in.

A completely different approach, pioneered in [19], leads to deterministic algorithms for computing the size of $\mathcal{C} \subseteq \mathbb{F}_p^2$ with time polynomial in $\log p$ (and exponential in the degree); see [17], [11]. A method for higher-dimensional varieties is in [12].

If the set S is given in some reasonable sense, e.g., if we have an efficient way to enumerate all elements of S, then we can compute $\#\mathcal{C}(S)$ and all $r_i(S)$ for $0 \le i \le n$ in time $O\tilde{}(|S| n \log q)$ (see Lemma 2.5 below). Thus $\#\mathcal{C}$ and r_i^* may be computed in exponential time of order $O\tilde{}(nq)$ by this "brute force" algorithm. Here, we use the "soft-Oh" notation: $A = O\tilde{}(B)$ if and only if $A = B(\log B + 2)^{O(1)}$.

Continuing the work in [7], we show that for certain small sets S, the numbers $q\#\mathcal{C}(S)/\#S$ or $qM(S)/\#S^2$, and $qr_i(S)/\#S$ or $qt_i(S)/\#S^2$ are rather good approx-

*Received by the editors November 18, 1994; accepted for publication (in revised form) February 26, 1997; published electronically September 14, 1998. An extended abstract of this paper has appeared in *Proc. ISAAC* '94, Beijing, P. R. China, Ding-Zhu Du and Xiang-Sun Zhang, eds., Lecture Notes in Comput. Sci. 834, Springer-Verlag, New York, 1994, pp. 297–305.
http://www.siam.org/journals/sicomp/28-3/27741.html

†Fachbereich Mathematik-Informatik, Universität-GH Paderborn, D-33098 Paderborn, Germany (gathen@uni-paderborn.de).

‡School of MPCE, Macquarie University, Sydney, NSW 2109, Australia (igor@mpce.mq.edu.au).

imations to the curve size $\#\mathcal{C}$ and the projection statistics r_i^*, respectively. (The quality of these approximations is described in detail below.) In particular, to estimate $\#\mathcal{C}$, this is true for random sets of cardinality of order n^3, for any set of size of order $n^2q^{1/2}$, and for a random shift of any set of size of order n^4. The latter is a positive answer to Question 7.2 of [7] and is an example of reducing the number of random choices required in probabilistic algorithms. These results motivate the strip-counting terminology, in that it is sufficient to count points in the strip $S \times \mathbb{F}_q$ over S.

We consider mainly the case of finite prime fields, but we also show how some results can be generalized to the case of general finite fields, and outline some difficulties that do not allow us to generalize all results.

From $r_i(S)$ and $t_i(S)$ for $0 \le i \le n$ (or their approximations) we can compute (or estimate) the numbers $\#\mathcal{C}(S)$ and $M(S)$, respectively, as

$$\#\mathcal{C}(S) = \sum_{1 \le i \le n} r_i(S)i, \quad M(S) = \sum_{1 \le i \le n} t_i(S)i. \tag{1.1}$$

A connection in the opposite direction is given in Lemma 2.2 below.

The more general problem about the number of $u \in \mathbb{F}_q$ for which the polynomial $f(u, y) \in \mathbb{F}_q[y]$ has a given "factorization pattern" can be reduced to calculating analogues of r_i^* in extensions of the ground field $\mathbb{F}_q$.

For a curve of the form $f(x, y) = x - h(y)$, with $h(y) \in \mathbb{F}_q[y]$, $r_0^* = 0$ is equivalent to h being a permutation polynomial over $\mathbb{F}_q$.

Throughout the paper, we use the following terminology. Let K be an algebraic closure of $\mathbb{F}_q$ and $\mathcal{X} \subseteq K^{m+1}$ be an algebraic curve over K, defined over $\mathbb{F}_q$, and $\mathcal{C} = \mathcal{X} \cap \mathbb{F}_q^{m+1}$ the $\mathbb{F}_q$-rational points on $\mathcal{X}$. Since we are only interested in set-theoretic (counting) properties of $\mathcal{C}$ (and not sheaf-theoretic ones), we assume that $\mathcal{X}$ is reduced and without embedded points; $\mathcal{X}$ may be reducible and have singular points. Most of our results deal with the case $m = 1$, where we assume that $\mathcal{C}$ (and $\mathcal{X}$) are given by some polynomial $f \in \mathbb{F}_q[x, y]$, as $\mathcal{C} = \{f = 0\}$. Since the curve is reduced, f is squarefree. In the proofs, certain fiber products of $\mathcal{C}$ occur. A further assumption, without loss of generality, is that $\mathcal{C} \subseteq \mathbb{F}_q^2$ contains no vertical lines; this is defined in section 2. We denote by σ the number of absolutely irreducible components, i.e., the number of irreducible components of $\mathcal{C}$ over K that are defined over $\mathbb{F}_q$, and we use parameters λ_i defined in (2.3), via the fiber power of the curve. Lemmas 2.1 and 2.3 show that we automatically get approximations of order $O(q^{1/2})$ (for n fixed) to $\#\mathcal{C}$ and r_i^*, respectively, from approximations to σ and λ_i. So we shall mainly concentrate on algorithms to compute the latter parameters. Moreover, it also follows from those lemmas that in order to determine σ and r_i^*, it is enough to get approximations to $\#\mathcal{C}$ and to r_i^* with absolute errors less than $q/2$ and $q/2n!$, respectively. We consider the following three important special cases: $\sigma = 0$ ("exceptional curves"), $\sigma = 1$ ("almost absolutely irreducible curves"), and $\lambda_0 = 0$ ("almost permutation curves").

Our algorithms address a fairly difficult problem and have the following properties:

- they are easy to state and implement;
- their proofs of correctness rely on deep results from arithmetical algebraic geometry.

Table 1 below summarizes our algorithmic results.

2. Some general results. We start by collecting some facts about curves over finite fields. The following inequality is a consequence of the famous Weil result and Lemma 2.2 of [7], which gives a bound for the number of points on intersections

TABLE 1

Computing various parameters for a curve in $\mathbb{F}_q^2$ of degree n: absolutely irreducible components, *λ_i (see (2.3)),* exceptional, one component, *and $\lambda_0 = 0$. The* time *is the number of operations in $\mathbb{F}_q$, and* random *the number of random elements; both in the $O^{\sim}$-sense. If random is* 0, *we have a deterministic algorithm. For all probabilistic algorithms, the error probability is at most δ, and $q \geq$ indicates the lower bound on q, in the $O^{\sim}$-sense. The* condition *is either Condition A from section* 4, *or that the field size be a prime p.*

parameter	time	random	$q \geq$	cond	alg
comp	$n^4 \log \delta^{-1} \log q$	$n^3 \log \delta^{-1}$	n^4		3.2
comp	$n^5 \delta^{-2} \log q$	1	n^4		4.9
comp	$n^3 q^{1/2}$	0	n^4	A	4.7
λ_i	$(n!)^2 \log \delta^{-1} \log q$	$n!^2 \log \delta^{-1}$	$(n!)^2 n^{4n}$		3.8
λ_i	$(n!)^2 n^{4n} \delta^{-1} \log p$	1	$(n!)^2 n^{4n}$	p	4.13
λ_i	$n! n^{2n} p^{1/2}$	0	$(n!)^2 n^{4n}$	p	4.11
except	$n^3 \log \delta^{-1} \log q$	$n^2 \log \delta^{-1}$	n^4		3.4
one comp	$n^3 \log \delta^{-1} \log q$	$n^2 \log \delta^{-1}$	n^4		3.6
$\lambda_0 = 0$	$n! \log \delta^{-1} \log q$	$n! \log \delta^{-1}$	$(n!)^2 n^{4n}$		3.10

of absolutely irreducible curves and for the number of points on irreducible but not absolutely irreducible curves.

LEMMA 2.1. *Let $\mathcal{C} \subseteq \mathbb{F}_q^{m+1}$ be a curve of degree n over $\mathbb{F}_q$ with σ absolutely irreducible components defined over $\mathbb{F}_q$. Then*

$$\bigl|\#\mathcal{C} - \sigma q\bigr| \leq n^2 q^{1/2}.$$

Proof. Let $\mathcal{C}_1, \ldots, \mathcal{C}_\tau$ be the irreducible components of $\mathcal{C}$ over $\mathbb{F}_q$, with $\mathcal{C}_i$ absolutely irreducible if and only if $i \leq \sigma$. From the proof of Lemma 2.2 in [7], we find

$$\begin{aligned}
|\#\mathcal{C} - \sigma q| &\leq \Biggl|\#\mathcal{C} - \sum_{1 \leq i \leq \sigma} \#\mathcal{C}_i\Biggr| + \sum_{1 \leq i \leq \sigma} \bigl|\#\mathcal{C}_i - q\bigr| \\
&< \sum_{1 \leq i < j \leq \sigma} n_i n_j + \sum_{\sigma < i \leq \tau} n_i^2/4 + q^{1/2} + \sum_{1 \leq i \leq \sigma} n_i^2 \\
&\leq \Biggl(\sum_{1 \leq i \leq \tau} n_i\Biggr)^2 q^{1/2} \leq n^2 q^{1/2}. \qquad \square
\end{aligned}$$

Let $\mathcal{C} \subset \mathbb{F}_q^{m+1}$ be a curve. Throughout this paper, we assume that $\mathcal{C}$ is *without vertical components*, i.e., no absolutely irreducible component of $\mathcal{C}$ is contained in a hyperplane $\{a\} \times \mathbb{F}_q^m$, for some $a \in \mathbb{F}_q$. For a plane curve $\mathcal{C} = \{f = 0\}$, with $f = \sum_i f_i y^i \in \mathbb{F}_q[x, y]$ and all $f_i \in \mathbb{F}_q[x]$, this is the case if and only if $\gcd(f_0, f_1, \ldots) = 1$. In that case, we also say that $\mathcal{C}$ is *without vertical lines*. For the computational problems we consider, the general case is easily reduced to this (slightly) restricted one.

Furthermore, let $i \in \mathbb{N}$ and $S \subseteq \mathbb{F}_q$. We consider the difference map $\delta\colon S^2 \to \mathbb{F}_q$ with $\delta(u_1, u_2) = u_1 - u_2$, and denote by id the identity on $\mathbb{F}_q^m$. We define the following:

$$\begin{aligned}
\mathcal{C}(S) &= \mathcal{C} \cap (S \times \mathbb{F}_q^m) \;=\; \{(u, v)\colon (u, v) \in \mathcal{C},\ u \in S\} \subseteq \mathbb{F}_q^{m+1}, \\
R_i(S) &= \{u \in S\colon \#\mathcal{C}(\{u\}) = i\}, \\
r_i(S) &= \#R_i(S), \\
\mathcal{C}^\delta(S) &= (\delta \times \mathrm{id})^{-1}(\mathcal{C}) = \{(u_1, u_2, v)\colon (u_1 - u_2, v) \in \mathcal{C},\ u_1, u_2 \in S\},
\end{aligned}$$

$$M(S) = \#\mathcal{C}^{\delta}(S),$$
$$t_i(S) = \#\delta^{-1}(R_i(S)).$$

We also set $r_i^* = r_i(\mathbb{F}_q)$ and $t_i^* = t_i(\mathbb{F}_q)$. All these definitions coincide with the ones in the introduction if $\mathcal{C}$ is a plane curve. For a plane curve $\mathcal{C}$ given by $f \in \mathbb{F}_q[x,y]$ and $1 \le k \le n$, we define the curve $\mathcal{C}_k \subseteq \mathbb{F}_q^{k+1}$ as the closure of

$$\begin{aligned} S_k \;=\; & \{(u, v_1, \dots, v_k) \in \mathbb{F}_q^{k+1} : \\ & f(u,v_1) = \cdots = f(u,v_k) = 0,\ v_i \neq v_j \text{ for } 1 \le i < j \le k\}. \end{aligned} \tag{2.1}$$

To define this closure of S_k, we take the set X of all points in K^{k+1} satisfying the equations and inequalities in (2.1), its (Zariski-) closure $\overline{X}$ (i.e., all points satisfying all polynomials over K that vanish on X), and then $\mathcal{C}_k = \overline{X} \cap \mathbb{F}_q^{k+1}$. The geometry of $\mathcal{C}_2$ and the equations defining it as a complete intersection are described in detail in [8] and an example is given below. $\mathcal{C}_k$ is the k-fold fiber power of $\mathcal{C}$ along the first projection; it may be empty. Applying Bézout's theorem to the equations in (2.1), we find deg $\mathcal{C}_k \le n^k$; in fact, deg $\mathcal{C}_k \le n(n-1)\cdots(n-k+1)$. It can, of course, also be defined for curves in $\mathbb{F}_q^{m+1}$ with $m > 1$. The following statement is essentially Lemma 3.2 of [7].

LEMMA 2.2. *For a plane curve $\mathcal{C} \subseteq \mathbb{F}_q^2$ without vertical lines and of degree n, $S \subseteq \mathbb{F}_q$, and $0 \le i \le n$, we have*

$$r_i(S) = \frac{1}{i!} \sum_{i \le k \le n} \frac{(-1)^{i+k}\, \#\mathcal{C}_k(S)}{(k-i)!}, \tag{2.2}$$
$$t_i(S) = \frac{1}{i!} \sum_{i \le k \le n} \frac{(-1)^{i+k}\, \#\mathcal{C}_k^{\delta}(S)}{(k-i)!}.$$

In view of these expressions, we consider the number σ_k of absolutely irreducible components defined over $\mathbb{F}_q$ of $\mathcal{C}_k$, with $\sigma_0 = 1$, and for $0 \le i \le n$ set

$$\lambda_i = \frac{1}{i!} \sum_{i \le k \le n} \frac{(-1)^{i+k}\sigma_k}{(k-i)!} \in \mathbb{Q}. \tag{2.3}$$

LEMMA 2.3. *Let $\mathcal{C} \subseteq \mathbb{F}_q^2$ be a curve without vertical lines given by $f \in \mathbb{F}_q[x,y]$ of degree n, and $\lambda_0, \dots, \lambda_n$ as above. Then for $0 \le i \le n$, we have $n!\lambda_i \in \mathbb{Z}$, and*

$$|r_i^* - \lambda_i q| \le 2n^{2n} q^{1/2}. \tag{2.4}$$

Proof. Noting that $\mathcal{C}_k$ is of degree at most n^k and using σ_k as above, we find from Lemmas 2.1 and 2.2 that

$$\begin{aligned} \left| r_i^* - q\lambda_i \right| &= \frac{1}{i!} \left| \sum_{i \le k \le n} \frac{(-1)^{i+k}(\#\mathcal{C}_k - \sigma_k q)}{(k-i)!} \right| \\ &\le \frac{q^{1/2}}{i!} \sum_{i \le k \le n} \frac{n^{2k}}{(k-i)!} \le \frac{q^{1/2}(n^2)^{n+1} - 1}{n^2 - 1} \le 2n^{2n} q^{1/2} \end{aligned}$$

for $n \ge 2$; the case $n = 1$ is trivial. Furthermore $i!(k-i)!$ divides $n!$ for all $0 \le i \le k \le n$. ∎

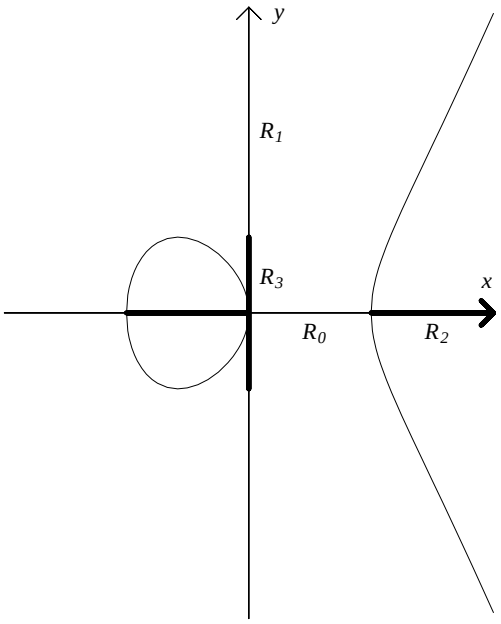


FIG. 1. *The elliptic curve $y^2 = x^3 - x$ over $\mathbf{R}$, the sets R_0 and R_2 for the first projection, and the sets R_1 and R_3 for the second projection.*

EXAMPLE 2.4. *We take the (irreducible) elliptic curve $\mathcal{C} = \{f = 0\}$ of degree $n = 3$ given by $f = y^2 - x^3 + x \in \mathbb{F}_q[x, y]$, where $q = 1019$ is prime. Figure* 1 *gives the picture over $\mathbb{R}$. Then $\mathcal{C}_2$ is given by the equations*

$$-u^3 + u = v_1^2, v_1 + v_2 = 0.$$

The latter equation comes from eliminating u and dividing the result $v_1^2 - v_2^2$ by $v_1 - v_2$. Thus $\mathcal{C}_2$ is isomorphic to $\mathcal{C}$ and irreducible, and $\sigma_2 = 1$. Furthermore, $\mathcal{C}_3 = \emptyset$ and $\sigma_3 = 0$, so that

$$\lambda_0 = \frac{1}{2}, \quad \lambda_1 = 0, \quad \lambda_2 = \frac{1}{2}, \quad \lambda_3 = 0.$$

For the two sets $S_1 = \mathbb{F}_q$ and $S_2 = \{0, 1, 2, \ldots, 49\}$ we find

i	$r_i(S_1) = r_i^*$	$\#\mathcal{C}_i(S_1)$	$r_i^* - \lambda_i q$	$r_i(S_2)$	$\#\mathcal{C}_i(S_2)$
0	508	1019	-1.5	26	50
1	3	1019	3	2	2
2	508	1016	-1.5	22	22
3	0	0	0	0	0

Of course, (2.2) *could have been used to predict the r_i^* approximately. The pessimistic bound* (2.4) *actually holds with the error term $3 < 46{,}541.95 \approx 2n^{2n}q^{1/2}$. As expected from the picture of the curve $\{y^2 = x^3 - x\}$ over $\mathbb{R}$, there are (almost) no points with* 1 *or* 3 *preimages under the first projection. The other two possibilities, of* 0 *or* 2 *preimages, occur equally often. The three points with one preimage are $0, 1, -1$.*

It is instructive to also look at the projection of $\mathcal{C}$ onto the y-axis. To preserve terminology, we thus take $f = x^2 - y^3 + y$. Then $\mathcal{C}_2$ is

$$\mathcal{C}_2 = \{(u, v_1, v_2) \in \mathbb{F}_q^3 \colon f(u, v_1) = 0, v_1^2 + v_1 v_2 + v_2^2 - 1 = 0\}. \tag{2.5}$$

$\mathcal{C}_2$ is irreducible, and $\sigma_2 = 1$. For $\mathcal{C}_3$, we have to add the equation

$$v_1 + v_2 + v_3 = 0$$

to those in (2.5); *thus $\mathcal{C}_3 \cong \mathcal{C}_2$ and $\sigma_3 = 1$. We find*

$$\lambda_0 = \frac{1}{3}, \lambda_1 = \frac{1}{2}, \lambda_2 = 0, \lambda_3 = \frac{1}{6},$$

and with S_1 and S_2 as above, we have

i	$r_i(S_1) = r_i^*$	$\#\mathcal{C}_i(S_1)$	$r_i^* - \lambda_i q$	$r_i(S_2)$	$\#\mathcal{C}_i(S_2)$
0	340	1019	0.33	14	50
1	508	1019	−1.5	24	60
2	2	1020	2	0	72
3	169	1020	−0.83	12	72

Again, (2.4) *holds with the bound* $2 < 46{,}541.95$. *This example shows how the λ_i's comprise in a concise way reasonably good information about the projection statistics of the curve. Note that in the picture over $\mathbb{R}$ the set corresponding to R_0 is empty.*

We denote by $\mathsf{M}(n)$ the Boolean complexity of multiplication of two n-bit numbers. The currently best estimate [18] of this function is

$$\mathsf{M}(n) = O(n \log n \operatorname{loglog} n).$$

As in the proof of Lemma 2.5 of [7], we find the following result.

LEMMA 2.5. *Let $\mathcal{C} \subseteq \mathbb{F}_q^2$ be without vertical lines and given by $f \in \mathbb{F}_q[x, y]$ of degree n, and $u \in \mathbb{F}_q$. Then $\#\mathcal{C}(\{u\})$ can be computed with $O(\mathsf{M}(n) \log(nq))$ arithmetic operations in $\mathbb{F}_q$.*

3. Counting with random elements. Throughout this section, $\mathcal{C}$ is a plane curve without vertical lines. We extend our notions $\mathcal{C}(S)$, $R_i(S)$, $r_i(S)$ to a sequence $S = (s_1, \ldots, s_h)$ of elements of $\mathbb{F}_q$ in the obvious way, e.g., we set $\#\mathcal{C}(S) = \sum_{1 \le i \le h} \#\mathcal{C}(\{s_i\})$. In particular, when S is a sequence of random elements of $\mathbb{F}_q$, $\#\mathcal{C}(S)$ and $r_i(S)$ are random variables. We state our algorithms in this section for a sequence of h random elements, because for a computer implementation such a sequence is slightly more natural than a random subset of size h; the results also hold for such a random subset.

The following bound on the difference between a sample mean and the true expected value is a direct consequence of the general result of [13] (see also Theorem 7.2 of [7]) and the trivial bounds $\#\mathcal{C} \le nq$ and $r_i^* \le q$.

LEMMA 3.1. *Let S be a sequence of h independently and uniformly distributed random elements of $\mathbb{F}_q$, $0 \le i \le n$, and $\delta > 0$. Then the following hold with probability at least $1 - \delta$:*

$$\begin{aligned} |\#\mathcal{C} - q\#\mathcal{C}(S)h^{-1}| &\le \left(2n(n+1)q\#\mathcal{C} \log(2n/\delta)h^{-1}\right)^{1/2} \\ &\le nq\left(2(n+1)\log(2n/\delta)h^{-1}\right)^{1/2}, \end{aligned}$$

$$|r_i^* - qr_i(S)h^{-1}| \le 2\left(qr_i^* \log(2/\delta)h^{-1}\right)^{1/2} \le 2q\left(\log(2/\delta)h^{-1}\right)^{1/2}.$$

ALGORITHM 3.2. *Components.*
Input: $f \in \mathbb{F}_q[x,y]$ *of degree* n, *and* $\delta > 0$.
Output: *An estimate of the number of absolutely irreducible components of* $\mathcal{C} = \{f = 0\}$ *defined over* $\mathbb{F}_q$.

1. *Set* $h = \lceil 72n^2(n+1)\log(2n/\delta)\rceil$.
2. *Choose a sequence* S *of* h *random independently uniformly distributed elements of* $\mathbb{F}_q$.
3. *Compute* $\#\mathcal{C}(S)$.
4. *Return the nearest integer to* $\#\mathcal{C}(S)/h$.

THEOREM 3.3. *Assume that* $q \geq 36n^4$. *Then Algorithm* 3.2 *outputs the number* σ *of absolutely irreducible components correctly with probability at least* $1-\delta$. *It uses* $O(n^3\log(n/\delta))$ *random elements and* $O(n^3\mathsf{M}(n)\log(n/\delta)\log(nq))$ *arithmetic operations in* $\mathbb{F}_q$.

Proof. The cost bound follows from Lemma 2.5, and Lemmas 2.1 and 3.1 show that

$$|\sigma - \#\mathcal{C}(S)h^{-1}| \leq n^2 q^{-1/2} + n\left(2(n+1)\log(2n/\delta)h^{-1}\right)^{1/2} \leq 1/6 + 1/6 = 1/3$$

with probability at least $1-\delta$. □

We call a curve $\mathcal{C}$ over $\mathbb{F}_q$ *exceptional* (over $\mathbb{F}_q$) if and only if none of the irreducible components of $\mathcal{C}$ defined over $\mathbb{F}_q$ is absolutely irreducible. In particular, a plane curve $\mathcal{C} = \{f = 0\}$ with $f \in \mathbb{F}_q[x,y]$ is exceptional if and only if none of the irreducible factors of f over $\mathbb{F}_q$ is absolutely irreducible. This notion plays a central role in the study of permutation polynomials: $g \in \mathbb{F}_q[x]$ is a permutation polynomial if and only if $(g(x)-g(y))/(x-y)$ is exceptional provided that $q \geq 16(\deg g)^4$ [4], [5].

ALGORITHM 3.4. *Exceptional test.*
Input: $f \in \mathbb{F}_q[x,y]$ *of degree* n, *and* $\delta > 0$.
Output: *YES if* f *is exceptional, and NO otherwise.*

1. *Set* $h = \lceil 16n(n+1)\log(2n/\delta)\rceil$.
2. *Choose a sequence* S *of* h *random independently uniformly distributed elements of* $\mathbb{F}_q$.
3. *Compute* $\#\mathcal{C}(S)$.
4. *If* $\#\mathcal{C}(S) \leq n^2/4$ *then return YES else return NO.*

THEOREM 3.5. *Assume that* $q \geq 4n^4$. *If* f *is exceptional, Algorithm* 3.4 *answers correctly. If* f *is not exceptional, Algorithm* 3.4 *answers correctly with probability at least* $1-\delta$. *It uses* $O(n^2\log(n/\delta))$ *random elements and* $O(n^2\mathsf{M}(n)\log(n/\delta)\log(nq))$ *arithmetic operations in* $\mathbb{F}_q$.

Proof. Let σ be the required number of components. If $\sigma = 0$, then $\#\mathcal{C} \leq n^2/4$; see Lemma 5.2(ii) of [7] for an example. Thus the algorithm answers correctly in this case. It is sufficient to estimate the probability that $\#\mathcal{C}(S) \leq n^2$ when $\sigma \geq 1$. From Lemma 2.1 we get

$$\#\mathcal{C} \geq q - n^2 q^{1/2} \geq q/2.$$

Assuming that $\delta \leq 1$, Lemma 3.1 implies that with probability at least $1-\delta$ we have

$$\begin{aligned}\#\mathcal{C}(S) &\geq \frac{h\#\mathcal{C}}{q} - \frac{h}{q}\left(2n(n+1)q\#\mathcal{C}\log(2n/\delta)h^{-1}\right)^{1/2} \\ &\geq \frac{h\#\mathcal{C}}{q}\left(1 - \left(\frac{q}{\#\mathcal{C}}\cdot\frac{1}{8}\right)^{1/2}\right) \;\geq\; \frac{h}{2}\left(1 - \left(\frac{1}{4}\right)^{1/2}\right) \;=\; \frac{h}{4} \;>\; n^2. \qquad \square\end{aligned}$$

ALGORITHM 3.6.
Input: $f \in \mathbb{F}_q[x,y]$ *of degree* n, *and* $\delta > 0$.
Output: *YES if* $\mathcal{C} = \{f = 0\}$ *has exactly one absolutely irreducible component defined over* $\mathbb{F}_q$, *and NO otherwise.*

1. *Return NO if* f *is exceptional, using Algorithm* 3.4 *with input* $(f, \delta/2)$.
2. *Set* $h = \lceil 90n(n+1)\log(4n/\delta)\rceil$.
3. *Choose a sequence* S *of* h *random independent uniformly distributed elements of* $\mathbb{F}_q$.
4. *Compute* $\#\mathcal{C}(S)$.
5. *If* $\#\mathcal{C}(S) \le 17h/12$ *then return YES else return NO.*

THEOREM 3.7. *Let* $q \ge 16n^4$. *With probability at least* $1-\delta$, *Algorithm* 3.6 *decides correctly whether* $\mathcal{C}$ *has exactly one absolutely irreducible component defined over* $\mathbb{F}_q$. *It uses* $O(n^2\log(n/\delta))$ *random elements and* $O(n^2\mathsf{M}(n)\log(n/\delta)\log(nq))$ *arithmetic operations in* $\mathbb{F}_q$.

Proof. Let σ be the number of components. The cost estimate follows from Lemma 2.5. We may assume that $\sigma \ge 1$, and have to bound the error probability. If $\sigma = 1$, then we get from Lemma 2.1 that

$$\#\mathcal{C} \le q + n^2 q^{1/2} \le 5q/4,$$

and from Lemma 3.1 that

$$\begin{aligned}\#\mathcal{C}(S) &\le h\#\mathcal{C}/q + \frac{h}{q}\left(2n(n+1)q\#\mathcal{C}\log(4n/\delta)h^{-1}\right)^{1/2}\\ &\le \frac{5h}{4} + h\left(\frac{5}{4\cdot 45}\right)^{1/2} = \frac{17h}{12}\end{aligned}$$

with probability at least $1-\delta/2$. Otherwise,

$$7q/4 \le 2q - n^2q^{1/2} \le \#\mathcal{C} \le 2q + n^2q^{1/2} \le 9q/4,$$

and with probability at least $1-\delta/2$

$$\begin{aligned}\#\mathcal{C}(S) &\ge \frac{h\#\mathcal{C}}{q} - \frac{h}{q}\left(2n(n+1)q\#\mathcal{C}\cdot\log(4n/\delta)h^{-1}\right)^{1/2}\\ &\ge \frac{7h}{4} - h\cdot\left(\frac{9}{4\cdot 45}\right)^{1/2} = \frac{(35-2\sqrt{5})h}{20} > \frac{17h}{12}.\end{aligned}$$

□

Our next algorithm computes the rational numbers $\lambda_0,\dots,\lambda_n$. In view of (2.3), this is equivalent to calculating $\sigma_0,\dots,\sigma_n$ up to a triangular system of linear equations; we do not know a direct easy way to compute these σ_i's.

ALGORITHM 3.8.
Input: $f \in \mathbb{F}_q[x,y]$ *of degree* n, *and* $\delta > 0$.
Output: *The parameters* $\lambda_0,\dots,\lambda_n$ *of* $\mathcal{C} = \{f = 0\}$ *as defined in* (2.3).

1. *Set* $h = \lceil 144(n!)^2\log(2/\delta)\rceil$.
2. *Choose a sequence* S *of* h *random independently uniformly distributed elements of* $\mathbb{F}_q$.
3. *For* $i = 1,\dots,n$ *do steps* 4, 5, 6.
4. *Compute* $r_i(S)$.
5. *Compute the nearest integer* Λ_i *to* $n!r_i(S)/h$.
6. Return $\Lambda_i/n!$.

THEOREM 3.9. *If $q \geq 144n^{4n}(n!)^2$, then Algorithm* 3.8 *computes the parameters $\lambda_0, \ldots, \lambda_n$ of $\mathcal{C} = \{f = 0\}$ correctly with probability at least $1 - \delta$. It uses $O((n!)^2 \log(\delta^{-1}))$ random elements, and $O((n!)^2 \mathsf{M}(n) \log(\delta^{-1}) \log(nq))$ arithmetic operations in $\mathbb{F}_q$.*

Proof. From Lemmas 2.3 and 3.1, we find that with probability at least $1 - \delta$

$$|\lambda_i - r_i(S)h^{-1}| \leq 2n^{2n}q^{-1/2} + 2\big(\log(2/\delta)h^{-1}\big)^{1/2} < 1/6n! + 1/6n! < 1/3n!,$$

and in this case the output is correct. The cost estimate follows from Lemma 2.5. □

Together with Lemma 2.3, this gives an estimate for the r_i's, and, with (1.1) for $S = \mathbb{F}_q$, also for $\#\mathcal{C}$.

Let us now consider the special case of testing if $\lambda_0 = 0$. For a curve of the form $f = y - g(x)$ with $g \in \mathbb{F}_q[x]$, the condition $\lambda_0 = 0$ implies $r_0 = 0$ (at least for q large enough), i.e., that h is a permutation polynomial (see [5] for details).

ALGORITHM 3.10.
Input: *$f \in \mathbb{F}_q[x, y]$ of degree n, and $\delta > 0$.*
Output: *YES if $\lambda_0 = 0$ for $\mathcal{C} = \{f = 0\}$, else NO.*

1. *Set $h = \lceil 256(n!)^2 \log(2/\delta) \rceil$.*
2. *Choose a sequence S of h random independently uniformly distributed elements of $\mathbb{F}_q$.*
3. *Compute $\#\mathcal{C}(S)$.*
4. *Return YES if $\#\mathcal{C}(S) \leq h/4n!$, else NO.*

THEOREM 3.11. *If $q \geq 256n^{4n}(n!)^2$, then the output of Algorithm* 3.10 *is correct with probability at least $1 - \delta$. It uses $O(n! \log(\delta^{-1}))$ random elements and $O(n! \mathsf{M}(n) \log(\delta^{-1}) \log(nq))$ arithmetic operations in $\mathbb{F}_q$.*

Proof. The cost estimate follows from Lemma 2.5. To bound the error probability, we have from Lemma 2.3 that

$$\lambda_0 q - q/8n! \leq \lambda_0 q - 2n^{2n}q^{1/2} \leq r_0^* \leq \lambda_0 q + 2n^{2n}q^{1/2} \leq \lambda_0 q + q/8n!.$$

If $\lambda_0 = 0$, then we find from Lemma 3.1 that with probability at least $1 - \delta$

$$r_0(S) \leq hr_0^*/q + 2\big(h \log(2/\delta)\big)^{1/2} \leq h/8n! + h/8n! = h/4n!.$$

Now suppose that $\lambda_0 \neq 0$. Then $|\lambda_0| \geq 1/n!$. Furthermore, $\lambda_0 < 0$ would imply that $0 \leq r_0^* \leq -q/n! + q/8n! < 0$. Thus $\lambda_0 \geq 1/n!$, and with probability at least $1 - \delta$

$$r_0(S) \geq hr_0^*/q - 2\big(h \log(2/\delta)\big)^{1/2} \geq 7h/8n! - h/8n! = 3h/4n!. \qquad \square$$

4. Counting in additive strips. In this section, we continue to study properties of curves in additive strips. Our main tool is the bound from [1] on exponential sums along a curve. The various estimates we obtain give methods for approximating $\#\mathcal{C}$ and the r_i's, at least when q is large relative to n.

For integers a and h, we denote by $A(a, h)$ the interval

$$A(a, h) = \{(a + j) \bmod p \colon 0 \leq j < h\} \subseteq \mathbb{F}_q,$$

where $p = \operatorname{char} \mathbb{F}_q$, and for a curve $\mathcal{C} \subseteq \mathbb{F}_q^{m+1}$, we write

$$\mathcal{C}(a, h) = \mathcal{C}(A(a, h)), \qquad r_i(a, h) = r_i(A(a, h)),$$

$$M(a,h) = M(A(a,h)), \qquad t_i(a,h) = t_i(A(a,h)).$$

It follows from Lemma 2.1 of [7] that if x is not a constant along any absolutely irreducible component of $\mathcal{C}$ and $n = \deg\ \mathcal{C}$, then for any integers a and $h \le p$,

$$\bigl|\#\mathcal{C}(a,h) - h\#\mathcal{C}/p\bigr| \le 2n^2 p^{1/2}\log p. \tag{4.1}$$

Let K be an algebraic closure of $\mathbb{F}_q$. We will repeatedly use the following assumption on a curve $\mathcal{C} \subseteq \mathbb{F}_q^{m+1}$, which arises in Bombieri's work.

HYPOTHESIS A. *For every absolutely irreducible component $\mathcal{D}$ of $\mathcal{C}$ and every rational function g on K^{m+1}, x is different from $g^p - g$ on $\mathcal{D}$, where $p = \operatorname{char}\mathbb{F}_q$.*

In general, given the equations for $\mathcal{C}$, it seems not easy to check whether $\mathcal{C}$ satisfies Hypothesis A. If $x = g^p - g = \prod_{u\in\mathbb{F}_p}(g-u)$, then each $g-u$ has the same poles as x, and in particular the degree of the pole divisor of x is divisible by p. Thus

$$\deg\mathcal{C} < p \quad\Longrightarrow\quad \mathcal{C} \text{ satisfies Hypothesis A;} \tag{4.2}$$

see also Lemma 4 of [2].

Below we show that for the parameter $\#\mathcal{C}^\delta(S)$ a slightly stronger result than (4.1) holds for an arbitrary set $S \subseteq \mathbb{F}_q$.

LEMMA 4.1. *Let $\mathcal{C} = \mathbb{F}_q^{m+1}$ be a curve without vertical components and of degree n satisfying Hypothesis A, $S \subseteq \mathbb{F}_q$, and $h = \#S$. Then*

$$\bigl|\#\mathcal{C}^\delta(S) - h^2\#\mathcal{C}/q\bigr| < 2n^2 h q^{1/2}.$$

Proof. Let χ be a nontrivial additive character of $\mathbb{F}_q$. Then

$$\bigl|\#\mathcal{C}^\delta(S)\bigr| = \frac{1}{q}\sum_{(a,b)\in\mathcal{C}}\ \sum_{u,v\in S}\ \sum_{\lambda\in\mathbb{F}_q}\chi\bigl(\lambda(a-u+v)\bigr) = h^2\#\mathcal{C}/q + t/q,$$

where $a \in \mathbb{F}_q$ and $b \in \mathbb{F}_q^m$ in the sum, and

$$t = \sum_{\lambda\in\mathbb{F}_q^\times}\ \sum_{(a,b)\in\mathcal{C}}\chi(\lambda a)\sum_{u,v\in S}\chi(\lambda(u-v)).$$

The bound of [1, Theorem 6] implies that for $\lambda \in \mathbb{F}_q^\times$,

$$\Bigl|\sum_{(a,b)\in\mathcal{C}}\chi(\lambda a)\Bigr| \le (n^2-n)q^{1/2} + n^2 < 2n^2 q^{1/2}.$$

Therefore

$$\begin{aligned} t &< 2n^2 q^{1/2}\Bigl|\sum_{\lambda\in\mathbb{F}_q^\times}\ \sum_{u,v\in S}\chi(\lambda(u-v))\Bigr| \\ &= 2n^2 q^{1/2}\Bigl|\sum_{u,v\in S}\ \sum_{\lambda\in\mathbb{F}_q}\chi\bigl(\lambda(u-v)\bigr) - h^2\Bigr|. \end{aligned}$$

Since the inner sum equals 0 when $u \ne v$ and q otherwise, we get

$$t < 2n^2 h q^{3/2}. \qquad \square$$

We note that this lemma is nontrivial for sets of cardinality $h \ge 2n^2q^{1/2}$, while the above-mentioned result from [7] works only in case of a prime field $\mathbb{F}_q = \mathbb{F}_p$ and needs $h \ge 2n^2p^{1/2}\log p$.

For $w \in \mathbb{F}_q$ and $S \subseteq \mathbb{F}_q$, we denote by S_w the w-shift of S:

$$S_w = \{w + u : u \in S\}.$$

The following lemma shows that $q\#\mathcal{C}(S_w)/h$ is a good approximation to $\#\mathcal{C}$ for almost all w-shifts of any set $S \subseteq \mathbb{F}_q$ with $\#S \gg n^3$.

LEMMA 4.2. *Let $\mathcal{C} \subseteq \mathbb{F}_q^{m+1}$ be a curve without vertical components and of degree n satisfying Hypothesis A, $w \in \mathbb{F}_q$, $S \subseteq \mathbb{F}_q$, $h = \#S$, and*

$$s = \frac{1}{q}\sum_{w\in\mathbb{F}_q}\big(\#\mathcal{C}(S_w) - h\#\mathcal{C}/q\big)^2.$$

Then $s \le 4n^4h$, and if $q \ge n^2$, then $s \le n^4h$.

Proof. Let χ be a nontrivial additive character on $\mathbb{F}_q$. We have, as in the proof of the previous lemma,

$$\#\mathcal{C}(S_w) - h\#\mathcal{C}/q = \frac{1}{q}\sum_{\lambda\in\mathbb{F}_q^\times}\sum_{(a,b)\in\mathcal{C}}\chi(\lambda a)\sum_{u\in S}\chi\big(-\lambda(w+u)\big),$$

where $a \in \mathbb{F}_q$ and $b \in \mathbb{F}_q^m$. Hence

$$\sum_{w\in\mathbb{F}_q}\big|\#\mathcal{C}(S_w) - h\#\mathcal{C}/q\big|^2 = tq^{-2},$$

where

$$\begin{aligned} t &= \sum_{w\in\mathbb{F}_q}\Big|\sum_{\lambda\in\mathbb{F}_q^\times}\sum_{(a,b)\in\mathcal{C}}\chi(\lambda a)\sum_{u\in S}\chi\big(-\lambda(w+u)\big)\Big|^2 \\ &= \sum_{\lambda_1,\lambda_2\in\mathbb{F}_q^\times}\sum_{(a_1,b_1),(a_2,b_2)\in\mathcal{C}}\chi(\lambda_1a_1 - \lambda_2a_2)\sum_{u_1,u_2\in S}\chi(-\lambda_1u_1 + \lambda_2u_2) \\ &\quad\times\sum_{w\in\mathbb{F}_q}\chi\big(w(-\lambda_1+\lambda_2)\big). \end{aligned}$$

Since the last sum equals 0 when $\lambda_1 \ne \lambda_2$ and q otherwise, we find from Theorem 6 of [1] that

$$\begin{aligned} t &= q\sum_{\lambda\in\mathbb{F}_q^\times}\sum_{(a_1,b_1),(a_2,b_2)\in\mathcal{C}}\chi\big(\lambda(a_1-a_2)\big)\sum_{u_1,u_2\in S}\chi\big(\lambda(-u_1+u_2)\big) \\ &= q\sum_{\lambda\in\mathbb{F}_q^\times}\Big|\sum_{(a,b)\in\mathcal{C}}\chi(\lambda a)\Big|^2\sum_{u_1,u_2\in S}\chi\big(\lambda(-u_1+u_2)\big) \\ &\le q\big((n^2-n)q^{1/2}+n^2\big)^2\sum_{\lambda\in\mathbb{F}_q^\times}\sum_{u_1,u_2\in S}\chi\big(\lambda(u_1-u_2)\big) \\ &\le 4n^4q^2\Big|\sum_{\lambda\in\mathbb{F}_q}\sum_{u_1,u_2\in S}\chi(\lambda(u_1-u_2)) - h^2\Big|. \end{aligned}$$

We can replace 4 by 1 if $n^2 \le q$. The sum is zero when $u_1 \neq u_2$ and q otherwise, so that

$$t \le 4n^4q^2(qh - h^2) \le 4n^4hq^3. \qquad \square$$

COROLLARY 4.3. *Let $\mathcal{C} \subseteq \mathbb{F}_q^{m+1}$ be a curve without vertical components and of degree n satisfying Hypothesis A, $\delta > 0$, $S \subseteq \mathbb{F}_q$, and $h = \#S$. Then*

$$\big|\#\mathcal{C}(S_a) - h\#\mathcal{C}/q\big| \le 2\delta^{-1/2}n^2h^{1/2}$$

holds with probability at least $1 - \delta$ for random $a \in \mathbb{F}_q$.

LEMMA 4.4. *Let p be a prime, $\mathcal{C} \subseteq \mathbb{F}_p^2$ be a plane curve without vertical lines of degree n satisfying Hypothesis A, $0 \le i \le n$, $p > n^n$, and $a, h \in \mathbb{N}$ with $h \le p$. Then*

$$|t_i(a,h) - h^2r_i^*/p| \le 3n^{2n}hp^{1/2},$$
$$|r_i(a,h) - \ hr_i^*/p| \le 3n^{2n}p^{1/2}\log p.$$

Proof. For $1 \le k \le n$, we have deg $\mathcal{C}_k \le n^k < p$, and thus $\mathcal{C}_k$ satisfies Hypothesis A, by (4.2). Lemma 4.1 implies that

$$\big|\#\mathcal{C}_k^\delta(a,h) - h^2\#\mathcal{C}_k/p\big| \le 2n^{2k}hp^{1/2}.$$

Let $0 \le i \le n$. From Lemma 2.2, we have

$$|t_i(a,h) - h^2t_i^*/p| \le \frac{1}{i!}\sum_{i\le k\le n}\frac{\big|\#\mathcal{C}_k^\delta(a,h) - h^2\#\mathcal{C}_k/p\big|}{(k-i)!}$$
$$\le\ \frac{1}{i!}\sum_{i\le k\le n}\frac{2n^{2k}hp^{1/2}}{(k-i)!} \le 2n^{2n}hp^{1/2}\frac{n^2}{n^2-1}\ \le\ 3n^{2n}hp^{1/2}.$$

Using (4.1), the second bound follows in a similar way. $\square$

We next show that for almost all a a much stronger bound than the second estimate in Lemma 4.4 holds.

LEMMA 4.5. *Let $p > n^n$ be a prime, $\mathcal{C} \subseteq \mathbb{F}_p^2$ a plane curve without vertical lines and of degree n, $0 \le i \le n$, and $h \le p$. Then*

$$\frac{1}{p}\sum_{0\le a<p}\big(r_i(a,h) - hr_i^*/p\big)^2 \le 8n^{4n}h.$$

Proof. For $0 \le k \le n$, we have deg $\mathcal{C}_k \le n^k < p$, and $\mathcal{C}_k$ satisfies Hypothesis A by (4.2). Using Lemma 4.2, we find

$$\sum_{0\le a<p}\big(r_i(a,h) - hr_i^*/p\big)^2 \le \frac{1}{i!^2}\sum_{0\le a<p}\Big(\sum_{i\le k\le n}\frac{\big(\#\mathcal{C}_k(a,h) - h\#\mathcal{C}_k/p\big)}{(k-i)!}\Big)^2$$
$$\le \sum_{i\le k\le n}\ \sum_{0\le a<p}\big(\#\mathcal{C}_k(a,h) - h\#\mathcal{C}_k/p\big)^2$$
$$\le 4ph\sum_{i\le k\le n}n^{4k}\ \le\ 8n^{4n}ph. \qquad \square$$

COROLLARY 4.6. *Let $p > n^n$ be a prime, $\mathcal{C} \subseteq \mathbb{F}_p^2$ a plane curve without vertical lines and of degree n, $0 < \delta < 1$, and $h \le p$. Then*

$$|r_i(a,h) - hr_i^*/p| \le n^{2n}(8h\delta^{-1})^{1/2}$$

holds with probability at least $1-\delta$ *for a random element* $a \in \mathbb{F}_p$.

It was proved in [7] that the number of absolutely irreducible components of a plane curve $\mathcal{C} \subseteq \mathbb{F}_p^2$ of degree n can be determined with $O(n^2\mathsf{M}(n)p^{1/2}\log^2 p)$ arithmetic operations in $\mathbb{F}_p$. A similar result is true for r_i, namely, we can find the parameters λ_i as in (2.3) with $O(n!n^{2n}\mathsf{M}(n)p^{1/2}\log^2 p)$ arithmetic operations in $\mathbb{F}_p$. Indeed, choose

$$h = \lceil 18n!n^{2n}p^{1/2}\log p\rceil.$$

Setting $\Lambda_i = \lambda_i n! \in \mathbb{Z}$, we find from Lemmas 4.4 and 2.3

$$\begin{aligned}|n!\,r_i(0,h)h^{-1} - \Lambda_i| &\le n!h^{-1}|r_i(0,h) - hr_i^*p^{-1}| + n!p^{-1}|r_i^* - \lambda_i p|\\ &\le n!h^{-1}\cdot 3n^{2n}p^{1/2}\log p + n!p^{-1}\cdot 2n^{2n}p^{1/2}\\ &\le 1/6 + 1/6 \;=\; 1/3,\end{aligned}$$

if $p \ge 144n^{4n}(n!)^2$. Thus we may determine Λ_i as the nearest integer to $n!r_i(0,h)h^{-1}$.

Below we show how to improve this method and partially generalize it to the case of arbitrary finite fields.

ALGORITHM 4.7. *Deterministic components.*
Input: $f \in \mathbb{F}_q[x,y]$ *of degree* $n \le q^{1/4}/4$, *and a basis* $\omega_1,\dots,\omega_k$ *of* $\mathbb{F}_q$ *over* $\mathbb{F}_p$, *where* $p = \operatorname{char}\mathbb{F}_q$ *and* $q = p^k$.
Output: *The number of absolutely irreducible components of* $\mathcal{C} = \{f = 0\}$ *defined over* $\mathbb{F}_q$.

1. *Set* $H = \lceil 12n^2q^{1/2}\rceil$.
2. *Compute integers* l, h_0, *and* h *such that*

$$p^{l-1} \le H < p^l, \quad (h_0-1)p^{l-1} \le H < h_0p^{l-1}, \quad h = \min\{(p-1)/2, h_0\}.$$

3. *Set*

$$S = \{a_1\omega_1 + \cdots + a_l\omega_l \colon a_1,\dots,a_{l-1} \in \mathbb{F}_p,\ a_l \in A(0,h)\}.$$

4. *Compute* $M(S)$.
5. *Return the nearest integer to* $M(S)/\#S^2$.

THEOREM 4.8. *Let* $q > 256n^4$, *and* $\mathcal{C} \subseteq \mathbb{F}_q^2$ *be a plane curve without vertical lines and of degree* n *satisfying Hypothesis A. Algorithm* 4.7 *correctly computes the number of absolutely irreducible components of* $\mathcal{C}$. *It uses* $O(n^2\mathsf{M}(n)q^{1/2}\log q)$ *arithmetic operations in* $\mathbb{F}_q$.

Proof. Since $H \le 8n^2q^{1/2} + 1 < 16n^2q^{1/2} \le q$, we have $l \le k$. Using $\delta\colon S^2 \to \mathbb{F}_q$, and that $l \ge 1$, $h_0 \ge 2$, we find

$$H/2 \le \#S \le \#\delta(S^2) \le 2\#S \le 4H,$$

and for any $a = a_1\omega_1 + \cdots + a_l\omega_l \in \delta(S^2)$, with $-h < a_l < h$, the number $\#\delta^{-1}(\{a\})$ of $(u_1,u_2) \in S^2$ with $a = u_1 - u_2$ is equal to $p^{l-1}(h - |c_l|)$. Using Lemma 2.5, we can compute $M(S)$ in $O(\mathsf{M}(n)\#S\cdot\log q)$ or $O(n^2\mathsf{M}(n)q^{1/2}\log q)$ arithmetic operations in $\mathbb{F}_q$. From Lemmas 2.1 and 4.1, we get

$$\begin{aligned}\left|\sigma - M(S)/\#S^2\right| &\le |\sigma - \#\mathcal{C}/q| + \left|\#\mathcal{C}/q - M(S)/\#S^2\right|\\ &< n^2q^{-1/2} + 2n^2q^{1/2}/\#S \;\le\; 1/16 + 1/3 \;=\; 19/48 < 1/2. \qquad \square\end{aligned}$$

ALGORITHM 4.9. *Components.*
Input: *A curve* $\mathcal{C} \subseteq \mathbb{F}_q^2$ *without vertical lines, given by* $f \in \mathbb{F}_q[x,y]$ *of degree* n *satisfying Hypothesis A, and* $\delta > 0$.
Output: *An estimate of the number of absolutely irreducible components of* $\mathcal{C}$ *defined over* $\mathbb{F}_q$.

1. *Set* $H = \lceil 288\,\delta^{-2} n^4 \rceil$.
2. *Determine the set* $S \subseteq \mathbb{F}_q$ *as in Algorithm* 4.7.
3. *Choose* $a \in \mathbb{F}_q$ *at random.*
4. *Compute* $\#\mathcal{C}(S_a)$.
5. *Return the nearest integer to* $\#\mathcal{C}(S_a)/\#S^2$.

THEOREM 4.10. *If* $q > 36n^4$, *then Algorithm* 4.9 *computes the number of absolutely irreducible components of* $\mathcal{C}$ *correctly with probability at least* $1-\delta$. *It uses one random element and* $O(n^4 \mathsf{M}(n) \delta^{-2} \log q)$ *arithmetic operations in* $\mathbb{F}_q$.

Proof. The cost estimate follows from the fact that $\#\mathcal{C}(S_a)$ can be computed in $O(\mathsf{M}(n)\#S \cdot \log q)$ or $O(\delta^{-2} n^4 \mathsf{M}(n) \log q)$ arithmetic operations in $\mathbb{F}_q$. Corollary 4.3 implies that

$$\left|\#\mathcal{C}(S_a) - \#S \cdot \#\mathcal{C}/q\right| \le 2\delta^{-1} n^2 (\#S)^{1/2}$$

with probability at least $1-\delta$. From Lemma 2.1 we obtain

$$\begin{aligned} \left|\sigma - \#\mathcal{C}(S_a)/\#S\right| &\le |\sigma - \#\mathcal{C}/q| + \left|\#\mathcal{C}/q - \#\mathcal{C}(S_a)/\#S\right| \\ &\le\ n^2 q^{-1/2} + 2\delta^{-1} n^2 (\#S)^{-1/2} \le 1/6 + 1/6\ =\ 1/3 \end{aligned}$$

with probability at least $1-\delta$. □

ALGORITHM 4.11. *Parameters* λ_i.
Input: *A curve* $\mathcal{C} \subseteq \mathbb{F}_q^2$ *without vertical lines and given by* $f \in \mathbb{F}_p[x,y]$ *of degree* n, *and* $\delta > 0$.
Output: *The parameters* λ_i *for* $0 \le i \le n$.

1. *Set* $h = \lceil 12 n! n^{2n} p^{1/2} \rceil$.
2. *For* $0 \le i \le n$, *compute* $t_i(0,h)$.
3. *For* $0 \le i \le n$, *let* Λ_i *be the nearest integer to* $t_i(0,h) n!/h^2$, *and return* $\lambda_i = \Lambda_i / n!$.

THEOREM 4.12. *Let* $p > 576 (n!)^2 n^{4n}$ *be a prime. Then Algorithm* 4.11 *computes* λ_i *for* $0 \le i \le n$. *It uses* $O(n! n^{2n} \mathsf{M}(n) p^{1/2} \log p)$ *arithmetic operations in* $\mathbb{F}_p$.

Proof. Set $S = A(0,h)$, and let $0 \le i \le n$. It follows from Lemmas 2.3 and 4.4 that

$$\begin{aligned} \left|\lambda_i - t_i(0,h)/h^2\right| &\le |\lambda_i - r_i^*/p| + \left|r_i^*/p - t_i(0,h)/h^2\right| \\ &\le 2n^{2n} p^{-1/2} + 3n^{2n} p^{1/2} h^{-1} \\ &< 1/24n! + 1/4n!\ =\ 7/(24n!). \end{aligned}$$

Thus the algorithm works correctly. Since $12 n! n^{2n} p^{1/2} < p/2$, we have $h \le (p+1)/2$, and for $-h < a < h$ the number of $u_1, u_2 \in \mathbb{N}$ with $a = u_1 - u_2$ and $0 \le u_1, u_2 < h$ is equal to $h - |a|$. Using this fact and Lemma 2.5, we can compute $t_i(0,h)$ in $O(\mathsf{M}(n) h \log p)$ or $O(n! n^{2n} \mathsf{M}(n) p^{1/2} \log q)$ arithmetic operations in $\mathbb{F}_p$. □

ALGORITHM 4.13. *Parameters* λ_i.
Input: *A curve* $\mathcal{C} \subset \mathbb{F}_q^2$ *without vertical lines and given by* $f \in \mathbb{F}_q[x,y]$ *of degree* n, *and* $\delta > 0$.
Output: *An estimate of the parameters* λ_i *for* $0 \le i \le n$.

1. *Set* $h = \lceil 288(n!)^2 n^{4n} \delta^{-1} \rceil$.
2. *Choose* $a \in \mathbb{F}_p$ *at random.*
3. *For* $0 \le i \le n$, *compute* $r_i(a,h)$ *and the nearest integer* Λ_i *to* $n! r_i(a,h)/h$, *and return* $\lambda_i = \Lambda_i / n!$.

THEOREM 4.14. *Let* $p > 144(n!)^2 n^{4n}$ *be a prime,* $\mathcal{C} \subseteq \mathbb{F}_p^2$ *a plane curve of degree* n *without vertical lines, and* $\delta > 0$. *Algorithm* 4.13 *computes* λ_i *for* $0 \le i \le n$ *correctly with probability at least* $1 - \delta$. *It uses one random element and* $O((n!)^2 n^{4n} \mathsf{M}(n) \delta^{-1} \log p)$ *arithmetic operations in* $\mathbb{F}_p$.

Proof. The algorithm uses $O(\mathsf{M}(n) h \log p)$ or $O((n!)^2 n^{4n} \mathsf{M}(n) \delta^{-1} \log p)$ arithmetic operations in $\mathbb{F}_p$. Let $0 \le i \le n$. It follows from Corollary 4.6 that

$$|r_i(a,h) - h r_i^*/p| \le n^{2n} (8h\delta^{-1})^{1/2}$$

with probability at least $1 - \delta$. If this inequality holds, we find from Lemma 2.3 that

$$\begin{aligned} |\lambda_i - r_i(a,h)/h| &\le |\lambda_i - r_i^*/p| + |r_i^*/p - r_i(a,h)/h| \\ &\le 2n^{2n} p^{-1/2} + n^{2n} (8/h\delta)^{1/2} \\ &\le 1/6n! + 1/6n! = 1/3n!. \end{aligned}$$

Then $\lambda_i = \Lambda_i / n!$ is the correct answer. □

5. Distribution of points in multiplicative strips. In the previous sections we did not succeed in computing the projection distribution parameters r_i in an arbitrary finite field, as we have to know the behavior of x along absolutely irreducible components of the fiber product curves $\mathcal{C}_k$. Instead of additive strips, we consider in this section multiplicative strips that may help us in some cases.

Our main tool is the bound from [15] on multiplicative character sums along an algebraic curve, rather than Bombieri's bound that we used before.

For $\lambda \in \mathbb{F}_q^\times$ and integers a and h, we denote by $M(\lambda, a, h)$ the multiplicative interval

$$M(\lambda, a, h) = \{\lambda^{a+t} : 1 \le t \le h\} \subseteq \mathbb{F}_q^\times,$$

and given a curve $\mathcal{C} \subseteq \mathbb{F}_q^{m+1}$, we let

$$\mathcal{C}(\lambda, a, h) = \mathcal{C}(M(\lambda, a, h)).$$

We prove some analogues of Lemma 2.1 of [7] and Lemma 4.2 of this paper. The following condition on a curve $\mathcal{C} \subseteq \mathbb{F}_q^{m+1}$ is used in Perel'muter's theorem.

HYPOTHESIS B. *The first coordinate function* x *is not a power* g^e *of a rational function* g *on any absolutely irreducible component of* $\mathcal{C}$, *where* g *is defined over an algebraic closure of* $\mathbb{F}_q$, *and* $e \ge 2$ *is an integer.*

THEOREM 5.1. *Let* $\mathcal{C} \subseteq \mathbb{F}_q^m$ *be a curve of degree* n *without vertical components and satisfying Hypothesis B,* $\lambda \in \mathbb{F}_q^\times$ *be of order* τ, *and* a *and* $h \le \tau$ *be integers. Then*

$$\left| \#\mathcal{C}(\lambda, a, h) - h \#\mathcal{C}/q \right| \le 2n^2 q^{1/2} \log q.$$

Proof. Let $\theta \in \mathbb{F}_q$ be a primitive element such that $\lambda = \theta^{(q-1)/\tau}$.

Denote by $\operatorname{ind} u$ the index of $u \in \mathbb{F}_q^\times$ in base θ, i.e., the smallest nonnegative integer t with $u = \theta^t$, so that

$$\operatorname{ind}(\lambda^{a+t}) \equiv (q-1)(a+t)\tau^{-1} \bmod (q-1).$$

Then

$$\#\mathcal{C}(\lambda,a,h)=\frac{1}{q-1}\sum_{(u,v)\in\mathcal{C}}\sum_{1\le t\le h}\sum_{0\le s\le q-2}\exp\biggl(\frac{2\pi is(\operatorname{ind}\,u-(q-1)(a+t)\tau^{-1})}{q-1}\biggr)$$
$$=\frac{1}{q-1}\sum_{0\le s\le q-2}\sum_{(u,v)\in\mathcal{C}}\chi_s(u)\sum_{1\le t\le h}\exp(-2\pi is(a+t)/\tau),$$

where $u\in\mathbb{F}_q$ and $v\in\mathbb{F}_q^m$ in the sums. For $0\le s\le q-2$, define a multiplicative character χ_s on $\mathbb{F}_q$ by

$$\chi_s(u)=\exp[2\pi is\operatorname{ind}\,u/(q-1)],$$

for $u\in\mathbb{F}_q^\times$, and set $\chi_s(0)=0$. Separating the term corresponding to $s=0$ we get

$$\begin{aligned}\#\mathcal{C}(\lambda,a,h)\ &=\ \frac{h}{q-1}(\#\mathcal{C}-E)\\ &+\frac{1}{q-1}\sum_{1\le s\le q-2}\sum_{(u,v)\in\mathcal{C}}\chi_s(u)\sum_{1\le t\le h}\exp(-2\pi is(a+t)/\tau),\end{aligned}\tag{5.1}$$

where

$$E=\sum_{(0,v)\in\mathcal{C}}1=\#(\mathcal{C}\cap\{x=0\})\le n,$$

by Bézout's theorem. Theorem 2 of [15] implies that for any s

$$\biggl|\sum_{(u,v)\in\mathcal{C}}\chi_s(x)\biggr|\le(n^2-3n)q^{1/2}+n^2.$$

Since $h\le q-1$, we have

$$\begin{aligned}&\bigl|\#\mathcal{C}(\lambda,a,h)-h\#\mathcal{C}/(q-1)\bigr|\\ &\le\frac{hn}{q-1}+\frac{(n^2-3n)q^{1/2}+n^2}{q-1}\sum_{1\le s\le q-2}\biggl|\sum_{1\le t\le h}\exp(2\pi is(a+t)/\tau)\biggr|\\ &\le n+\frac{(n^2-3n)q^{1/2}+n^2}{\tau}\sum_{1\le s<\tau}\biggl|\sum_{1\le t\le h}\exp(-2\pi ist/\tau)\biggr|.\end{aligned}$$

Using the well-known inequality

$$\sum_{1\le s<\tau}\biggl|\sum_{1\le t\le h}\exp(2\pi ist/\tau)\biggr|\le\tau\log\tau,$$

we get

$$\bigl|\#C(\lambda,a,h)-h\#\mathcal{C}/(q-1)\bigr|\le n+\bigl((n^2-3n)q^{1/2}+n^2\bigr)\log\tau.$$

Taking into account that $\#\mathcal{C}\le nq$ and thus

$$\biggl|\frac{h\#C}{q-1}-\frac{h\#C}{q}\biggr|=\frac{h\#C}{q(q-1)}\le n,\tag{5.2}$$

we obtain finally that

$$\begin{aligned}\bigl|\#\mathcal{C}(\lambda,a,h) - h\#\mathcal{C}/q\bigr| &\le n + n + \bigl((n^2-3n)q^{1/2}+n^2\bigr)\log\tau\\ &\le 2n + \bigl((n^2-3n)q^{1/2}+n^2\bigr)\log\tau\\ &\le 2n^2q^{1/2}\log q. \qquad \square\end{aligned}$$

THEOREM 5.2. *Let $\mathcal{C}\subseteq\mathbb{F}_q^m$ be a curve without vertical components and of degree $n\ge 2$ satisfying Hypothesis B, $\lambda\in\mathbb{F}_q^\times$ be of order τ, and $h\le\tau$. Then*

$$\sum_{0\le a\le q-2}\bigl(\#\mathcal{C}(\lambda,a,h) - h\#\mathcal{C}/q\bigr)^2 \le 8n^4qh.$$

Proof. Using the notation of the previous proof, we have

$$(q-1)\left|\frac{\#\mathcal{C}}{q} - \frac{\#\mathcal{C}-E}{q-1}\right| = \left|\frac{-\#\mathcal{C}}{q}+E\right| \le n,$$

$$\begin{aligned}&\sum_{0\le a\le q-2}\bigl(\#\mathcal{C}(\lambda,a,h) - h\#\mathcal{C}/q\bigr)^2\\ &\le 2\sum_{0\le a\le q-2}\left[\#\mathcal{C}(\lambda,a,h) - \frac{h(\#\mathcal{C}-E)}{q-1}\right]^2 + 2\sum_{0\le a\le q-2}\left[\frac{h\#\mathcal{C}}{q} - \frac{h(\#\mathcal{C}-E)}{q-1}\right]^2\\ &\le 2W + 2n^2h^2/(q-1),\end{aligned}$$

where

$$\begin{aligned}W &= \sum_{0\le a\le q-2}\left[\#\mathcal{C}(\lambda,a,h) - \frac{h(\#\mathcal{C}-E)}{q-1}\right]^2\\ &= (q-1)^{-2}\sum_{0\le a\le q-2}\left|\sum_{1\le s\le q-2}\sum_{(u,v)\in\mathcal{C}}\chi_s(u)\sum_{1\le t\le h}\exp(2\pi is(a+t)/\tau)\right|^2,\end{aligned}$$

by (5.1). Using $|\alpha|^2=\alpha\bar\alpha$ for $\alpha\in\mathbb{C}$, we have

$$\begin{aligned}W = (q-1)^{-2}&\sum_{1\le s_1,s_2\le q-2}\sum_{(u_1,v_1),(u_2,v_2)\in\mathcal{C}}\chi_{s_1}(u_1)\chi_{s_2}(u_2^{-1})\\ &\cdot\sum_{1\le t_1,t_2\le h}\exp(2\pi i(s_1t_1-s_2t_2)/\tau)\sum_{0\le a\le q-2}\exp(2\pi ia(s_1-s_2)/\tau).\end{aligned}$$

Noting that the inner sum equals 0 when $s_1\ne s_2$ and $q-1$ otherwise, we get

$$\begin{aligned}W &= (q-1)^{-1}\sum_{1\le s\le q-2}\sum_{(u_1,v_1),(u_2,v_2)\in\mathcal{C}}\chi_s(u_1u_2^{-1})\sum_{1\le t_1,t_2\le h}\exp(2\pi is(t_1-t_2)/\tau)\\ &= (q-1)^{-1}\sum_{1\le s\le q-2}\left|\sum_{(u,v)\in\mathcal{C}}\chi_s(u)\right|^2\left|\sum_{1\le t\le h}\exp(2\pi ist/\tau)\right|^2.\end{aligned}$$

Theorem 2 of [15] yields

$$W \le \frac{\bigl((n^2-3n)q^{1/2}+n^2\bigr)^2}{q-1}\sum_{1\le s\le q-2}\left|\sum_{1\le t\le h}\exp(2\pi ist/\tau)\right|^2.$$

Taking into account the equality

$$\sum_{0\le s\le q-2}\Big|\sum_{1\le t\le h}\exp(2\pi ist/\tau)\Big|^2=\sum_{1\le t_1,t_2\le h}\ \sum_{0\le s\le q-2}\exp(2\pi is(t_1-t_2)/\tau)=h(q-1),$$

we obtain

$$W\le\frac{\big((n^2-3n)q^{1/2}+n^2\big)^2}{q-1}\cdot\big(h(q-1)-h^2\big),$$

$$\sum_{0\le a\le q-2}\big(\#\mathcal{C}(\lambda,a,h)-h\#\mathcal{C}/q\big)^2\le 2n^2h^2/(q-1)+2\big((n^2-3n)q^{1/2}+n^2\big)^2h$$

$$\le 2n^2h\big(1+((n-3)q^{1/2}+n)^2\big)\le 2n^2h(nq^{1/2}+n)^2\ \le\ 8n^4hq. \qquad \square$$

We now show that Hypothesis B is not a severe restriction, in that it is satisfied after a generic linear transformation. This is most naturally shown for a projective curve over an algebraically closed field $\mathbb{K}$.

So let $\mathcal{X}\subseteq\mathbb{P}_{\mathbb{K}}^{m+1}$ be a reduced curve of degree n, possibly reducible or singular, $\mathbb{H}\cong\mathbb{P}_{\mathbb{K}}^{m+1}$ the space of hyperplanes in $\mathbb{P}_{\mathbb{K}}^{m+1}$, and for $\mathcal{H}\in\mathbb{H}$, let $l_{\mathcal{H}}$ be the rational linear function whose zero set is $\mathcal{H}$. We say that $\mathcal{H}\in\mathbb{H}$ *intersects* $\mathcal{X}$ *transversally* if and only if $\#(\mathcal{X}\cap\mathcal{H})=n$. The following facts are well known.

FACT 5.3. *Let $\mathcal{X}$ be as above, and $\mathcal{H}\in\mathbb{H}$.*

(i) *If no component of $\mathcal{X}$ is contained in $\mathcal{H}$, then $\#(\mathcal{X}\cap\mathcal{H})\le n$.*
(ii) *If $\mathcal{H}$ does not contain a tangent line to $\mathcal{X}$ or a singular point of $\mathcal{X}$, then $\mathcal{H}$ intersects $\mathcal{X}$ transversally.*
(iii) *There is a proper closed subvariety $E\subseteq\mathbb{H}$ of degree at most $n(n-1)$ such that $\mathcal{H}$ intersects $\mathcal{X}$ transversally if $\mathcal{H}\in\mathbb{H}\setminus E$.*
(iv) *If $\mathcal{H}$ intersects $\mathcal{X}$ transversally, then $l_{\mathcal{H}}$ is not a power g^e of a rational function g on any absolutely irreducible component of $\mathcal{X}$, with $e\ge 2$.*

For a plane curve, (iii) follows from, e.g., Proposition 5.2.2 of [3]. For a curve $\mathcal{C}\subseteq\mathbb{F}_q^{m+1}$, Fact 5.3 implies that almost all linear transformations of $\mathcal{C}$ satisfy Hypothesis B. We only make this explicit for $m=1$. We need the fact that there exists a line (over $\mathbb{K}$) through the origin which is not a tangent to $\mathcal{C}$; this is true for all curves except the "strange" conic in characteristic two (see [10, Theorem IV.3.9]).

PROPOSITION 5.4. *Let $f\in\mathbb{F}_q[x,y]$ be squarefree of degree n, with either $n\ne 2$ or $\operatorname{char}\mathbb{F}_q\ne 2$, and for $\alpha\in\mathbb{F}_q$, let*

$$\mathcal{C}_\alpha=\{f(x,y+\alpha x)=0\}=\{(a,b)\in\mathbb{F}_q^2\colon f(a,b+\alpha a)=0\}.$$

Then there exists $E\subseteq\mathbb{F}_q$ with $\#E\le n(n-1)$ and such that $\mathcal{C}_\alpha$ satisfies Hypothesis B for all $\alpha\in\mathbb{F}_q\setminus E$.

In order to design algorithms from the above results, we have to construct wide enough multiplicative strips or, equivalently, to find elements $\lambda\in\mathbb{F}_q^*$ of sufficiently large order, as in [9]; certainly a primitive root is sufficient. Results about the construction, distribution, and density of primitive roots can be found in [14]; see [22] for a survey and also [6], [16], [20], [21] for the currently best results in this area.

Acknowledgments. Part of the first author's work was done on a visit to Macquarie University and during a sabbatical visit to the Institute for Scientific Computation at ETH Zürich, whose hospitality is gratefully acknowledged. The research was also supported by the Information Technology Research Centre and the Natural Sciences and Engineering Research Council of Canada. Part of the second author's work was done during a sabbatical visit to Universität Paderborn, which was supported by Deutsche Forschungsgemeinschaft. We thank Gerhard Frey and Henning Stichtenoth for pointing out (4.2).

REFERENCES

[1] E. Bombieri, *On exponential sums in finite fields*, Amer. J. Math., 88 (1966), pp. 71–105.
[2] E. Bombieri and H. Davenport, *On two problems of Mordell*, Amer. J. Math., 88 (1966), pp. 61–70.
[3] E. Brieskorn and H. Knörrer, *Plane Algebraic Curves*, Birkhäuser-Verlag, Basel, Switzerland, 1986.
[4] S. D. Cohen, *The distribution of polynomials over finite fields*, Acta Arith., 17 (1970), pp. 255–271.
[5] J. von zur Gathen, *Values of polynomials over finite fields*, Bull. Austral. Math. Soc., 43 (1991), pp. 141–146.
[6] J. von zur Gathen and M. Giesbrecht, *Constructing normal bases in finite fields*, J. Symbolic Comput., 10 (1990), pp. 547–570.
[7] J. von zur Gathen, M. Karpinski, and I. E. Shparlinski, *Counting curves and their projections*, Comput. Complexity, 6 (1996), pp. 64–99.
[8] J. von zur Gathen and I. E. Shparlinski, *Finding points on curves over finite fields*, in Proc. 36th Ann. IEEE Symp. on Foundations of Computer Science, 1995, pp. 284–292.
[9] J. von zur Gathen and I. E. Shparlinski, *Orders of Gauss periods in finite fields*, Appl. Algebra Engrg. Comm. Comput., 9 (1998), pp. 15–24.
[10] R. Hartshorne, *Algebraic Geometry*, Springer-Verlag, New York, 1977.
[11] M.-D. Huang and D. Ierardi, *Counting rational points on curves over finite fields*, in Proc. 34th Ann. IEEE Symp. on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 616–625.
[12] M.-D. Huang and Y.-C. Wong, *Solving systems of polynomial congruences modulo a large prime*, in Proc. of 1996 IEEE Symp. on Foundations of Computer Science, 1996, pp. 115–124.
[13] R. M. Karp, M. Luby, and N. Madras, *Monte-Carlo approximation algorithms for enumeration problems*, J. Algorithms, 10 (1989), pp. 429–448.
[14] R. Lidl and H. Niederreiter, *Finite fields*, in Encyclopedia of Mathematics and Its Applications 20, Addison-Wesley, Reading, MA, 1983.
[15] G. I. Perel'muter, Оценка суммы вдол алгебраическои кривои (Bounds on sums along algebraic curves), Mat. Zametki, 5 (1969), pp. 373–380.
[16] G. I. Perel'muter and I. E. Shparlinski, О распределении нервообразных корнеи в конечных полях (On the distribution of primitive roots in finite fields), Uspekhi Mat. Nauk, 45 (1990), pp. 185–186.
[17] R. Pila, *Frobenius maps of Abelian varieties and finding roots of unity in finite fields*, Math. Comput., 55 (1990), pp. 745–763.
[18] A. Schönhage and V. Strassen, *Schnelle Multiplikation großer Zahlen*, Computing, 7 (1971), pp. 281–292.
[19] R. J. Schoof, *Elliptic curves over finite fields and the computation of square roots mod p*, Math. Comput., 44 (1985), pp. 483–494.
[20] V. Shoup, *Searching for primitive roots in finite fields*, Math. Comp., 58 (1992), pp. 369–380.
[21] I. E. Shparlinski, *On primitive elements in finite fields and on elliptic curves*, Math. USSR Sbornik, 71 (1992), pp. 41–50.
[22] I. E. Shparlinski, *Computational and Algorithmic Problems in Finite Fields*, Math. Appl. 88., Kluwer Academic Publishers, Norwell, MA, 1992.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 841–846

BIPARTITE EDGE COLORING IN $O(\Delta m)$ TIME*

ALEXANDER SCHRIJVER†

Abstract. We show that a minimum edge coloring of a bipartite graph can be found in $O(\Delta m)$ time, where Δ and m denote the maximum degree and the number of edges of G, respectively. It is equivalent to finding a perfect matching in a k-regular bipartite graph in $O(km)$ time.

By sharpening the methods, a minimum edge coloring of a bipartite graph can be found in $O((p_{\max}(\Delta) + \log \Delta)m)$ time, where $p_{\max}(\Delta)$ is the largest prime factor of Δ. Moreover, a perfect matching in a k-regular bipartite graph can be found in $O(p_{\max}(k)m)$ time.

Key words. bipartite, edge-coloring, complexity, timetabling, perfect, matching

AMS subject classifications. Primary: 68R10, 90C27; Secondary: 05C70, 05C85

PII. S0097539796299266

1. Introduction.

In a classical paper, König [9] showed that the edges of a bipartite graph G can be colored with $\Delta(G)$ colors, where $\Delta(G)$ is the maximum degree of G. (In this paper, "coloring" edges presumes that edges that have a vertex in common obtain different colors.)

König's proof is essentially algorithmic, yielding an $O(nm)$ time algorithm (n and m denote the numbers of vertices and edges, respectively, of the graph). As was shown by Gabow [4], the $O(\sqrt{n}m)$ bipartite matching algorithm of Hopcroft and Karp [8] implies an $O(\sqrt{n}m \log \Delta)$ bipartite edge-coloring algorithm. This was improved by Cole and Hopcroft [1] to $O(m \log m)$ by extending methods of Gabow and Kariv [5], [6].

Fixing the maximum degree Δ, the methods found as yet are superlinear (albeit slightly). In this paper we give a linear-time method for fixed or bounded Δ. More precisely, we give an $O(\Delta m)$ method for bipartite edge coloring. It implies (in fact, is equivalent to) finding a perfect matching in a k-regular bipartite graph in $O(km)$ time.

Ultimately one would hope for an $O(m \log k)$ (or even $O(m)$) algorithm finding a perfect matching in a k-regular bipartite graph and for an $O(m \log \Delta)$ algorithm for bipartite edge coloring (the first algorithm implies the second, by a method of Gabow [4] — see below). We did not find such algorithms, although our methods can be extended to obtain some supporting results.

In particular, define, for any natural number k,

$$\phi(k) := \sum_{i=1}^{t} \frac{p_i}{\prod_{j=1}^{i-1} p_j}, \tag{1}$$

where $p_1 \leq \cdots \leq p_t$ are primes with $k = p_1 \cdot \ldots \cdot p_t$. We give an $O((\phi(\Delta) + \log \Delta)m)$ bipartite edge-coloring algorithm. Note that in $\phi(\Delta) + \log \Delta$, the term $\phi(\Delta)$ dominates if Δ is prime, while $\log \Delta$ dominates if Δ is a power of 2. Note also that $\phi(\Delta) \leq 2p_{\max}(\Delta)$, where $p_{\max}(\Delta)$ denotes the largest prime factor in Δ. So

*Received by the editors February 26, 1996; accepted for publication (in revised form) February 16, 1997; published electronically September 22, 1998.
http://www.siam.org/journals/sicomp/28-3/29926.html

†CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands, and Department of Mathematics, University of Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands (lex@cwi.nl).

fixing the maximum prime factor of Δ, there is an $O(m \log \Delta)$ bipartite edge-coloring algorithm.

Moreover, we give an $O(\phi(k)m)$ algorithm finding a perfect matching in a k-regular bipartite graph. So bounding the maximum prime factor of k, there is a linear-time perfect matching algorithm for k-regular bipartite graphs.

The proof idea is an extension of the following idea of Gabow [4] to find a perfect matching in a 2^t-regular bipartite graph G in linear time. First find an Eulerian orientation of G (taking $O(m)$ time), and consider those edges that are oriented from vertex-color class I to vertex-color class II (in the 2-vertex coloring of G). This gives a 2^{t-1}-regular subgraph of G. Repeating this, we end up with a 1-regular subgraph of G, being a perfect matching in G. The time is $O(m + \frac{1}{2}m + \frac{1}{4}m + \cdots) = O(m)$.

One can similarly find a 2^t-edge coloring in $O(tm)$ time. In extending this method to prime factors other than 2 we use some techniques of [10] for estimating the number of perfect matchings and edge colorings of bipartite graphs.

In this paper, all graphs may have multiple edges.

2. Some practical motivation. As is well known, bipartite edge coloring can be applied in timetabling. A pure instance of timetabling consists of a set of teachers, a set of classes, and a list L of pairs (t, c) of a teacher t and a class c, indicating that teacher t has to teach class c during a time slot (say, an hour) within the time span of the schedule (say, a week). A pair (t, c) may occur several times in the list, indicating the number of hours the pair t, c should meet weekly.

A timetable then is an assignment of the pairs in the list to hours, from a set H of possible hours, in such a way that no teacher t and no class c occurs in two pairs that are assigned to the same hour. This clearly is a bipartite edge-coloring problem, and by König's theorem, there is a timetable if and only if $|H|$ is not smaller than the number of times that any teacher t or any class c occurs in L. So by the result of Cole and Hopcroft [1] a timetable can be found in $O(|L| \log |L|)$ time, and by our theorem, it can be found also in $O(|H| \cdot |L|)$ time. (In practice, several additional constraints are put on a timetable, making the problem usually NP-complete—cf. Even, Itai, and Shamir [3].)

In many countries, schools are merging, yielding an increase in size, including in numbers of teachers and of classes. So the list L grows. However, the number of hours during a week does not grow. This gives that, in this interpretation, the algorithm is linear in the size of the school.

Moreover, often H is built up from smaller units (say, days), implying that $|H|$ does not have large prime factors. ($|H|$ typically has prime factors 2, 3, and 5 only, and sometimes 7.) This gives that applying the $O(\phi(|H|+\log |H|)|L|)$-time algorithm can be fruitful. Similarly, the method is not very sensitive to doubling or tripling the time span (say to two or three weeks).

3. An $O(\Delta m)$ bipartite edge-coloring algorithm. Basic in the edge-coloring algorithm (as in [4]) is a subroutine finding a matching that covers all maximum-degree vertices, and that hence can serve as our first color. To this end we show Theorem 1.

THEOREM 1. *A perfect matching in a k-regular bipartite graph can be found in $O(km)$ time.*

Proof. Let $G = (V, E)$ be a k-regular bipartite graph. For any function $w : E \longrightarrow \mathcal{Z}_+$, let E_w be the set of edges with $w(e) > 0$. For any $F \subseteq E$, denote $w(F) := \sum_{e \in F} w(e)$ and let χ^F be the incidence vector of F.

Initially set $w(e) := 1$ for each edge e. Next apply the following iteratively:

(2) Find a circuit C in E_w. Let $C = M \cup N$ for matchings M and N with $w(M) \geq w(N)$. Reset $w := w + \chi^M - \chi^N$.

Note that, at any iteration, the equality $w(\delta(v)) = k$ is maintained for all $v \in V$ (where $\delta(v)$ is the set of edges incident with v).

To see that the process terminates, first note that at any iteration the sum

$$\sum_{e \in E} w(e)^2 \tag{3}$$

increases by

$$\sum_{e \in M} ((w(e)+1)^2 - w(e)^2) + \sum_{e \in N} ((w(e)-1)^2 - w(e)^2) = 2w(M) + |M| - 2w(N) + |N|, \tag{4}$$

which is at least $|C|$ (as $w(M) \geq w(N)$). Moreover, (3) is bounded, since $w(e) \leq k$ for each edge e. So the process terminates.

At termination, we have that the set E_w is a forest and hence is a perfect matching (since $w(e) = k$ for each end edge e of E_w). This implies that at termination the sum (3) is equal to $\frac{1}{2}nk^2 = km$.

Now by depth first search we can find a circuit C in (2) in $O(|C|)$ time on average. Indeed, keep a path P of edges e with $0 < w(e) < k$. Let v be the last vertex of P. Choose an edge $e = vu$ incident with v but not in P. If u does not occur in P, we reset $P := P \cup \{e\}$ and iterate. If u does occur in P, let C be the circuit in $P \cup \{e\}$, and apply (2) to C. Next reset $P := P \setminus C$, and iterate.

If $P = \emptyset$, choose any edge e with $0 < w(e) < k$, and set $P := \{e\}$. If no such edge e exists, we are done. □

For k smaller than $\sqrt{\log n}$, the $O(km)$ bound is asymptotically better than the $O(m + n \log n (\log k)^2)$ bound proved by Cole and Hopcroft [1]. (An algorithm related to, but different from, the algorithm described in Theorem 1 was proposed by Csima and Lovász [2], giving an $O(n^2 k \log k)$ time bound.)

By applying a technique of Gabow [4], one can derive from Theorem 1 the following stronger statement:

COROLLARY 1a. *A k-edge coloring of a k-regular bipartite graph can be found in $O(km)$ time.*

Proof. If k is odd, first find a perfect matching M, remove M from G, and apply recursion (M will serve as color).

If k is even, find an Eulerian orientation of G. Let $k' = \frac{1}{2}k$. Then split G into two k'-regular graphs $G_1 = (V, E_1)$ (with E_1 the set of edges oriented from vertex-color class I to vertex-color class II) and $G_2 = (V, E_2)$ (with $E_2 := E \setminus E_1$). Find recursively k'-edge colorings of G_1 and G_2. The union of the two colorings is a k-edge coloring of G.

The time is bounded as follows. Starting with G, we can find M (if k is odd), find the Eulerian orientation, and split G into G_1 and G_2, in time ckm for some constant c. Then the whole recursion takes time $2ckm$. This can be shown inductively, as $2ckm = ckm + 2ck'm' + 2ck'm'$, where $m' = |E(G_1)| = |E(G_2)| = \frac{1}{2}m$. □

This implies the sharper statement as shown in Corollary 1b.

COROLLARY 1b. *A $\Delta(G)$-edge-coloring of a bipartite graph $G = (V, E)$ can be found in $O(\Delta(G)m)$ time.*

Proof. Let $k := \Delta(G)$. First iteratively merge any two vertices in the same color class of G if each has degree at most $\frac{1}{2}k$. The final graph H will have at most two vertices of degree at most $\frac{1}{2}k$, and moreover, $\Delta(H) = k$ and any k-edge coloring of H yields a k-edge coloring of G. Next make a copy H' of H, and join each vertex v of H by $k - d_H(v)$ parallel edges with its copy v' in H' (where $d_H(v)$ is the degree of v in H). This gives the k-regular graph G', with $|E(G')| = O(|E(G)|)$. By Corollary 1a we can find a k-edge coloring of G' in $O(k|E(G')|)$ time. This gives a k-edge coloring of H and hence a k-edge coloring of G. □

4. Toward an $O(m \log \Delta)$ method. The results of section 3 can be sharpened by using divisibility properties of $\Delta(G)$. First we sharpen Corollary 1a. We repeat the definition of $\phi(k)$ for any natural number k:

$$\phi(k) := \sum_{i=1}^{t} \frac{p_i}{\prod_{j=1}^{i-1} p_j}, \tag{5}$$

where $p_1 \leq \cdots \leq p_t$ are primes with $k = p_1 \cdot \ldots \cdot p_t$.

THEOREM 2. *A k-edge coloring of a k-regular bipartite graph $G = (V, E)$ can be found in $O((\phi(k) + \log k)m)$ time.*

Proof. Let $k = pk'$ with p prime. Split each vertex v into k' new vertices $v_1, \ldots, v_{k'}$, and distribute the edges incident with v over $v_1, \ldots, v_{k'}$ in such a way that each vertex v_i is incident with exactly p edges. This gives the p-regular graph $\tilde{G}$. Find a p-edge-coloring of $\tilde{G}$. The colors give a partition of E into classes $E_1, \ldots, E_p$ in such a way that each graph $G_j = (V, E_j)$ is k'-regular. Next find a k'-edge coloring of G_p, yielding perfect matchings $M_1, \ldots, M_{k'}$.

Now we apply the following iteratively. We have a partition of E into perfect matchings $M_1, \ldots, M_{\alpha k'}$ and k'-regular graphs $E_1, \ldots, E_{p-\alpha}$. (Initially, $\alpha = 1$.) Let $q := \min\{\alpha, p - \alpha\}$. Choose r such that $qk' + r$ is a power of 2 and such that $r \leq qk'$. Let $E' := M_1 \cup \cdots \cup M_r \cup E_1 \cup \cdots \cup E_q$. Then $G' := (V, E')$ is a $qk' + r$-regular graph. Next $qk' + r$-edge-color G', yielding colors $N_1, \ldots, N_{qk'+r}$. Now replace $M_1, \ldots, M_r$ by $N_1, \ldots, N_{qk'+r}$ and $E_1, \ldots, E_{p-\alpha}$ by $E_{q+1}, \ldots, E_{p-\alpha}$ and iterate. We stop if $\alpha = p$.

So at any iteration, α is replaced by $\alpha + q$. Moreover, at any iteration except possibly the last iteration, we have $q = \alpha$. So at any iteration except possibly the last one, q is twice as large as at the previous iteration.

By [4], the work in the iteration takes time $O(|E'| \log(qk' + r)) = O(|E'| \log k)$, since $qk' + r$ is a power of 2 and since $qk' + r \leq k$. Since $|E'| = \frac{1}{2}(qk' + r)n \leq qk'n$, over all iterations the work is $O((1 + 2 + 2^2 + \cdots + 2^{\log p})k'n \log k) = O(pk'n \log k) = O(m \log k)$.

To this time bound we must add the time needed to obtain $G_1, \ldots, G_p$, which takes $O(pm)$ time by Corollary 1b, since it amounts to p-edge coloring the p-regular graph $\tilde{G}$, having m edges, and the time needed to edge color G_p, which takes (by induction) $O((\phi(k') + \log k')m')$ time, where $m' = m/p$ is the number of edges of G_p. Since $\phi(k) = p + \phi(k')/p$, we have the required time bound. □

This gives Corollary 2a.

COROLLARY 2a. *A $\Delta(G)$-edge coloring of a bipartite graph G can be found in $O((\phi(\Delta(G)) + \log \Delta(G))m)$ time.*

Proof. It is proved directly from Theorem 2 by the method of Corollary 1b. □

Note that

$$\phi(k) \leq 2p_{\max}(k) \tag{6}$$

(where $p_{\max}(k)$ is the largest prime factor of k). This follows inductively, since if $k = pk'$, with p the smallest prime factor of k, then $\phi(k) = p + \phi(k')/p \leq p_{\max}(k) + (2p_{\max}(k')/p) \leq 2p_{\max}(k)$. This implies Corollary 2b.

COROLLARY 2b. *A $\Delta(G)$-edge coloring of a bipartite graph G can be found in $O((p_{\max}(\Delta(G)) + \log \Delta(G))m)$ time.*

Proof. The proof follows directly from Corollary 2a with (6). □

Note that in performing this method one does not need to apply deep number-theoretic algorithms to find the prime factorization of k. Indeed, the factors $p_1, \ldots, p_t$ can be found in $O(\phi(k)k)$ time, since the smallest prime factor p can be found in time $O(pk)$ by trying $i = 2, 3, \ldots$ as divisor of k (for each i taking $O(k)$ time), until we reach p. Next we can apply recursion to $k' := k/p$, taking recursively $O(\phi(k')k')$ time. This gives $O(\phi(k)k)$ time overall, since $\phi(k) = p + \phi(k')/p$.

A sharpening can be obtained also for finding perfect matchings in k-regular bipartite graphs.

THEOREM 3. *A perfect matching in a k-regular bipartite graph G can be found in $O(\phi(k)m)$ time.*

Proof. Write $k = pk'$ with p the smallest prime factor of k. Make the graph $\tilde{G}$ as in the proof of Theorem 2. So $\tilde{G}$ is p-regular. Find a perfect matching M in $\tilde{G}$. It gives a k'-regular subgraph $G' = (V, E')$ of G. In G' we find recursively a perfect matching.

Finding perfect matching M in $\tilde{G}$ takes time $O(pm)$ by Theorem 1. Finding matching N in G' takes time $O(\phi(k')m/p)$ by induction (as G' is k'-regular and has m/p edges). Since $\phi(k) = p + \phi(k')/p$, the whole process takes $O(\phi(k)m)$ time. □

COROLLARY 3a. *A matching covering all maximum-degree vertices in a bipartite graph can be found in $O(\phi(\Delta)m)$ time.*

Proof. The proof follows directly from Theorem 3, using the technique of Corollary 1b. □

By (6), Theorem 3 can be stated in a weaker form as Corollary 3b.

COROLLARY 3b. *A perfect matching in a k-regular bipartite graph can be found in $O(p_{\max}(k)m)$ time.*

Proof. The proof follows directly from Theorem 3, using (6). □

5. Some open questions. It would be surprising if divisibility properties of the maximum degree $\Delta(G)$ of a bipartite graph G would determine the complexity of edge coloring G. However, our results are blocked by the primes. If $\Delta(G)$ is a prime, we do not have anything better than an $O(\Delta(G)m)$-time algorithm. So the main problem is to "break" a prime. More precisely,

(7) Is there an $O(m \log k)$ algorithm for finding a perfect matching in a k-regular bipartite graph?

The method of Cole and Hopcroft [1] gives an $O(m + n \log n \log^2 k)$ algorithm to find a perfect matching in any k-regular bipartite graph. If there would be an $O(m \log k)$ perfect matching algorithm for k-regular bipartite graphs, there exists an $O(m \log \Delta)$ bipartite edge-coloring algorithm (by methods like in Theorem 2 above), thus answering our second question:

(8) Is there an $O(m \log \Delta)$ algorithm for bipartite edge coloring?

Similar methods as used for proving Theorem 2 give an approximative method, namely, a bipartite $(\Delta + \lfloor \log(\Delta - 1) \rfloor)$-edge-coloring algorithm, with time bound $O(m \log \Delta)$. Indeed, let $G = (V, E)$ be a bipartite graph of maximum degree Δ. In $O(m)$ time we can split E into E' and E'' such that both $G' = (V, E')$ and $G'' = (V, E'')$ have maximum degree at most $\Delta' := \lceil \frac{1}{2}\Delta \rceil$. We may assume that $|E'| \leq \frac{1}{2}m$. Let $t := \Delta' + \lfloor \log(\Delta' - 1) \rfloor$. Then t-edge color G' recursively, giving colors $M_1, \ldots, M_t$. Choose $s \leq t$ such that $\Delta' + s$ is a power of 2. Next $(\Delta' + s)$-edge color the graph H made by $M_1 \cup \cdots \cup M_s \cup E''$. With the remaining $M_{s+1}, \ldots, M_t$ it gives an edge coloring of G with

$$(\Delta' + s) + (t - s) = 2\Delta' + \lfloor \log(\Delta' - 1) \rfloor \leq \Delta + \lfloor \log(\Delta - 1) \rfloor \tag{9}$$

colors. Since the number of edges in G' is at most $\frac{1}{2}m$ and since edge coloring H takes $O(m \log(\Delta' + s)) = O(m \log \Delta)$ time, this gives an $O(m \log \Delta)$ time bound.

The nonbipartite case is NP-complete, by the well-known result of Holyer [7]: it is NP-complete to decide if a 3-regular graph can be 3-edge colored. However, it is not difficult to see that a 3-regular graph can be 4-edge colored in *linear* time. Actually, any graph of maximum degree 3 can be 4-edge colored in $O(m)$ time.

By Vizing's theorem, each simple graph G can be $(\Delta(G) + 1)$-edge colored. (If $\Delta(G) \leq 3$ we can delete the condition that G be simple.) This prompts the question:

(10) Is there an $O(\Delta m)$-time $(\Delta+1)$-edge coloring algorithm for simple graphs?

Of course, the stronger question is to ask for an $O(m \log \Delta)$ algorithm.

REFERENCES

[1] R. Cole and J. Hopcroft, *On edge coloring bipartite graphs*, SIAM J. Comput., 11 (1982), pp. 540–546.

[2] J. Csima and L. Lovász, *A matching algorithm for regular bipartite graphs*, Discrete Appl. Math., 35 (1992), pp. 197–203.

[3] S. Even, A. Itai, and A. Shamir, *On the complexity of timetable and multicommodity flow problems*, SIAM J. Comput., 5 (1976), pp. 691–703.

[4] H. N. Gabow, *Using Euler partitions to edge color bipartite multigraphs*, Internat. J. Comput. Inform. Sci., 5 (1976), pp. 345–355.

[5] H. N. Gabow and O. Kariv, *Algorithms for edge coloring bipartite graphs*, in Conference Record of the Tenth Annual ACM Symposium on Theory of Computing, 10th STOC, San Diego, CA, May 1–3, 1978, Association for Computing Machinery, New York, 1978, pp. 184–192.

[6] H. N. Gabow and O. Kariv, *Algorithms for edge coloring bipartite graphs and multigraphs*, SIAM J. Comput., 11 (1982), pp. 117–129.

[7] I. G. Holyer, *The NP-completeness of edge-colouring*, SIAM J. Comput., 10 (1981), pp. 718–720.

[8] J. Hopcroft and R. M. Karp, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM J. Comput., 2 (1973), pp. 225–231.

[9] D. König, *Graphok és alkalmazásuk a determinánsok és a halmazok elméletére*, Mathematikai és Természettudományi Értesitö, 34 (1916), pp. 104–119 (in Hungarian); *Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre*, Math. Ann., 77 (1916), pp. 453–465 (in German).

[10] A. Schrijver, *On the number of edge-colourings of regular bipartite graphs*, Discrete Math., 38 (1982), pp. 297–301.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 847–863

ROW-MAJOR SORTING ON MESHES*

JOP F. SIBEYN†

Abstract. In all recent near-optimal sorting algorithms for meshes, the packets are sorted with respect to some snake-like indexing. In this paper we present deterministic algorithms for sorting with respect to the more natural row-major indexing.

For 1-1 sorting on an $n \times n$ mesh, we give an algorithm that runs in $2 \cdot n + o(n)$ steps, matching the distance bound, with maximal queue size five. It is considerably simpler than earlier algorithms. Another algorithm performs k-k sorting in $k \cdot n/2 + o(k \cdot n)$ steps, matching the bisection bound.

Furthermore, we present *uniaxial* algorithms for row-major sorting. We show that 1-1 sorting can be performed in $2\frac{1}{2} \cdot n + o(n)$ steps. Alternatively, this problem is solved with maximal queue size five in $4\frac{1}{3} \cdot n$ steps, without any additional terms.

Key words. parallel computation, meshes, sorting, row-major indexing, uniaxial routing

AMS subject classification. 68Q22

PII. S009753979427011X

1. Introduction.

Various models for parallel machines have been considered. One of the best-studied machines with a fixed interconnection network is the *mesh*. In this model the processing units (PUs) form an array of size $n \times n$ and are connected by a two-dimensional grid of communication links.

Problems. The problems concerning the exchange of packets among the PUs are called *communication problems*. The packets must be sent to their destinations such that at most one packet traverses any wire during a single step. Quality measures are *run time*, which is the maximum number of steps T a packet may need to reach its destination, and *queue size* Q, the maximum number of packets any PU may have to store.

Routing is the basic communication problem. In this problem the packets have a known destination. The routing problem in which every PU is the source and destination of k packets is called the k-k *routing* problem. 1-1 routing is commonly known as *permutation routing*.

Sorting is, next to routing, one of the most considered communication problems. Several variants of the problem have been studied. In the 1-1 sorting problem, each PU initially holds a single packet, where each packet contains a key drawn from a totally ordered set. The packets have to be rearranged such that the packet with the key of rank i is moved to the PU with index i for all i. In the k-k sorting problem, each PU is the source and destination of k packets.

Scattering is a weak variant of sorting: the packets should be rearranged such that as few as possible packets with the same key stand in the same column. It is a subroutine of many deterministic algorithms for other communication problems, and often their queue size linearly depends on the quality of the scattering algorithm. Scattering can be performed by sorting the packets in row-major order.

*Received by the editor June 22, 1994; accepted for publication (in revised form) January 21, 1997; published electronically September 22, 1998.
http://www.siam.org/journals/sicomp/28-3/27011.html
†Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany (jopsi@mpi-sb.mpg.de, http://www.mpi-sb.mpg.de/∼jopsi).

TABLE 1
Run times for k-k sorting in row-major order. For large n, the lower-order terms are omitted.

	Uniaxial		Biaxial
k	All n	Large n	Large n
1	$4\frac{1}{3} \cdot n$	$2\frac{1}{2} \cdot n$	$2 \cdot n$
2	$5\frac{1}{4} \cdot n$	$3 \cdot n$	$2\frac{1}{2} \cdot n$
k	$(7/4 \cdot k + 6) \cdot n$	$k \cdot n$	$k/2 \cdot n$

Models and indexings. The model in which the PUs can communicate with all their neighbors at the same time, sending and receiving up to four packets in a single step, is called the *MIMD* model. Alternatively the PUs may send only in a specific direction during any step. This is called the *SIMD* model. In a *half-MIMD* all PUs can either send and receive packets along the horizontal or along the vertical connections. Algorithms running on a half-MIMD are called *uniaxial*. Algorithms for the MIMD will be called *biaxial*.

If MIMD algorithms are directly run on an SIMD, they are slowed down by a factor of four. Half-MIMD algorithms are slowed down by a factor of two only. Because in many cases half-MIMD algorithms are hardly slower than MIMD algorithms, it may be advantageous to design half-MIMD algorithms. Another feature is that on an MIMD two of these algorithms can be perfectly overlapped. This is a basic observation underlying many algorithms involving some kind of "coloring" [12, 10, 21, 6].

Several recent sorting algorithms [2, 11, 8] were designed for (blocked) snake-like row-major indexings. However, in many cases it is desirable to have the packets in the more natural row-major or column-major order. Furthermore, sorting in snake-like order is unsuitable for scattering.

In the "one-packet" model considered by Schnorr and Shamir [18], the best-known upper bound for row-major sorting is higher than for sorting in snake-like row-major order. In our model a PU may hold a constant number of packets and packets may be copied. The results of this paper demonstrate that in this model, sorting in row-major order is not substantially harder than sorting in snake-like order.

Results. This paper gives numerous new results for row-major sorting, summarized in Table 1. The queue sizes for 1-1 and 2-2 sorting range between four and nine, and in the k-k sorting from k to $k+2$.

Theoretically the results for large n are the most appealing. So far, the fastest biaxial row-major sorting algorithm has $T = 2\frac{1}{4} \cdot n + o(n)$ and $Q = \mathcal{O}(1)$. It was recently designed by Krizanc and Narayanan [9]. However, this algorithm works only for the subproblem that all the keys are 0 or 1 (though some extension seems possible). The first near-optimal sorting algorithm,[1] $T = 2 \cdot n + o(n)$, was presented by Kaklamanis and Krizanc [2]. The algorithm is randomized and sorts the packets in blocked snake-like row-major order. Kaufmann, Sibeyn, and Suel [8] came up with a deterministic version. These algorithms are considerably more involved than the algorithm of this paper, and have queue sizes around 20. The best uniaxial row-major sorting algorithm so far appears to be a modification of the algorithm of Schnorr and Shamir [18]. It takes $4 \cdot n + o(n)$ steps. The first near-optimal algorithm for k-k sorting was discovered by Kaufmann and Sibeyn [7]. Then in [11] by Kunde and slightly later also in [8], deterministic versions of this randomized algorithm were described. All

[1] An algorithm is called near-optimal if its time consumption equals a lower-bound plus lower-order terms.

these algorithms use blocked snake-like row-major indexings. In this paper we present the first near-optimal algorithm for k-k sorting in row-major order.

Most current communication algorithms strive for $T = \alpha \cdot n + o(n)$, with α as small as possible. This completely neglects the fact that actual meshes tend to be of fairly moderate sizes, for which the $o(n)$ often dominates. In this paper we also aim for algorithms with a routing time without hidden terms. A sorting time that can be expressed as $T \leq \alpha \cdot n$ for all n also has theoretical relevance in recursive or divide-and-conquer algorithms, where the effective size of the network decreases, and this α may be decisive for the overall performance [21].

The remainder of the paper is organized as follows. In section 3 we give the algorithms for uniaxial row-major sorting for all n. Then we introduce in section 4 the "desnakification" of the k-k sorting algorithm for large n. This powerful technique is then applied in section 5 and culminates in the near-optimal 1-1 sorting algorithm.

2. Preliminaries.

Basics of routing and sorting. We speak of *edge contention* when several packets residing in a PU have to be routed over the same connection. Contentions are resolved using a priority scheme. We apply the *farthest-first strategy*, which gives priority to the packet that has to go farthest. For one-dimensional sorting we apply a suitable variant of an odd-even transposition sort. For the analysis of the routing on higher-dimensional meshes we need the "routing lemma" for routing a distribution of packets on a one-dimensional mesh [15, 7], and the corresponding result on sorting. Define for a given distribution of packets over the PUs $h_{\text{right}}(i,j) = \#\{$*packets passing from left to right through both* P_i and $P_j\}$, where P_i denotes the PU with index i. Define $h_{\text{left}}(j,i)$ analogously.

LEMMA 1. *Routing a distribution of packets on a linear array with n PUs, using the farthest-first strategy, takes* $\max_{i<j}\{\max\{h_{right}(i,j), h_{left}(j,i)\} + j - i - 1\}$ *steps. This bound is sharp. When the packets are evenly distributed, then the same bound can be achieved for sorting.*

Because of the distance a packet may have to go, $2 \cdot n - 2$ steps is a lower bound for any general routing or sorting problem on the two-dimensional mesh. We call this the *distance bound*. Because of the number of packets that may have to pass from one half of the mesh into the other half over only n connections, $k \cdot n/2$ steps is a lower bound for any k-k routing or sorting problem. This is called the *bisection bound*.

A 0-1 *distribution* is a distribution of packets that all have key 0 or 1. In a 0-1 distribution a row is called *dirty* if it contains both zeros and ones. In our analysis we frequently use the "0-1 lemma" (see [13]), which states that under light conditions a sorting algorithm is correct if it sorts any 0-1 distribution.

Indexings. The PU at position (i,j) is denoted $P_{i,j}$. Here $0 \leq i,j \leq n-1$, and position $(0,0)$ lies in the upper-left corner. In the row-major indexing, $P_{i,j}$ has index $i \cdot n + j$; in the column-major indexing, it has index $i + j \cdot n$; and in the reversed row-major indexing, it has index $i \cdot n + (n-j)$. In the snake-like row-major indexing, the indexing of the odd rows is reversed. For a given indexing the PU with index i, $0 \leq i \leq n^2 - 1$, is denoted P_i. A row i is said to be *sorted rightward* if the packets stand in increasing order from $P_{i,0}$ to $P_{i,n-1}$. Analogously, rows can be sorted *leftward* and columns *downward* or *upward*.

For k-k sorting there are several natural ways to index the $k \cdot n^2$ destination locations. In a *layered* indexing, location r in P_i has index $r \cdot n^2 + i$. Our default is a *nonlayered* indexing, under which location r in P_i, $0 \leq r < k$, $0 \leq i < n^2$, has index

0 16	1 17	2 18	3 19
4 20	5 21	6 22	7 23
8 24	9 25	10 26	11 27
12 28	13 29	14 30	15 31

0 1	2 3	4 5	6 7
8 9	10 11	12 13	14 15
16 17	18 19	20 21	22 23
24 25	26 27	28 29	30 31

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

FIG. 1. *Row-major indexings, for $k = 2$, $n = 4$: layered, nonlayered, and semilayered, respectively.*

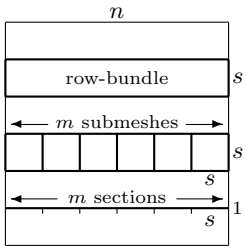


FIG. 2. *Subdivisions for the case $s = n/6$, $m = 6$.*

$k \cdot i + r$. A nonlayered row-major indexing is an indexing as if we have an $n \times k \cdot n$ mesh with row-major indexing. In our row-major sorting we use a *semilayered* indexing, under which location $P_{i,j}$ has index $(i + r) \cdot n + j$. This is as if we have an $n \times k \cdot n$ mesh with row-major indexing. These indexings are illustrated in Figure 1.

DEFINITION 1. *An indexing is* continuous *if for all i, $0 \leq i \leq n-2$, P_i is adjacent to P_{i+1} in the mesh. An indexing is* piecewise continuous *with parameter s if for every i, $0 \leq i < n^2$, there is an interval $\mathcal{I}_i \subset [0, n^2 - 1]$, with $i \in \mathcal{I}_i$ and $\#\mathcal{I}_i \geq s$, such that P_j is adjacent to P_{j+1} for all $j, j + 1 \in \mathcal{I}_i$.*

The row-major indexing is piecewise continuous with parameter n, but not continuous. The snake-like row-major indexing is continuous. One of the achievements of this paper is to show that for near-optimal sorting a *piecewise*-continuous indexing suffices.

Subdivisions. The mesh is divided into regular $s \times s$ submeshes. Let $m = n/s$. The submeshes $B_{i,j}$, $0 \leq i, j \leq m-1$, are indexed as the PUs starting with $(0, 0)$ in the upper-left corner. *Row-bundle i* consists of the PUs in $\cup_{j=0}^{m-1} B_{i,j}$. *Column-bundle j* consists of $\cup_{i=0}^{m-1} B_{i,j}$. *Section l*, S_l, $0 \leq l \leq m \cdot n - 1$, consists of the PUs with index in $\{s \cdot l, \ldots, s \cdot (l + 1) - 1\}$. Under a row-major indexing the sections regularly subdivide the rows and the submeshes. All subdivisions are depicted in Figure 2.

DEFINITION 2. *An* m-way merge *is a procedure that turns m^2 sorted submeshes into a sorted $n \times n$ mesh.*

3. Uniaxial sorting for small n.

This section is practically the most important. We present a variety of sorting algorithms that have no additional terms in their time consumptions.

3.1. Powers of two.

LEMMA 2. *On 2×2 meshes, uniaxial sorting in arbitrary order takes three steps, with queue size two.*

Proof. Perform gossiping (all-to-all routing) along rows and then along columns. This takes three steps. The PUs that finally should hold the packets with rank 0 and

1 need to conserve only the two smallest packets, the other PUs only the two largest packets. □

3.1.1. Larger n. For $n = 2^l$, $l > 1$, we use an optimized merge-sort algorithm combining several recent techniques and adding some new ideas. The first merge-sort algorithm with the optimal time *order* was given by Thompson and Kung [19]. Initially we have four sorted $n/2 \times n/2$ submeshes: those in the left half in row-major order; those in the right half in reversed row-major order. Then we perform the following.

ALGORITHM MERGE.

1. In the left half, shift the packets $n/4$ steps to the right. In the right half, shift the packets $n/4$ steps to the left.

2. In the central $n/2$ columns, sort the packets downward.

3. Copy the smallest packet in every $P_{i,j}$, $0 < i \leq n-1$, $n/4 \leq j \leq 3/4 \cdot n - 1$, to $P_{i-1,j}$. Copy the largest packet in every $P_{i,j}$, $0 \leq i < n-1$, $n/4 \leq j \leq 3/4 \cdot n - 1$, to $P_{i+1,j}$.

4. In every row, sort the section of the row that lies in the central $n/2$ columns. If this submesh is going to be the right half of a larger mesh in the next merge, then the sorting is leftward, otherwise rightward.

5. Throw away the packets in $P_{i,j}$ with $j \in [n/4, 3/8 \cdot n - 1] \cup [5/8 \cdot n, 3/4 \cdot n - 1]$. For any $P_{i,j}$, with $3/8 \cdot n \leq j \leq 5/8 \cdot n - 1$, send the packet with rank r, $0 \leq r \leq 3$, to $P_{i,4\cdot(j-3/8\cdot n)+r}$.

We analyze the routing time and the correctness of MERGE. Step 1 takes $n/4$ steps, step 2 can be performed in n steps, and step 3 takes a single step. This step can easily be made to coincide with the last step of the sorting. Its purpose is expressed by the following lemma.

LEMMA 3. *After step* 2 *all packets that actually should be in a row can be found either in the row itself, or among the smallest packets of the row below, or among the largest packets of the row above.*

Proof. First we consider a modified problem. Suppose that initially four $n/2 \times n/2$ submeshes stand above each other in an $2 \cdot n \times n/2$ mesh. Two of these submeshes are sorted in row-major order, the other two in reversed row-major order. Consider a 0-1 distribution. It is easy to check that after sorting the columns of this mesh, there are at most two dirty rows. These dirty rows can be resolved as follows: copy every row to the row above and the row below; sort the rows; spread the packets from the central $n/3$ columns. In the real problem every two rows of the high and narrow mesh are compressed in one row in which every PU in the center holds two packets. □

LEMMA 4. *Step* 4 *can be performed in* $3/4 \cdot n$ *steps.*

Proof. For the number of required steps we analyze the worst possible 0-1 distributions after step 2. These are of the form given on the left in Figure 3. After step 3 this gives a distribution as on the right in Figure 3. According to Lemma 1, sorting this row takes $3/4 \cdot n$ steps. □

Finally, step 5 takes $3/8 \cdot n$ steps.

LEMMA 5. MERGE *takes at most* $2^3/_8 \cdot n$ *steps.*

3.1.2. Improvement by overlapping. In MERGE, steps 4 and 5 involve routing along the same axis. So, we might overlap these steps, without impairing the uni-axiality of the algorithm. The central observation is that the packets to throw away are known well before the end of step 4. After throwing them away, we can proceed with a combination of odd-even transposition sort and routing packets outward: in

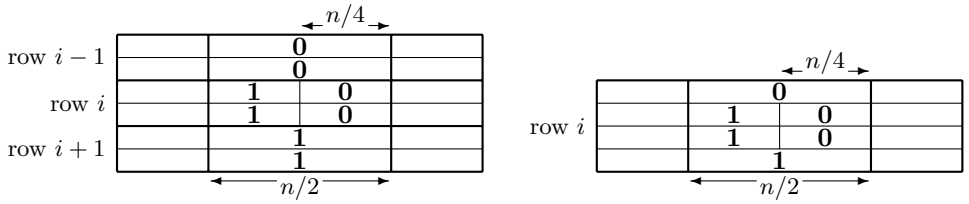
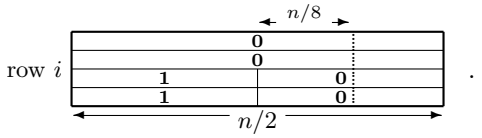


FIG. 3. *Left: worst case distribution after step* 2; *right: situation after step* 3.

the same step that we are sure which packets to throw away, we also know the largest surviving packet. One step later we know the second largest, and so on. Without further comparison these packets can be routed to their destinations, reducing the maximal distance the packets may have to travel after the end of the sorting.

LEMMA 6. *In step* 4 *all packets that will be thrown away have reached their destination region in* $5/8 \cdot n$ *steps.*

Proof. We consider the packets that will be thrown away on the high side in some row i. Assume that the sorting is rightward. If there are more than $n/2$ candidates (because a critical key occurs more than once), then some of these packets stand more to the right than necessary. So, let there be precisely $n/2$ packets with key 1, while all the others have key 0. According to Lemma 1 we must analyze how far the ones can stand to the left. Notice that a row that holds both zeros and ones is a dirty row. There are at most two dirty rows. This gives the following worst case:



It takes $n/8 + n/2$ steps until all ones have drifted into the region on the right. □

After $t = 5/8 \cdot n$ we would like to discard in every row i the packets that do not belong in it. Suppose that the sorting is rightward. $P = P_{i,5/8 \cdot n-1}$ is the rightmost PU that preserves its packets, and $P' = P_{i,5/8 \cdot n}$ is the leftmost PU of the section in which the the largest packets are thrown away. If the transposition sort works without making copies, the packets continue to move back and forth. In that case Lemma 6 guarantees only that all packets to throw away have *reached* their destination regions, not that they actually reside there! Thus, P' has to operate carefully, to prevent it throwing away the wrong packets. A solution is to let P' throw away its largest packet after step $t-1$. Then in step t it keeps a copy of its smallest packet p', which it sends to P. After step t P' throws away all its packets, except for p when it has a smaller key than p'.

Notice that the smaller of p and p', p' say, is the largest of all surviving packets: the destination of p' is in $P_{i,n-1}$. Thus, p' can be sent toward its destination, while the sorting in the central part of the row continues. It is easy to check that the second, third, ..., largest packet ultimately arrives in P in step $t+1$, $t+2$, ... (consider a worst case distribution in which the largest surviving packets reside as far to the left as possible). So, P continues to send its largest packets to the right without receiving packets from there. It sends its smallest packets leftward as long as it holds more than one packet. After this its left neighbor takes over its role of "frontier" PU. After the sorting has finished, all packets are routed as in step 5 of MERGE-m. This concludes our description of the modified steps 4 and 5. We summarize the main points:

4′. For all i, $0 \leq i < n$, until step $5/8 \cdot n$, sort the packets in the central $n/2$ PUs of

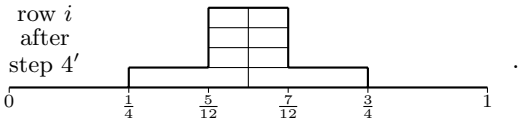
row i. Throw away the packets that stand outside the central $n/4$ columns. A PU that holds more than one packet continues to sort. A PU in the left (right) half that holds only one packet sends this packet leftward (rightward).

5′. Route the packets in row i to their destinations.

Step 4′ takes as long as before: the sorting is influenced in no way by the action going on in the periphery. On the other hand, step 5′ is substantially cheaper than step 5.

LEMMA 7. *Step 5′ can be performed in $n/4$ steps.*

Proof. Suppose that the sorting is rightward. In step 4′, the packets that do not belong in some row i are thrown away after $5/8 \cdot n$ steps. The remaining packets stand concentrated in the central $n/4$ PUs. From now on the packets spread out of this region without delay until the end of the sorting after $3/4 \cdot n$ steps. Then the situation is as follows (omitting factors n):



Because the packets are properly arranged by now, none of them still has to travel farther than $n/4$ steps. □

Combining Lemma 7 with the earlier results gives the following.

LEMMA 8. *The improved version of* MERGE *takes at most $2^{1}/_{4} \cdot n$ steps. The queue size is at most four.*

3.1.3. Sorting. Starting with sorted 2×2 meshes, MERGE can be used repeatedly for sorting on an $n \times n$ mesh. Call this algorithm SORT. We have the following result.

THEOREM 1. *For all $n = 2^l$,* SORT *performs row-major sorting on an $n \times n$ mesh in $4^{1}/_{2} \cdot n$ steps.* SORT *is uniaxial, and the queue size is four.*

Proof. Summing the number of steps required for all merges, we find that the sorting takes less than $3 + 2^{1}/_{4} \cdot (4 + 8 + \cdots + n) < 2^{1}/_{4} \cdot n \cdot \sum_{i=0} 2^{-i}$ steps. □

3.2. Powers of two, three, ... We derived an efficient 1-1 sorting algorithm for $n = 2^l$. However, in practice, processor networks may not have different side lengths. Furthermore, some algorithms in which sorting is used as a subroutine, e.g., the algorithms of [21], specifically require that $n = m^l$ for some $m \neq 2$. In principle we could use SORT by rounding n up to the nearest power of 2. But this might give sorting times that are almost twice as large as necessary. In this section we present m-way merge algorithms, which perform well for $m \leq 5$. By combining them, we can efficiently sort $n \times n$ meshes for arbitrary n.

***m* × *m* meshes.** The following algorithm efficiently sorts an $m \times m$ mesh in row-major order:

1. In all rows i, concentrate the packets in $P_{i,\lfloor m/2 \rfloor}$.
2. Sort the packets in column $\lfloor m/2 \rfloor$ downwards.
3. In all rows i, spread the packets over the row.

LEMMA 9. *Uniaxial sorting on $m \times m$ meshes can be performed in $m^2/2 + m$ steps, for m even, and $m \cdot (m+1)/2$ steps, for m odd. The queue size is m.*

Proof. Steps 1, 2, and 3 take $\lfloor m/2 \rfloor$, $m \cdot \lfloor m/2 \rfloor$, and $\lfloor m/2 \rfloor$ steps, respectively. □

TABLE 2
Run times and queue sizes of uniaxial row-major sorting algorithms for $n = 2 \cdot m^l$.

m	2	3	4	5	6
Q	2	5	6	9	10
T	$5\frac{1}{2} \cdot n$	$4\frac{1}{3} \cdot n$	$4\frac{1}{2} \cdot n$	$4.61 \cdot n$	$4.65 \cdot n$

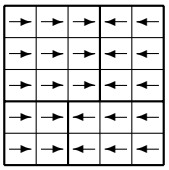


FIG. 4. *For merging* 25 *sorted submeshes, the rightward (leftward) arrows indicate submeshes that are sorted in (reversed) row-major order.*

Larger n. For performing an m-way merge for $m \geq 3$, we can proceed as in MERGE: wipe all submeshes together, sort the columns, etc. However, for algorithms of this type the number of dirty rows equals $\lceil m^2/2 \rceil$, which leads to long queues, and rapidly growing time to resolve them with increasing m. It is better to first sort the row-bundles, then to merge the sorted row-bundles. In this way the number of dirty rows is limited to $\lceil m/2 \rceil$. For $m \geq 3$ this approach is faster. Proceeding along these lines, we get the following theorem.

THEOREM 2. *For all $n = 2 \cdot m^l$, uniaxial row-major sorting on $n \times n$ meshes can be performed with the time consumptions and queue sizes given in Table* 2.

Proof. For numbers of the form $2 \cdot m^l$, first the 2×2 submeshes are sorted, then we repeatedly perform an m-way merge. At the start of an m-way merge the $n/m \times n/m$ submeshes are appropriately sorted. For even m, in every row-bundle $m/2$ submeshes are sorted in row-major order and $m/2$ in reversed row-major order. For odd m, $\lceil m/2 \rceil$ submeshes are sorted in row-major order in the highest $\lceil m/2 \rceil$ row-bundles, and $\lfloor m/2 \rfloor$ in the lowest $\lfloor m/2 \rfloor$ row-bundles. The other submeshes are sorted in reversed row-major order. See Figure 4 for an example. Then we sort the row-bundles and subsequently the column-bundles. The complete algorithm and its analysis can be found in [20]. Again a reduction of the routing time is achieved by overlapping the phases, as was done in section 3.1.2. □

Sorting on $n \times n$ meshes for arbitrary n can be performed by approximating n by the closest number of the form $2^{l_2} \cdot 3^{l_3} \cdot 5^{l_5}$, and then using the basic two-, three-, and five-way merges in the optimal order, performing the most efficient ones in the final merges, when the submeshes are large. In this way we get the following result.

THEOREM 3. *Uniaxial row-major sorting on $n \times n$ meshes can be performed in less than $4.75 \cdot n$ steps for all n. The queue size is at most nine.*

Proof. The result follows by estimating the maximum factor between n and the smallest $n' > n$ that can be written as $n' = 2^{l_2} \cdot 3^{l_3} \cdot 5^{l_5} \cdot 7^{l_7} \geq n$. Here $l_7 \leq 1$, and if $l_2 = 0$, then $l_5 + l_7 \leq 1$. The sorting is performed in the time required for an $n' \times n'$ mesh. Details are given in [20]. □

3.3. k-k sorting. We present an algorithm for uniaxial k-k sorting in row-major order. Asymptotically optimal performance is achieved by the uniaxial version of the algorithm of section 4, which requires $\max\{4 \cdot n, k \cdot n\} + \mathcal{O}((k \cdot n)^{5/6})$. But the algorithm presented here is far better for small n. We assume that $n = 2^l$.

Four $n/2 \times n/2$ submeshes are sorted in semilayered row-major order on the left,

TABLE 3
Run times and queue sizes for uniaxial k-k sorting in row-major order.

k	T	Q
2	$5\frac{1}{4} \cdot n$	4
3	$7 \cdot n$	5
4	$8\frac{2}{3} \cdot n$	6
k	$(7/4 \cdot k + 6) \cdot n$	$k + 2$

and semilayered reversed row-major order on the right. The merging is almost the same as MERGE of section 3.1.

ALGORITHM KKMERGE.

1. $P_{i,j}$, $0 \le i, j < n$, sends its packets with rank r, $0 \le r < k$, such that $\text{odd}(k \cdot i + r + j)$, to $P_{i,(j+n/2) \bmod n}$.

2. In all columns, sort the packets downward.

3. In every $P_{i,j}$, $0 < i \le n-1$, $0 \le j \le n-1$, copy the smallest packet to $P_{i-1,j}$. In every $P_{i,j}$, $0 \le i < n-1$, $0 \le j \le n-1$, copy the largest packet to $P_{i+1,j}$.

4. Sort the rows. If this submesh is going to be the left half of a larger mesh in the next merge, then the sorting is rightward, otherwise leftward.

5. In every row, throw away the n packets with the smallest and the n packets with the largest indices. If this is the final merge step, then spread the remaining $k \cdot n$ packets that stand in every row. Else route the packets to the destinations as given by step 1 of the next merge, and continue with step 2.

Analyzing the algorithm step by step, and partially overlapping step 4 and step 5, we obtain (see [20]) the following lemma.

LEMMA 10. *An intermediate application of* KKMERGE *can be performed in* $(5 \cdot k^2 + 14 \cdot k + 4 - \min\{k^2, 2 \cdot k + 8\})/(4 \cdot k + 8) \cdot n$ *and the final application in* $(2 \cdot k^2 + 12 \cdot k + 4)/(4 \cdot k + 8) \cdot n$ *steps.*

Let KKSORT1 be the k-k sorting algorithm based on KKMERGE.

THEOREM 4. *For all* $k \ge 2$, KKSORT1 *performs uniaxial* k-k *sorting in row-major order on* $n \times n$ *meshes in* $(7 \cdot k^2 + 26 \cdot k + 8 - \min\{k^2, 2 \cdot k + 8\})/(4 \cdot k + 8) \cdot n$ *steps, with queue size* $k + 2$.

Proof. We start with sorted PUs. It takes $k/2$ steps to obtain the situation at the beginning of step 2 of the merge in 2×2 meshes. Thus, the general estimate for k-k sorting on $n \times n$ meshes is $k/2 + (5 \cdot k^2 + 14 \cdot k + 4 - \min\{k^2, 4 \cdot k + 12\})/(4 \cdot k + 8) \cdot (2 + 4 + \cdots + n/2) + (2 \cdot k^2 + 12 \cdot k + 4)/(4 \cdot k + 8) \cdot n$. □

From Theorem 4 we computed the results in Table 3. For small n they are extremely competitive, even though asymptotically uniaxial k-k sorting can be performed almost twice as fast. Applying KKMERGE for $k = 1$ gives a good alternative for $n = 2^l$: $T = 5 \cdot n$ and $Q = 2$.

3.4. Minimizing the queue sizes. It may be desirable to have minimal queue size even at the expense of slightly more routing steps. For example, if in an algorithm, which has $Q = q$ for some structural reason, some local sorting operations are used as subroutines, then we do not want to take Q larger just because of this sorting (see [5]). In this section we give a general idea for minimizing the queue sizes of the presented algorithms.

General idea. The core of all presented algorithms consists of steps of the following type:

1. Rearrange the packets within the rows.

2. Sort (sections of) the columns.
3. Copy the q smallest packets of every PU to its upper neighbor, and the q largest to its lower neighbor.
4. Sort sections of width s of the rows. Throw away the $q \cdot s$ smallest and largest packets.

The queue size depends on the degree of concentration c after step 1, and the number q: $Q = c + 2 \cdot q$. The degree of concentration is an essential feature of the algorithm and was chosen carefully to minimize its run time. q equals the number of dirty rows minus one. However, there is no need to clean away all dirt in a single operation. Repeating the following steps instead of steps 3 and 4, we can obtain $Q = c$. An additional advantage is that no packets are copied anymore. We assume that the sorting is rightward.

3′. In every section, spread the $s/2$ smallest packets over the leftmost $s/2$ PUs, and the $s/2$ largest packets over the $s/2$ rightmost PUs. Call these packets *active*. Compensate for the concentration in the middle by pushing the packets that follow (precede) the actives in rank rightward (leftward). Shift the actives in the right half one row down.
4′. Sort the actives in every section. Shift the rightmost $s/2$ actives one row up. Sort the sections.

The number of iterations of steps 3′ and 4′ follows from the distribution that arises after step 2. Before the first application the packets in every row should be sorted. As the algorithm is given we get $Q = c+1$ in the right halves of the sections in row 0. In order to get $Q = c$, we should shift one nonactive packet up from every PU in the right half of row 0 at the end of step 3′. These are returned in step 4′.

LEMMA 11. *One iteration of steps* 3′ *and* 4′ *on a section of length* s *in which every PU holds* c *packets takes* $3/2 \cdot s - s/c + 3$ *steps.*

Proof. The spreading in step 3′ takes $s/2 - s/(2 \cdot c)$ steps. For sorting the actives it is essential that the packets in each half are already sorted. Therefore it can be performed in $s/2 + 1$ steps with an ordinary odd-even transposition sort without making copies (if this is desirable), and with one step fewer if copying is allowed. Sorting the sections takes as much as the spreading: no packet has to travel more than $s/2 - s/(2 \cdot c)$, and no connection is heavily loaded. □

***k*-*k* sorting.** For the k-k merge, it can be shown that steps 3′ and 4′ need to be performed only once. This means that at little extra cost, the queue size of the k-k sorting algorithm can be reduced to k.

1-1 sorting. In our machine model a packet that only passes a PU is not inserted into its queue but transferred directly from its in-buffer to its out-buffer. In addition, it is possible to compare a buffered packet with a queued one, and to exchange them. In such a model the given algorithm can be applied for 1-1 sorting with $T = 5\frac{1}{2} \cdot n + \log n$ and $Q = 1$. Alternatively, the algorithm can be applied for sorting in the *one-packet model* in which every PU holds one packet at all times and the connections act as comparators. The algorithm of Schnorr and Shamir [18] requires approximately $4 \cdot n + 20 \cdot n^{2/3}$ steps for sorting in row-major order. For all mesh sizes smaller than 1000×1000, our result gives a significant improvement over this.

LEMMA 12. *For* $k = 1$, *the queue size of* MERGEKK *can be reduced to* 1. *An intermediate application takes* $3\frac{1}{4} \cdot n + 1$, *the final application* $2\frac{1}{4} \cdot n + 1$ *steps.*

Proof. Because the packets within the submeshes are already sorted, step 2 takes only $3/4 \cdot n$ steps. Sorting the sections takes $n - 1$, steps 3′ 1, and Step 4′, without

the final sorting, $n/2 + 1$ steps. In the final merge no more steps are needed. In an intermediate merge packets still may travel a distance n. □

THEOREM 5. KKSORT1 *performs uniaxial* 1-1 *sorting in row-major order on* $n \times n$ *meshes in* $5^1/_2 \cdot n + \log n$ *steps with queue size* 1.

Applying a technique called "vibration" [1, 14], the same bound can be achieved for "hot-potato" sorting (a paradigm in which no queuing is allowed). The hot-potato routing algorithms in [3, 4] apply such a sorting as a subroutine.

4. *k*-*k* sorting for large *n*. Earlier algorithms for k-k sorting [7, 11, 8] work according to the following basic scheme:

1. Route all packets to random destinations.
2. Estimate the ranks of the packets by local comparisons.
3. Route all packets to their preliminary destinations.
4. Rearrange the packets locally to bring them to their final destinations.

In the version of [8], the mesh is divided into $s \times s$ submeshes with $s = n^{2/3}/k^{1/3}$, and the randomization of step 1 is replaced by sorting the packets in the submeshes and unshuffling them regularly over the submeshes. Step 2 is performed by sorting within the submeshes. Step 4 is performed by sorting pairs of adjacent submeshes. On an MIMD the total sorting time is $k \cdot n/2 + \mathcal{O}(k^{2/3} \cdot n^{2/3})$. As the algorithm is given, step 4 requires that the indexing is continuous. In this section we introduce a novel technique, *desnakification*, to handle the final local sorting such that piecewise-continuous indexings are allowed.

The continuity of the indexing is required for sorting together pairs of submeshes with consecutive indices. Sorting such pairs of submeshes is necessary because the estimate of the rank in step 2 is accurate only up to one submesh. So, it may happen that after step 3, a packet with destination in submesh B_i actually resides in the preceding submesh B_{i-1} or the succeeding submesh B_{i+1}. However, this is easy to overcome: send for all packets p, of which the destination submesh is not uniquely determined, a copy to both submeshes in which its destination may lie. Now it is sufficient to sort within the submeshes. If for B_i the numbers cl of packets that actually belong in B_{i-1} and ch of packets that belong in B_{i+1} are exactly known, then the smallest cl and largest ch packets in B_i are thrown away, and the remaining packets are redistributed within B_i. All this is very similar to the way dirty rows are resolved in the algorithms of section 3. The only possible problem is that routing the copies might slow down the algorithm.

We work the desnakification out in detail for biaxial sorting. In order to bound the number of copies, we must take the submeshes larger than in [8]. The optimal choice is $s = n^{5/6}/k^{1/6}$ and $m = n/s = k^{1/6} \cdot n^{1/6}$. We suppose that the indexing is piecewise continuous with parameter s. For the sake of a simple exposition we assume that the mesh is divided into sections of length s, each of which is fully contained in a single submesh. The algorithm proceeds as follows.

ALGORITHM KKSORT2.

1. In each submesh, sort the packets. The intermediate destination of a packet p with rank r, $0 \le r < k \cdot s^2$, lies in submesh $r \bmod m^2$. If $r \bmod (2 \cdot m^2) < m^2$, then color p white, else black.

2. In each submesh rearrange the white (black) packets such that those with intermediate destinations in column-bundle l (row-bundle l), $0 \le l < m$, stand in the columns (rows) $[l \cdot s/m, (l+1) \cdot s/m - 1]$ of the submesh.

3. From column-bundle j, $0 \le j < m$, route the white packets with intermediate destinations in column-bundle l, $0 \le l < m$, as a block to the columns $[j \cdot s/m, (j+1) \cdot s/m - 1]$ of column-bundle l. Route the black packets analogously.

4. In each submesh rearrange the white (black) packets such that those with intermediate destinations in row-bundle l (column-bundle l), $0 \le l < m$, stand in the rows (columns) $[l \cdot s/m, (l+1) \cdot s/m - 1]$ of the submesh.

5. From row-bundle i, $0 \le i < m$, route the white packets with intermediate destinations in row-bundle l, $0 \le l < m$, as a block to the rows $[i \cdot s/m, (i+1) \cdot s/m - 1]$ of row-bundle l. Route the black packets analogously.

6. In each submesh, sort the packets. The preliminary destination of a packet p with rank r, $0 \le r < k \cdot s^2$, lies in section S_l, with $l = \lfloor r \cdot m^2/(s \cdot k) \rfloor$. If $\lfloor (r \cdot m^2 - m^4)/(s \cdot k) \rfloor = l - 1$, then create a copy p' of p with preliminary destination in S_{l-1}. If $\lfloor (r \cdot m^2 + m^4)/(s \cdot k) \rfloor = l + 1$, then create a copy p' of p with preliminary destination in S_{l+1}. If r is even, then color p (and p') white, else black.

7, 8, 9, 10. Like steps 2, 3, 4, and 5, respectively, for the preliminary destinations.

11. Route the packets within the submeshes to the sections of their preliminary destinations.

12. In each section, sort the packets.

13. In each section S_l, $0 \le l \le m \cdot n - 1$, throw away the m^4 packets with the smallest keys (except for S_0), and the m^4 packets with the largest keys (except for $S_{m \cdot n - 1}$). Redistribute the remaining $k \cdot s$ packets within S_l.

If packets have the same key, then special care should be taken not to throw away both copies of a packet, while keeping both copies of another packet. A solution is to take the index of the PU where a packet started as an additional comparison criterion, to assure that all packets have different keys. The algorithm can be made uniaxial by leaving out the coloring, and applying only uniaxial local operations.

THEOREM 6. *Let* $s = n^{5/6}/k^{1/6}$. KKSORT2 *performs biaxial* k*-*k *sorting with respect to a piecewise-continuous indexing with parameter* s *in* $\max\{4 \cdot n, k \cdot n/2\} + \mathcal{O}(k \cdot s)$ *steps. The queue size is* $k + 2$.

Proof. The following facts imply the correctness of KKSORT2. In step 6, the estimate of the global rank of a packet p with rank r within its submesh, $r \cdot m^2$, is accurate up to m^4. Hence, the index of the destination PU of p is accurate up to m^4/k. Thus after step 11, a (copy of a) packet resides in its destination section. After step 11 there are m^4 packets in S_l, $0 < l \le m \cdot n - 1$, that belong in S_{l-1}, because from each of the m^2 submeshes precisely m^2 copies of packets with estimated destination in S_{l-1} are sent to S_l. Likewise there are m^4 packets in S_l, $0 \le l < m \cdot n - 1$, that belong in S_{l+1}.

For the time analysis, only the four main steps, steps 3, 5, 8, and 10, are of importance. The other steps can be performed in $\mathcal{O}(k \cdot s) = \mathcal{O}(k^{5/6} \cdot n^{5/6})$ steps. Step 3 and step 5 are very regular. It is easy to check that no connection has to transfer more than $k \cdot n/8$ packets, and that packets travel less than n steps. At the beginning of step 8, there are in every submesh exactly m^3 packets and $2 \cdot m^2$ copies of packets with destination in any section S_l, $0 < l < m \cdot n - 1$ (m^2 copies for $l = 0$ or $l = m \cdot n - 1$). Because the sections are fully contained in the submeshes, this implies that every submesh holds $m^3 \cdot n$ packets and $2 \cdot m^2 \cdot n$ copies of packets with destination in any column-bundle. This means that step 7 can be performed such that the PUs in the columns $[l \cdot s/m, (l + 2 \cdot k/m) \cdot s/m - 1]$ all hold $k + 1$ packets and the PUs in all other columns hold exactly k packets. Clearly step 8 now takes $(1 + 2/m) \cdot k \cdot n/8 = k \cdot n/8 + s/4$. Performing step 9 appropriately, the same bound can be shown for step 10.

A PU never holds more than $k/2$ packets and one copy of both colors, and thus $Q \le k + 2$. □

5. 1-1 sorting for large n. We start with a uni-axial algorithm for 1-1 sorting in row-major order. It runs in $2^1/_2 \cdot n + o(n)$ steps. Asymptotically this is much faster then the algorithms of section 3. This algorithm is obtained by combining our new insight in merge sorting and the desnakification technique, with old knowledge about sorting with splitters. In section 5.2 it is turned into a near-optimal biaxial algorithm. Without loss of generality, we assume that all packets have different keys.

5.1. Uniaxial sorting. The mesh is divided into $s \times s$ submeshes. In the algorithm of this section $s = n^{5/6}$, and $m = n/s = n^{1/6}$. We distinguish packets and *splitters*. The splitters are copies of a small subset of the packets. They are broadcast and the packets estimate their ranks by comparison with the splitters. This widely known idea (going back to work of Reischuck [16] and Reif and Valiant [17]) has been used for randomized [15, 7, 2] and deterministic [8] sorting on meshes. In the k-k sorting algorithm of section 4 we do not need splitters because the packets are fully distributed over the mesh, and thus reliable estimates of the ranks of the packets can be obtained by local comparison among the packets themselves. In the case of 1-1 sorting this does not lead to efficient algorithms. The splitters allow us to spread the necessary information rapidly, while the packets are involved in more useful operations.

Algorithm. First we give the algorithm for selecting and routing the splitters.

ALGORITHM SPLITTER-ROUTE.

1. In every submesh, sort the packets. Copy the packets with ranks $i \cdot m^2$, $0 \le i \le s^2/m^2 - 1$. These are the splitters.

2. In every submesh $B_{i,j}$, $0 \le i, j < m$, rearrange the splitters such that they stand in the positions (i', j') of $B_{i,j}$, with $i \cdot s/m \le i' < (i+1) \cdot s/m$ and $j \cdot s/m \le j' < (j+1) \cdot s/m$.

3. Send the splitters along the rows. A splitter starting in position (i', j') of $B_{i,j}$ drops copies in the positions (i', j') of $B_{i,l}$, for all $0 \le l < m$.

4. Send the splitters along the columns. A splitter starting in position (i', j') of $B_{i,j}$ drops copies in the positions (i', j') of $B_{l,j}$ for all $0 \le l < m$.

LEMMA 13. SPLITTER-ROUTE *takes* $2 \cdot n + \mathcal{O}(s)$ *steps to complete. No connection has to transfer more than* $\mathcal{O}(s)$ *packets. Finally, each PU holds precisely one splitter, and all splitters are available in every* $s \times s$ *submesh.*

Proof. Step 1 and step 2 take $\mathcal{O}(s)$ steps, step 3 and step 4 take less than n steps. The rearrangement is such that the splitters in $B_{i,j}$ stand in "subsubmesh" (i, j). After the broadcast these splitters occupy the subsubmeshes (i, j) in all submeshes: the splitters from different submeshes fit perfectly next to each other. This arrangement also assures that during step 3 and step 4 a connection has to transfer at most $m/2 \cdot s/m = s/2$ splitters. □

When splitters and packets want to use the same connection, priority is given to the splitters. By the lemma this delays the packets by at most $\mathcal{O}(s)$. For the packets we perform a kind of m-way merge algorithm.

ALGORITHM 11SORT.

1. In every submesh, sort the packets in row-major order.

2. In every submesh $B_{i,j}$, $0 \le i, j < m$, shift the packets in row l, $0 \le l < s$, to row l of $B_{i,(j+l) \bmod (m/2)}$, and copies to row l of $B_{i,(j+l) \bmod (m/2)+m/2}$.

3. In all columns, sort the packets downward.

4. In every submesh, determine for every packet its "rank," the number r, $0 \le r \le s^2$, of splitters that are smaller. The preliminary destination of a packet p with

rank r lies in section S_l, with $l = \lfloor r \cdot m^2/s \rfloor$. If $\lfloor (r \cdot m^2 - m^4)/s \rfloor = l-1$, then create a copy p' of p with preliminary destination in S_{l-1}. If $\lfloor (r \cdot m^2 + m^4)/s \rfloor = l+1$, then create a copy p' of p with preliminary destination in S_{l+1}. Discard the splitters and the (copies of) packets that have preliminary destination in the other half of the mesh.

5. In every submesh, sort the packets in column-major order on their preliminary destination column-bundles.

6. In every row, route the packets to the first PUs in their preliminary destination column-bundles that hold less than two packets.

7. In each submesh, sort the packets in row-major order on their preliminary destination section.

8. In every column, route the packets to the sections of their preliminary destinations.

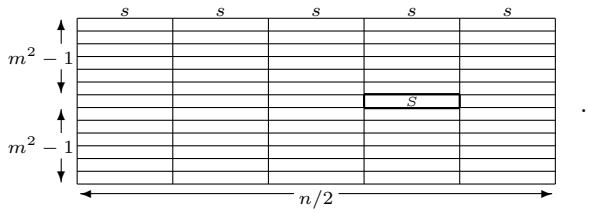
9. In every section, sort the packets.

10. In every section S_l, $0 < l \leq m \cdot n - 1$, throw away the m^4 packets with the smallest keys, and in each S_l, $0 \leq l < m \cdot n - 1$, throw away the m^4 packets with the largest keys. Redistribute the remaining $k \cdot s$ packets within S_l.

After step 3, there are in a 0-1 distribution at most m^2 dirty rows. For a general distribution this means that a packet resides at most $m^2 - 1$ rows away from its destination row. These three steps take $2 \cdot n + \mathcal{O}(s)$ steps, just as in SPLITTER-ROUTE. So, we may assume that after step 3 the splitters are available in the submeshes. Step 4, ..., step 10 resemble the final steps of KKSORT2 for $k = 1$.

Analysis. As the algorithm is given, it is not entirely correct. It is *not* true that, as in KKSORT2, exactly m^4 packets must be thrown away on both sides of every section: 11SORT orders the packets, but the sections do not necessarily hold exactly s packets. Fortunately, the numbers of packets that must be thrown away in a section on the low and high sides, respectively, can be determined.

Consider some section S and the sections from which it may receive packets after step 3:



$(2 \cdot m^2 - 1) \cdot n/2$ packets are stored in these sections, among which are the s packets with destination in S. After step 8, these s packets all reside in S, but also some packets that do not belong in S. How can we figure out which packets to keep, and which packets to throw away? Suppose that S is the lth section, $(m^2 - 1) \cdot n/s \leq l < m^2 \cdot n/s$, in the involved (whole) rows. Then finally S should hold the packets with ranks r, $l \cdot s \leq r < (l+1) \cdot s$, from among the $(2 \cdot m^2 - 1) \cdot n$ packets. Analogously to the merge algorithms of section 3, we could copy *all* packets to S, sort them, and throw away the smallest $l \cdot s$ packets and the largest $(2 \cdot m^2 - 1) \cdot n - (l+1) \cdot s$ packets. This gives a correct but very inefficient algorithm. However, it is not necessary to copy all packets to S. It is sufficient, if for each contributing section i, the counters, the numbers $under_{S,i}$ and $over_{S,i}$ of packets that are *not* sent to S because they are definitely too small or definitely too large, respectively, are known in S. The counters can easily be determined in step 4. They can be transferred to S during the subsequent

steps, in parallel with the packets. As every section sends and receives only $\mathcal{O}(m^3)$ counters in total, they can be routed without causing substantial delay. The numbers $Under_S = \sum_i under_{S,i}$ and $Over_S = \sum_i over_{S,i}$ can be computed in step 9. Finally, in step 10, the smallest $l \cdot s - Under_S$ packets and the largest $(2 \cdot m^2 - 1) \cdot n - (l+1) \cdot s - Over_S$ packets in S are thrown away, leaving exactly the s packets belonging in S. Now we get the following result.

THEOREM 7. *Uniaxial* 1-1 *sorting in row-major order can be performed in* $2^1/_2 \cdot n + \mathcal{O}(n^{5/6})$ *steps. The queue size is five.*

Proof. Let $s = n^{5/6}$. For the routing time and correctness, we only have to prove that step 6 can be performed in $n/2 + \mathcal{O}(s)$ steps. All other steps can be performed in $\mathcal{O}(s)$ steps.

The estimate of the rank of a packet, $r \cdot m^2$, is accurate up to m^4. This means that for some section S_l, only a packet (or its copy) with actual destination in some PU P_k, with $k \in [l \cdot s - 2 \cdot m^4, (l+1) \cdot s + 2 \cdot m^4]$, may get preliminary destination in S_l. Hence, at most $s^2 + 4 \cdot m^4 \cdot s$ packets have preliminary destination in any submesh $B_{i,j}$. By the sorting in step 5, they are distributed almost optimally over the rows of row-bundle i: at most $s + \mathcal{O}(m^4)$ packets stand in any row. The $m^2 \cdot s$ packets with destination in $B_{i-1,j}$ and $B_{i+1,j}$ that may stand in row-bundle i have no serious influence. This shows that step 6 can be performed as specified: no PU in $B_{i,j}$ has to receive more than two packets.

We consider the routing time of step 6. For a rightward-moving packet p, residing in some PU $P_{i,j}$ and moving to column l, with $j, l < n/2$, we are interested in the number h_l of packets within row i that go to some column k, with $k \geq l$. By the above analysis, we know that $h_l \leq n/2 - l + \mathcal{O}((n/2 - l)/s \cdot m^4) \leq n/2 - l + \mathcal{O}(s)$. p is delayed at most h_l times, and hence p finishes step 6 within $n/2 + \mathcal{O}(s)$ steps.

A PU may hold up to four (copies of) packets during step 4 and step 5. In addition step 4 can be organized such that a PU holds at most one splitter or counter. Hence, $Q \leq 5$. □

Other indexings. The algorithm is not suited for sorting with respect to any piecewise-continuous indexing: it is essential that after step 3 the packets do not have to make another long vertical move. However, the algorithm *is* correct for any piecewise indexing in which the pieces are scrambled within the rows.

2-2 sorting. For uniaxial 2-2 sorting, only step 2 has to be modified: the packets are not copied, but distributed evenly over the rows. For biaxial 2-2 sorting, we essentially apply two orthogonal versions of 11SORT. This gives the following theorem (details are provided in [20]).

THEOREM 8. *Uniaxial* 2-2 *sorting in row-major order can be performed in* $3 \cdot n + \mathcal{O}(n^{5/6})$ *steps. The queue size is five. Biaxially it takes* $2^1/_2 \cdot n + \mathcal{O}(n^{5/6})$ *steps, with a queue size of nine.*

5.2. Near-optimal biaxial sorting. Essentially 11SORT consists of three main routing phases: horizontal, vertical, and horizontal (step 2, step 3, and step 6). These phases take n, n, and $n/2$ steps, respectively. The connections between the left and right halves are not used anymore after step $n/2$. Thus it may happen that a packet p_1 that stands in column 0 after phase 1 is routed to a preliminary destination in column $n/2 - 1$ in phase 3. This is unnecessary: a copy of p_1 stands in column $n/2$. In a uniaxial algorithm this observation does not lead to a faster algorithm: there may be a packet p_2, after phase 1 in column $n/2 - 1$ and with preliminary destination in column 0, which has to travel $n/2$ steps in phase 3. On the other hand, in a

biaxial algorithm, it is possible to coalesce the phases. Then p_2 can start phase 3 after $3/2 \cdot n + \mathcal{O}(s)$ steps, and will reach its preliminary destination after $2 \cdot n + \mathcal{O}(s)$ steps.

We work out these ideas. Only step 4 is changed: instead of discarding the packets that have their destinations in the other half, we now perform the following.

In all columns j, $0 \leq j < n/2$, discard the (copies of) packets that have preliminary destination in some column j', with $j' \geq 2 \cdot j$. For $n/2 \leq j < n$, discard the packets with $j' < 2 \cdot j - n$.

Notice that by this rule again exactly one of the copies of a packet reaches every possible destination section. The steps are coalesced. Most importantly, this means that step 3 begins in column j after $n/2 + |n/2 - j|$ steps, and step 6 after $3/2 \cdot n + |n/2 - j|$ steps.

THEOREM 9. *Biaxial* 1-1 *sorting in row-major order can be performed in* $2 \cdot n + \mathcal{O}(n^{5/6})$ *steps. The queue size is five.*

Proof. A packet that starts step 6 after $2 \cdot n - d + \mathcal{O}(s)$ steps has to travel at most d steps to reach the column-bundle of its preliminary destination. We check this for a packet p that is routed in step 2 to some column j, with $j < n/2$. p starts step 6 after $2 \cdot n - j + \mathcal{O}(s)$ steps. In step 4 the preliminary destination of p is determined. p survives only when it goes to some column l, with $l < 2 \cdot j$: p has to travel at most j steps. By a refinement of the analysis in the proof of Theorem 7, it can be shown that p is not delayed more than $2 \cdot j - l$ times. Hence, step 7 can start in all submeshes after $2 \cdot n + \mathcal{O}(s)$ steps. □

In fact this algorithm is still *locally* uniaxial: every PU uses only horizontal or vertical connections.

6. Conclusion. We presented novel uniaxial and biaxial row-major algorithms for sorting on two-dimensional meshes. A tremendous improvement is our near-optimal algorithm for 1-1 sorting: it is much simpler than the earlier algorithm, it is suited for more useful indexings, it is locally uniaxial, and it has queue size five.

Future research could address (1) the optimality of the uniaxial sorting algorithm with run time $2\frac{1}{2} \cdot n + o(n)$ steps; (2) a further development of the merge-sort idea to obtain even faster sorting for all n.

REFERENCES

[1] U. FEIGE AND P. RAGHAVAN, *Exact analysis of hot-potato routing*, in Proc. 33rd Symp. on Foundations of Computer Science, IEEE, 1992, pp. 553–562.

[2] C. KAKLAMANIS AND D. KRIZANC, *Optimal sorting on mesh-connected processor arrays*, in Proc. 4th Symp. on Parallel Algorithms and Architectures, ACM, 1992, pp. 50–59.

[3] C. KAKLAMANIS, D. KRIZANC, AND S. RAO, *Hot-potato routing on processor arrays*, in Proc. 5th Symp. on Parallel Algorithms and Architectures, ACM, 1993, pp. 273–282,.

[4] M. KAUFMANN, H. LAUER, AND H. SCHRÖDER, *Fast deterministic hot-potato routing on meshes*, in Proc. 5th International Symp. on Algorithms and Computation, Lecture Notes in Comput. Sci. 834, Springer-Verlag, New York, 1994, pp. 333–341.

[5] M. KAUFMANN, U. MEYER, AND J. F. SIBEYN, *Towards practical permutation routing on meshes*, in Proc. 6th Symp. on Parallel and Distributed Processing, IEEE, 1994, pp. 664–671; Computers and Artificial Intelligence, 16 (1997), pp. 107–140.

[6] M. KAUFMANN, U. MEYER, AND J. F. SIBEYN, *Matrix transpose on meshes: theory and practice*, in Proc. 11th International Parallel Processing Symp., IEEE, 1997, pp. 315–319.

[7] M. KAUFMANN AND J. F. SIBEYN, *Randomized multipacket routing and sorting on meshes*, Algorithmica, 17 (1997), pp. 224–244.

[8] M. KAUFMANN, J. F. SIBEYN, AND T. SUEL, *Derandomizing algorithms for routing and sorting on meshes*, in Proc. 5th Symp. on Discrete Algorithms, ACM-SIAM, 1994, pp. 669–679.

[9] D. Krizanc and L. Narayanan, *Zero-one sorting on the mesh*, Parallel Process. Lett., 5 (1995), pp. 149–155.
[10] M. Kunde, *Concentrated regular data streams on grids: sorting and routing near to the bisection bound*, in Proc. 31st Symp. on Foundations of Computer Science, IEEE, 1991, pp. 141–150.
[11] M. Kunde, *Block gossiping on grids and tori: deterministic sorting and routing match the bisection bound*, in Proc. European Symp. on Algorithms, Lecture Notes in Comput. Sci. 726, Springer-Verlag, New York, 1993, pp. 272–283.
[12] T. Leighton, F. Makedon, and Y. Tollis, *A $2n-2$ step algorithm for routing in an $n \times n$ array with constant size queues*, Algorithmica, 14 (1995), pp. 291–304.
[13] T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes*, Morgan-Kaufmann, San Mateo, CA, 1992.
[14] I. Newman and A. Schuster, *Hot-potato worm routing via store-and-forward packet routing*, J. Parallel Distributed Comput., 30 (1995), pp. 76–84.
[15] S. Rajasekaran, *k-k routing, k-k sorting, and cut-through routing on the mesh*, J. Algorithms, 19 (1995), pp. 361–382.
[16] R. Reischuk, *Probabilistic parallel algorithms for sorting and selection*, SIAM J. Comput., 14 (1985), pp. 396–411.
[17] J. Reif and L. G. Valiant, *A logarithmic time sort for linear size networks*, J. ACM, 34 (1987), pp. 68–76.
[18] C. P. Schnorr and A. Shamir, *An optimal sorting algorithm for mesh connected computers*, in Proc. 18th Symp. on Theory of Computing, ACM, 1986, pp. 255–263.
[19] C. D. Thompson and H. T. Kung, *Sorting on a mesh-connected parallel computer*, Comm. ACM, 20 (1977), pp. 263–271.
[20] J. F. Sibeyn, *Desnakification of Mesh Sorting Algorithms*, Tech. Rep. MPI-I-94-102, Max-Planck Institut für Informatik, Saarbrücken, Germany, 1994.
[21] J. F. Sibeyn, B. S. Chlebus, and M. Kaufmann, *Deterministic permutation routing on meshes*, J. Algorithms, 22 (1997), pp. 111–141.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 864–889

ROBUST PROXIMITY QUERIES: AN ILLUSTRATION OF DEGREE-DRIVEN ALGORITHM DESIGN*

GIUSEPPE LIOTTA†, FRANCO P. PREPARATA‡, AND ROBERTO TAMASSIA‡

Abstract. In the context of methodologies intended to confer robustness to geometric algorithms, we elaborate on the exact-computation paradigm and formalize the notion of degree of a geometric algorithm as a worst-case quantification of the precision (number of bits) to which arithmetic calculation have to be executed in order to guarantee topological correctness. We also propose a formalism for the expeditious evaluation of algorithmic degree. As an application of this paradigm and an illustration of our general approach where algorithm design is driven also by the degree, we consider the important classical problem of proximity queries in two and three dimensions and develop a new technique for the efficient and robust execution of such queries based on an implicit representation of Voronoi diagrams. Our new technique offers both low degree and fast query time and for 2D queries is optimal with respect to both cost measures of the paradigm, asymptotic number of operations, and arithmetic degree.

Key words. geometric computing, robustness, arithmetic precision, proximity queries

AMS subject classifications. 68U05, 65Y25

PII. S0097539796305365

1. Introduction.

The increasing demand for efficient and reliable geometric software libraries in key applications such as computer graphics, geographic information systems, and computer-aided manufacturing is stimulating a major renovation in the field of computational geometry. The inadequacy of the traditional simplified framework has become apparent, and it is being realized that, in order to achieve an effective technology transfer, new frameworks and models are needed to design and analyze geometric algorithms that are efficient in a practical realm.

The real-RAM model with its implicit infinite-precision requirement has proved unrealistic and needs to be replaced with a realistic finite-precision model where geometric computations can be carried out either exactly or with a guaranteed error bound. This has motivated a great deal of research on the subject of robust computational geometry (see, e.g., [4, 12, 11, 19, 27, 28, 31, 36, 34, 39, 48, 54, 58, 21, 30, 32]). For an early survey of the different approaches to robust computational geometry the reader is referred to [38].

To a first, rough approximation, robustness approaches are of two main types: perturbing and nonperturbing. Perturbing approaches transform the given problem into one that is intended not to suffer from well-identified shortcomings; nonperturbing approaches are based on the notion of "exact" (rather than "approximate") computations, with the assumption that (bounded-length) input data are error-free. In this category falls the exact geometric computation paradigm independently advocated by

*Received by the editors June 19, 1996; accepted for publication (in revised form) January 17, 1997; published electronically September 22, 1998. This research was supported in part by U.S. Army Research Office grant DAAH04–96–1–0013, National Science Foundation grant CCR–9423847, the N.A.T.O.-C.N.R. Advanced Fellowships Programme, and EC ESPRIT Long Term Research Project ALCOM-IT contract 20244.

http://www.siam.org/journals/sicomp/28-3/30536.html

†Dipartimento di Informatica e Sistemistica, Universitá di Roma "La Sapienza", Via Salaria 113, Roma I-00198, Italy. The work of this author was performed in part while he was with the Center for Geometric Computing at Brown University (liotta@dis.uniroma1.it).

‡Center for Geometric Computing, Department of Computer Science, Brown University, 115 Waterman Street, Providence, RI 02912-1910 (franco@cs.brown.edu, rt@cs.brown.edu).

Yap [59] and by the Saarbrücken school [10], and so does our approach. Within this paradigm, we introduce the notion of *degree* of an algorithm, which describes, up to a small additive constant, the arithmetic precision (i.e., number of bits) needed by the exact-computation paradigm. Namely, if the coordinates of the input points of a degree-d geometric algorithm are b-bit integers, then, as we shall substantiate below, the algorithm may be required in some instances to perform arithmetic computations with bit precision $d(b + O(1))$.

Theoretical analysis and experimental results show that multiprecision numerical computations take up most of the CPU time of exact geometric algorithms (see, e.g., [41, 49]). Thus, we believe that, in defining the efficiency of a geometric algorithm, the degree should be considered as important as the asymptotic time complexity and should correspondingly play a major role in the design stage. In fact, the principal thrust of this paper is to present algorithm degree as a major design criterion for geometric computation. Research along these lines involves reexamining the entire rich body of computational geometry as we know it today.

In this paper, we consider as a test case a problem area, geometric proximity, which plays a major role in several applications and has recently attracted considerable attention because, due to its demands of high precision for exact computation, it is particularly appropriate in assessing effectiveness of robust approaches (see, e.g., [9, 11, 20, 29, 31, 27, 55, 32]). In particular we shall illustrate the role played by the degree criterion if one wishes to comply with the standard exact-computation paradigm.

To illustrate the approach, we recall that Voronoi diagrams are the search structures which permit answering a proximity query without evaluating all query/site distances. Therefore, given the set of sites, their Voronoi diagram is computed and supplied as a planar subdivision to a point location procedure. Assuming that the coordinates of all input data (also called *primitive* points) are b-bit integers, the coordinates of the points computed by the algorithm (referred to here as *derived* points, e.g., the vertices of a Voronoi diagram of points and segments) must be stored with a representation scheme that supports rational or algebraic numbers as data types (through multiprecision integers). Specifically, the coordinates (x, y) of a Voronoi vertex are rational numbers given by the ratio of two determinants (of respective orders 3 and 2) whose entries are integers of well-defined maximum modulus. The fundamental operation used by any point location algorithm is the point-line discrimination, which consists of determining whether the query point q is to the left or to the right of an edge between vertices v_1 and v_2. For the case of the Voronoi diagram $V(S)$, this is equivalent to evaluating the sign of a 3×3 determinant whose rows are the homogeneous coordinates of q, v_1, and v_2, a computation that needs about $6b$ bits of precision. This should be compared with the $O(n)$-time brute-force method that computes the (squares of the) distances from q to all the sites of S, and executes arithmetic computations with only $2b$ bits of precision (which is optimal).

Guided by the low-degree criterion, in this paper we present a technique—complying with the exact-computation paradigm—which uses a new point location data structure for Voronoi diagrams, such that the test operations executed in the point location procedure are distance comparisons, and are therefore executed with optimal $2b$ bits of precision. Hence, our approach reconciles efficiency with robustness and supports an object-oriented programming style where access to the geometry of Voronoi diagrams in point location queries is encapsulated in a small set of geometric test primitives. It must be pointed out that distance comparisons have already been used nontrivially for proximity search (extremal-search method [26]). However, we shall show that the latter method fails to achieve optimal degree because the search

TABLE 1

*Comparison of the degree and time of algorithms for some fundamental proximity query problems. An * denotes optimality. The new technique introduced in this paper (*point location in an implicit Voronoi diagram*) always outperforms previous methods and is optimal for* 2D *queries.*

Query	Method	Degree	Time
Nearest neighbor	brute-force distance comparison	2 *	$O(n)$
	point location in explicit Voronoi diagram	6	$O(\log n)$ *
	extremal-search method	4	$O(\log n)$ *
	point location in implicit Voronoi diagram	2 *	$O(\log n)$ *
k-nearest neighbors and circular range search	brute-force distance comparison	2 *	$O(n)$
	point location in explicit order-k Voronoi diagram	6	$O(\log n + k)$ *
	point location in implicit order-k Voronoi diagram	2 *	$O(\log n + k)$ *
Nearest neighbor among points and segments	brute-force distance comparison	6	$O(n)$
	point location in explicit Voronoi diagram	64	$O(\log n)$ *
	point location in implicit Voronoi diagram	6	$O(\log n)$ *
3D nearest neighbor	brute-force distance comparison	2 *	$O(n)$
	point location in explicit 3D Voronoi diagram	8	$O(\log^2 n)$
	point location in implicit 3D Voronoi diagram	3	$O(\log^2 n)$

is based on predicates requiring $4b$ bits of precision; moreover, the high overhead of the search technique (which uses the hierarchical polytope representation [22]) casts some doubts on the practicality of the method.

The main results of this work are summarized in Table 1. Considering, for the time being, the degree as a measure of complexity, we show that previous methods exhibit a sharp tradeoff between degree and query time. Namely, low degree is achieved by the slow brute-force search method, while fast algorithms based on point location in a preprocessed Voronoi diagram or on the extremal-search method have high degree. Our new technique gives instead both low degree and fast query time and is optimal with respect to both cost measures for queries in sets of 2D point sites.

The rest of this paper is organized as follows. In section 2, the concept of degree of a geometric algorithm is defined and a simple formalism to compute such degree is introduced. Such formalism is used in section 3 to analyze the performance of basic proximity primitives. In section 4, we consider the following fundamental proximity queries for a set of point sites in the plane: nearest neighbor search, k-nearest neighbors search, and circular range search. We show that the existing methods for efficiently answering such queries have degree either 6 (point location in explicit Voronoi diagram) or 4 (extremal-search method), and we present our new technique, based on implicit Voronoi diagrams, which achieves optimal degree 2. In sections 5–6, we extend our approach to nearest neighbor search queries in a set of 3D point sites and in a set of point and segment sites in the plane, respectively. Practical improvements are presented in section 7. Finally, further research directions are discussed in section 8.

2. Degree of geometric algorithms and problems. The numerical computations of a geometric algorithm are basically of two types: tests (predicates) and constructions. The two types of computations have clearly distinct roles. Tests are associated with branching decisions in the algorithm that determine the flow of control, whereas constructions are needed to produce the output data of the algorithm.

Approximations in the execution of constructions are acceptable, since approximate results are perfectly tolerable, provided that the error magnitude does not exceed the resolution required by the application. On the other hand, approximations in the execution of tests may produce an incorrect branching of the algorithm. Such event may have catastrophic consequences, giving rise to *structurally* incorrect results. The exact-computation paradigm therefore requires that tests be executed with total accuracy.

We shall therefore characterize geometric algorithms on the basis of the complexity of their test computations. Any such computation consists of evaluating the sign of an *algebraic expression* over the input variables, constructed using an adequate set of operators such as $\{+, -, \times, \div, \sqrt[2]{}, \ldots\}$. As we shall show below, the expressions under consideration are equivalent to multivariate polynomials.

Here we make the reasonable assumption that input data be *dimensionally consistent*, i.e., that if an entity with the physical dimension of a length is represented with b bits, then one with the dimension of an area be represented with $2b$ bits, and so on. Under the hypothesis of dimensional consistency (where point coordinates are b-bit entries), a polynomial expressing a test is homogeneous because all of its monomials must have the same physical dimension.

A *primitive variable* is an input variable of the algorithm and has conventional arithmetic degree 1. The arithmetic degree of a polynomial expression E is the common arithmetic degree of its monomials. The arithmetic degree of a monomial is the sum of the arithmetic degrees of its variables.

DEFINITION 1. *An algorithm has* degree d *if its test computations involve the evaluation of multivariate polynomials of arithmetic degree at most* d. *A problem* Π *has* degree d *if any algorithm that solves* Π *has degree at least* d.

Remark 1. Recently, Burnikel [9] has independently defined the notion of *precision of an algorithm*, which is equivalent to our notion of degree of an algorithm. Also, our definition of degree is related to that of *depth of derivation* proposed by Yap [58, 59]. Given a set of numbers, any number x of the set has depth 0. A number has depth at most d if it can be obtained by executing a rational operation on numbers with depth $d-1$ or it is the result of a root extraction from a degree-k polynomial whose coefficients have depth at most $d-k$. An algorithm has *depth* d if it performs only rational operations such that all the intermediate computed numbers have depth of derivation at most d with respect to the set of input numbers. Clearly, d is the least possible integer such that all the intermediate computed values have depth of derivation at most d. A problem has *depth* d if it can be solved by an algorithm with rational bounded depth d. Despite the relatedness of the notions of depth and degree, the latter seems more appropriate to our analysis, where we aim at minimizing the number of bits needed for computing an exact value, independently of its (possibly very high) depth.

Motivated by a standard feature of geometric algorithms, we also make the assumption that every multivariate polynomial of degree d used in tests depends upon a set of size s (a small constant) of primitive variables. Therefore, a multivariate polynomial has $O(s^d)$ distinct monomials with bounded integer coefficients, so that the maximum value of the multivariate polynomial is expressible with at most $db + d \log s$ bits. A consequence of Definition 1 and of the above assumption is the following fact, which justifies our use of the degree of an algorithm to characterize the precision required in test computations.

LEMMA 1. *If an algorithm has degree* d *and its input variables are* b*-bit integers, then all the test computations can be carried out with* $d(b + O(1))$ *bits.*

Typically the support of a geometric test is not naturally expressed by a multivariate polynomial but, rather, by a pair (E_1, E_2) of expressions involving the four arithmetic operations, powering, and the extraction of square roots, and the test consists of comparing the magnitudes of E_1 and E_2. Such expressions always have a physical dimension (a length, an area, a volume, etc.), so that if they have the form of ratios, the degree of the numerator exceeds that of the denominator.

Expressions such as E_1 and E_2 can be viewed as a binary tree, whose leaves represent input variables and whose internal nodes are of two types: binary nodes,

which are labeled with an operation from the set $\{+, -, \times, \div\}$, and unary nodes, which are labeled either with a power or with a square root extraction (notice that we restrict ourselves to this type of radical). If no radical appears in the trees of E_1 and E_2, then the test is trivially equivalent to the evaluation of the sign of a polynomial, since E_i is a rational function of the form $\frac{N_i}{D_i}$ ($i = 1, 2$, N_i, D_i are not both trivial polynomials and $D_i \neq 0$) and

$$E_1 \geq E_2 \Longleftrightarrow (-1)^{\sigma(D_1)+\sigma(D_2)}(N_1 D_2 - N_2 D_1) \geq 0,$$

where $\sigma(E) = 1$ if $E < 0$ and $\sigma(E) = 0$ if $E \geq 0$. (Note that the above predicate implies the inductive assumption that the signs of lower-degree expressions N_1,N_2, D_1, and D_2 are known.) Suppose now that at least one of the trees of E_1 and E_2 contains radicals. We prune the tree so that the pruned tree contains no radicals except at its leaves (notice that pruned subtrees may themselves contain radicals). Then N_i and D_i ($i = 1, 2$) can be viewed as polynomials whose variables are the radicals and whose coefficients are (polynomial) functions of nonradicals. Given a polynomial E in a set of radicals, for any radical R in this set, we can express E as $E = E''R + E'$ where neither E'' nor E' contains R. Then

$$E \geq 0 \Longleftrightarrow E''R \geq -E'.$$

The resulting expression $(E''^2 R^2 - E'^2)$ does not contain R. Therefore, by this device, referred to as *segregate and square*, we can eliminate one radical. This shows that by the four rational operations we can reduce the sign test to the computation of the signs of a collection of multivariate polynomials.

We now present a very simple, but useful, formalism that enables us to rapidly evaluate the degree of the multivariate polynomial which uniquely determines the sign of the original algebraic expression.

The terms involved in the formal manipulations are of two types: generic and specific. *Generic* terms have the form α^s (for a formal variable α and an integer s), representing an unspecified multivariate polynomial of degree s over primitive variables. *Specific* terms have the form ρ_j, for some integer index j, representing a specified expression involving the operators $\{+, -, \times, \div, \sqrt{\ }\}$. The terms are members of a free commutative semiring; i.e., addition and multiplication are associative and commutative, addition distributes over multiplication, and specific terms can be factored out. Besides these conventional algebraic rules, we have a set of *rewriting rules* of the form $A \rightarrow B$, meaning that the sign of A is *unambiguously determined* by the sign of B and by the signs of terms in A, which are inductively assumed to be known. This induction is either on the degree of the terms or, in case of addition of (same degree) terms, on the number of the latter.

We have seven rules, whose correctness can be proved with elementary algebra. Rule 1 performs genericization, i.e., a specific term ρ_j, which is known to be a polynomial of degree s over primitive variables, can be rewritten as α^s. Rules 2–4 involve generic terms, which reflect the fact that the only relevant feature of a polynomial is its degree. Finally, rules 5–7 concern specific terms. The role of specific terms is that we wish to keep track of their structure (that is, their definition) in order to exploit it when computing least common multiples or multiplying radicals together. Again, the R.H.S. of a rule gives the highest degree of the polynomials whose signs unambiguously determine the sign of the L.H.S. Recall that the stated hypothesis of nonnegative dimensionality implies that the degree of a numerator is never smaller

than that of its denominator. The rules are

$$\begin{array}{lccc}
(1) & \rho_j & \longrightarrow & \alpha^s \\
(2) & \alpha^s\alpha^r & \longrightarrow & \alpha^{s+r} \\
(3) & \alpha^s + \alpha^s & \longrightarrow & \alpha^s \\
(4) & -\alpha^s & \longrightarrow & \alpha^s \\
(5) & \frac{\rho_j}{\rho_i} \pm \frac{\rho_h}{\rho_i} & \longrightarrow & \rho_j \pm \rho_h \\
(6) & \frac{\rho_j}{\rho_i} \pm \frac{\rho_h}{\rho_k} & \longrightarrow & \rho_j\rho_k \pm \rho_i\rho_h \\
(7) & \rho_i \pm \rho_j & \longrightarrow & \rho_i^2 - \rho_j^2.
\end{array}$$

A discussion on how to compute the sign of an algebraic expression of the type considered by rule (7) can also be found in [57].

The preceding discussion establishes the following theorem.

THEOREM 1. *Rules* (1)–(7) *are adequate to evaluate the degree of multivariate polynomials whose sign, collectively, unambiguously determines the sign of an arbitrary algebraic expression involving square roots.*

While the above rules represent an adequate formalism for obtaining an upper bound to the degree of an algorithm, more subtle is the corresponding lower-bound question. In other words, given a predicate $\mathcal{P}$ that is essential to the solution of a given problem, what is the inherent degree of $\mathcal{P}$? Suppose that predicate $\mathcal{P}$ is expressed by a polynomial P of degree d, and we must decide whether the value of P is positive, negative, or zero. Can we answer this question by computing a discrete (ternary) function f of analogous evaluations of irreducible polynomials $P_1, \ldots, P_k$ of maximum degree smaller than d? Clearly, f changes value only when some P_j changes sign (exactly, when the value of P_j passes by 0). Thus, a 0 of P corresponds to a 0 of some P_j. Moreover, as the arguments of P_j vary while P_j remains 0, so does f and hence P. Therefore, P vanishes at all points for which P_j vanishes and, for a well-known theorem of polynomial algebra (see, e.g., [8, pp. 212–216]), we conclude that P_j is a factor of P. This is summarized as follows.

THEOREM 2. *The degree of the problem of evaluating a predicate expressed by a polynomial P is the maximum arithmetic degree of the factors of P that change sign over their domain.*

3. Basic proximity queries. In this section we use the formalism introduced above to analyze the degree of some geometric tests that answer basic proximity queries. We end the section by establishing a lower bound on the degree of the nearest neighbor search problem. In the proofs, we assume that a line r is represented by the coefficients of its equation. However, the results still hold if line r is represented by two of its points.

We start with the point-to-lines distance test; i.e., given two lines r_1 and r_2 on the plane and a query point q, determine whether q is closer to r_1 than to r_2.

LEMMA 2. *The* point-to-lines distance test *can be solved with degree* 6.

Proof. Let the equation of r_i be $a_ix + b_iy + c_i = 0$ $(i = 1, 2)$ and let $q \equiv (x_q, y_q)$. Then the test is to study the sign of $\frac{|a_1x_q+b_1y_q+c_1|}{\sqrt{a_1^2+b_1^2}} - \frac{|a_2x_q+b_2y_q+c_2|}{\sqrt{a_2^2+b_2^2}}$. By using the proposed notation, and with obvious meaning for ρ_1 and ρ_2, this test becomes (each arrow being superscripted with the rules used)

$$\frac{\alpha^2}{\rho_1} - \frac{\alpha^2}{\rho_2} \quad \longrightarrow^{(6)} \alpha^2\rho_2 - \alpha^2\rho_1 \quad \longrightarrow^{(7)} \alpha^4\rho_2^2 - \alpha^4\rho_1^2 \quad \longrightarrow^{(1)}$$
$$\alpha^4\alpha^2 - \alpha^4\alpha^2 \quad \longrightarrow^{(4,3)} \alpha^6. \qquad \square$$

The following lemmas describe the degree of other proximity primitives that will be useful in the rest of the paper. We omit the proofs of such lemmas, since they

are either straightforward or have been already proved in [9]. However, it is worth mentioning that the proofs in [9] can be substantially simplified by using the proposed notation.

Let p be a point and r a line in the plane. The point-to-point-line distance test determines whether a query point q is closer to p or to r.

LEMMA 3. *The* point-to-point-line distance test *can be solved with degree* 4.

Let p_1 and p_2 be two distinct points of the plane and let q be a query point. The point-to-points distance test determines whether q is closer to p_1 or to p_2.

LEMMA 4. *The* point-to-points distance test *can be solved with degree* 2.

The above lemma can be easily extended to any space of dimension d.

Another fundamental proximity primitive is the incircle test, that is, testing whether the circle determined by three distinct sites (points and/or segments) of the plane contains a given query site. The incircle test is a basic operation for many algorithms that construct the Voronoi diagram of the sites (see, e.g., [37, 41, 33, 3]). The degree of the incircle test has been extensively studied by Burnikel [9] and by Burnikel, Mehlhorn, and Schirra [11]. Following the notation of Burnikel [9], an incircle test is conveniently expressed as a quadruple $(a_1, a_2, a_3; a_4)$, where each $a_i \in \{p, l\}$ $(i = 1, \ldots, 4)$ is either a point or a line on the plane (a segment is seen by Burnikel as given by the pair of its endpoints and by the underlying line) and we test whether a_4 intersects the circle determined by a_1,a_2, and a_3.

The following lemma is proved observing that the incircle test $(p_1, p_2, p_3; p_4)$ can be answered by determining the sign of a 4×4 determinant that is an arithmetic degree-4 multivariate polynomial.

LEMMA 5 (see [9]). *The* incircle test $(p_1, p_2, p_3; p_4)$ *can be solved with degree* 4.

Lemma 5 can be easily extended to any dimension $d > 2$. We describe such a test as $(p_1, \ldots, p_{d+1}; p_{d+2})$, where points $p_1, \ldots, p_{d+1}$ determine a d-dimensional sphere and p_{d+2} is the query point.

LEMMA 6. *The* insphere test $(p_1, \ldots, p_{d+1}; p_{d+2})$ *in any fixed dimension* $d \geq 2$ *can be solved with degree* $d + 2$.

For the construction of the Voronoi diagram of a set of points and segments in the plane Burnikel shows that the most demanding test in terms of degree is the incircle test $(l_1, l_2, l_3; l_4)$ [9].

LEMMA 7 (see [9]). *The* incircle test $(l_1, l_2, l_3; l_4)$ *can be solved with degree* 40.

While the above lemmas provide an upper bound on the degree of a proximity problem, the next theorem gives a lower bound.

THEOREM 3. *The nearest neighbor search problem for a point set has degree* 2 *in any fixed dimension* $d \geq 2$.

Proof. We show the proof for the case $d = 2$. The proof for any other values of d is analogous. Let $p_1 \equiv (x_1, y_1)$, $p_2 \equiv (x_2, y_2)$, and $q \equiv (x_q, y_q)$ be three points in the plane. In order to determine which of p_1 and p_2 is the point nearest to q, a point-to-points distance test must be performed.

This is equivalent to the evaluation of the sign of the difference $d(p_1, q) - d(p_2, q)$, which, in turn, is equivalent to the evaluation of the sign of the polynomial $d^2(p_1, q) - d^2(p_2, q)$. This shows that this computation has degree at most 2. On the basis of Theorem 2, for the degree to be less than 2, polynomial $d^2(p_1, q) - d^2(p_2, q)$ should be factorable as the product of two degree-1 polynomials. We show below that this is not possible.

Suppose, for a contradiction, that there exist constants $a', a'', b', b'', c', c'', d', d'', e, e'', f', f''$ such that

$$d^2(p_1, q) - d^2(p_2, q) = x_1^2 + y_1^2 - x_2^2 - y_2^2 - 2x_1x_q + 2x_2x_q - 2y_1y_q + 2y_2y_q$$
$$= (a'x_1 + b'y_1 + c'x_2 + d'y_2 + e'x_q + f'y_q) \cdot (a''x_1 + b''y_1 + c''x_2 + d''y_2 + e''x_q + f''y_q).$$

The above equality implies $e'e'' = 0$, since there cannot be a term $e'e''x_q^2$. However, e' and e'' are not simultaneously 0, because there are nonzero terms having x_q as a factor. Assume w.l.o.g. that $e'' \neq 0$. Observe that $d'e'' = 0$ because there is no term $d'e''y_2y_q$; this implies $d' = 0$. However, we must also have $d'd'' = -1$ because of the term $-y_2^2$, a contradiction. □

Observe that an optimal degree algorithm for the nearest neighbor search problem in a planar point set can be easily obtained with the brute-force approach, where one computes all the distances between the query point and all other points and reports the point at minimum distance. However, such algorithm is both computationally inefficient (it requires quadratic time) and does not support repetitive-mode queries. In section 4 we present an optimal degree algorithm, complying with the exact-computation paradigm, whose query time and space are optimal.

4. Proximity queries for point sites in the plane. In this section, under our standard assumption that all input parameters — such as coordinates of sites and query points — are represented by b-bit integers, we consider the following proximity queries on a set S of point sites in the plane:

nearest neighbor search: given query point q, find a site of S whose Euclidean distance from q is less than or equal to that of any other site;

k-nearest neighbors search: given query point q, find k sites of S whose Euclidean distances from q are less than or equal to that of any other site;

circular range search: given query points q and r, find the sites of S that are inside the circle with center q passing through r.

It is well known that such queries are efficiently solved by performing point location in the Voronoi diagram (possibly of higher order) $V(S)$ of the sites [51]. For nearest neighbor search, the alternative extremal-search method [26] also exists.

We begin by examining in section 4.1 the geometric test primitives used by the theoretically optimal and practically efficient point location methods. We identify three fundamental geometric test primitives for accessing the geometry of a planar map, and we introduce the concepts of "native" and "ordinary" point location methods. In section 4.2, we show that the "conventional" approach of accessing the explicitly computed Voronoi diagram $V(S)$ of the sites causes point location queries, and hence proximity queries, to have degree at least 6. We also analyze the extremal-search method and show that it has degree 4. In sections 4.3–4.4, we describe our new implicit representation of Voronoi diagrams for point sites in the plane, which allows us to perform proximity queries with optimal degree 2.

4.1. Test primitives and methods for planar point location. The *chain method* [44], the *bridged-chain method* [25], the *trapezoid method* [50], the *subdivision refinement method* [42], and the *persistent search tree method* [53] are popular deterministic techniques for point location in a planar map that combine theoretical efficiency with good performance in practice (see, e.g., [24, 51]). Namely, denoting with n the size of the map, all the above point location methods require $O(n \log n)$ preprocessing time. The query time is $O(\log^2 n)$ for the chain method and $O(\log n)$ for the other methods. The space used is $O(n \log n)$ for the trapezoid method and $O(n)$ for the other methods. For monotone maps, the preprocessing time is $O(n)$ for the chain method and the bridged-chain method, and $O(n \log n)$ for the other methods. The *randomized-incremental method* [35] also exists. Such a method is specialized for point location in Voronoi diagrams, uses expected space $O(n)$, and has expected query time $O(\log^2 n)$.

By a careful examination of the query algorithms used in the point location methods presented in the literature, it is possible to clearly separate the primitive opera-

tions that access the geometry of the map from those that access only the topology. We say that a point location method is *native* for a class of maps if it performs point location queries in a map M of the class by accessing the geometry of M exclusively through the following three geometric test primitives that discriminate the query point with respect to the vertices and edges of M:

above-below(q, v) determine whether query point q is vertically above or below vertex v.

left-right(q, v) determine whether query point q is horizontally to the left or to the right of vertex v.

left-right(q, e) determine whether query point q is to the left or to the right of edge e; this operation assumes that edge e is not horizontal and its vertical span includes q.

Test primitive left-right(q, v) is typically used only in degenerate cases (e.g., in the presence of horizontal edges).

Some point location methods work on modified versions of the original subdivision by means of auxiliary geometric objects introduced in the preprocessing (e.g., triangulation or regularization edges). We say that a point location method is *ordinary* for a class of maps if it performs point location queries in a map M of the class by accessing the geometry of M through the three geometric test primitives described above for the native methods and through left-right(q, e) tests such that e is a fictitious edge connecting two vertices of M.

Now, we analyze the chain method [44] for point location in a monotone map M. A binary tree represents a balanced recursive decomposition of map M by means of vertically monotone polygonal chains covering the edges of M, called *separators*. A point location query consists of traversing a root-to-leaf path in this tree, where at each node we determine whether the query point q is to the left or to right of the separator associated with the node. The discrimination of point q with respect to a separator σ is performed in two steps:

1. we find the edge e of σ whose vertical span includes point q by means of binary search on the y coordinates of the vertices of σ, which consists of performing a sequence of a logarithmic number of above-below(q, v) tests;
2. we discriminate q with respect to σ by performing test left-right(q, e).

In the special case that separator σ has horizontal edges, the discrimination of point q with respect to σ uses also test primitive left-right(q, v). Hence, the chain method is native for monotone maps. For a map M that is not monotone, fictitious "regularization" edges are added to M and the point location in M is reduced to point location in the resulting refinement M' of M. Hence, the chain method is ordinary for general maps.

In the bridged-chain method [25], the technique of *fractional cascading* [17, 18] is applied to the sets of y-coordinates of the separators. Fractional cascading establishes "bridges" between the separator of a node and the separators of its children such that there are $O(1)$ vertices between any two consecutive bridges. Hence, except for the separator of the root, step 1 can be executed with $O(1)$ above-below(q, v) tests for the vertices between two consecutive bridges. The bridged-chain method is ordinary for general maps and native for monotone maps.

A similar analysis shows that all efficient point-location methods described in the literature are ordinary for general maps. More specifically, we have the following lemma.

LEMMA 8. *The trapezoid method and the persistent search tree method are native for general maps. The chain method and the bridged-chain method are ordinary for general maps and native for monotone maps. The subdivision refinement method*

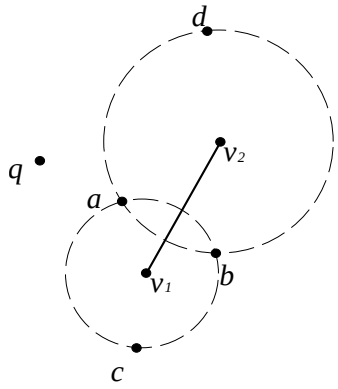


FIG. 1. *Illustration for Lemma* 9.

is ordinary for general maps. The randomized-incremental method is ordinary for Voronoi diagrams.

Hence, all the known planar point location methods described in the literature are ordinary for Voronoi diagrams.

4.2. Explicit Voronoi diagrams. Let S be a set of n point sites in the plane, where each site is a primitive point with b-bit integer coordinates. The Voronoi diagram $V(S)$ of S is said to be *explicit* if the coordinates of the vertices of $V(S)$ are computed and stored with exact arithmetic, i.e., as rational numbers (pairs of integers).

LEMMA 9. *The* left-right(q, e) *test primitive in an explicit Voronoi diagram of point sites in the plane has degree* 6.

Proof. Let $e \equiv (v_1, v_2)$ be a Voronoi edge such that $v_1 \equiv (x_1, y_1)$ is equidistant from three sites $a \equiv (x_a, y_a)$, $b \equiv (x_b, y_b)$, $c \equiv (x_c, y_c)$ and $v_2 \equiv (x_2, y_2)$ is equidistant from three sites $b \equiv (x_b, y_b)$, $c \equiv (x_c, y_c)$, and $d \equiv (x_d, y_d)$. See Figure 1. In an explicit Voronoi diagram, test primitive left-right(q, e) that determines whether query point $q \equiv (x_q, y_q)$ is to the left or to the right of edge $e \equiv (v_1, v_2)$ is equivalent to evaluating the sign of the following determinant:

$$\Delta = \begin{vmatrix} x_q & y_q & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = \begin{vmatrix} x_q & y_q & 1 \\ \frac{X_1}{2W_1} & \frac{Y_1}{2W_1} & 1 \\ \frac{X_2}{2W_2} & \frac{Y_2}{2W_2} & 1 \end{vmatrix} = \frac{1}{4W_1W_2} \begin{vmatrix} x_q & y_q & 1 \\ X_1 & Y_1 & 2W_1 \\ X_2 & Y_2 & 2W_2 \end{vmatrix} = \frac{\Delta'}{4W_1W_2},$$

where

$$X_1 = \begin{vmatrix} x_a^2 + y_a^2 & y_a & 1 \\ x_b^2 + y_b^2 & y_b & 1 \\ x_c^2 + y_c^2 & y_c & 1 \end{vmatrix}, \quad Y_1 = \begin{vmatrix} x_a & x_a^2 + y_a^2 & 1 \\ x_b & x_b^2 + y_b^2 & 1 \\ x_c & x_c^2 + y_c^2 & 1 \end{vmatrix}, \quad W_1 = \begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix}$$

and X_2, Y_2, and W_2 have similar expressions obtained replacing in the above determinants x_c with x_d and y_c with y_d. Evaluating the sign of Δ is equivalent to evaluating the signs of W_1, W_2 and of Δ'.

By using the notation introduced in section 2, we can rewrite X_i and Y_i as α^3, and W_i as α^2 $(i = 1, 2)$. Hence, Δ' is a degree-6 multivariate polynomial since it can be rewritten as

$$\alpha(\alpha^3\alpha^2 - \alpha^3\alpha^2) - \alpha(\alpha^3\alpha^2 - \alpha^3\alpha^2) + \alpha^3\alpha^3 - \alpha^3\alpha^3 \quad \longrightarrow^{(2,3,4)} \quad \alpha^6 + \alpha^6 \quad \longrightarrow^{(3)} \quad \alpha^6.$$

Although the explicit representation approach leads to Lemma 9, it should be noted that determinant Δ is a reducible polynomial,[1] one factor being the (always positive) incircle test polynomial of degree 4 for the four sites. □

An algorithm for proximity queries on a set S of point sites in the plane is said to be *conventional* if it computes the explicit Voronoi diagram $V(S)$ of S and then performs point location queries on $V(S)$ with an ordinary method. Note that the class of conventional proximity query algorithms includes all the efficient algorithms presented in the literature. A conventional proximity query algorithm needs to perform test primitive left-right(q, e). Thus, by Lemma 9 we have the following theorem.

THEOREM 4. *Conventional algorithms for the following proximity query problems on a set of point sites in the plane have degree at least* 6*:*

- *nearest neighbor query,*
- *k-nearest neighbor query,*
- *circular range query.*

We observe that a degree-6 algorithm implies that a k-bit arithmetic unit can handle with native precision queries for points in a grid of size at most $2^{k/6} \times 2^{k/6}$. For example, if $k = 32$, the points that can be treated with single-precision arithmetic belong to a grid of size at most 64×64.

The *extremal-search method* [26], also designed for proximity queries, reduces the nearest neighbor search problem for a set S of 2D point sites to the following extremal-search problem. Let $\mathcal{P}$ be the paraboloid with equation $z = x^2 + y^2$, and let S' be the set of 3D points obtained by lifting S to $\mathcal{P}$. Given a query point q in the plane, let $\vec{r}$ be the unit vector orthogonal to the plane tangent to $\mathcal{P}$ at the lifted query point $q' \equiv (x_q, y_q, x_q^2 + y_q^2)$. The extremal-search problem for S' and query vector $\vec{r}$ consists of determining the first site s' of S' hit by a plane orthogonal to $\vec{r}$ translating from infinity toward S'. Projecting s' down onto the xy-plane gives the nearest neighbor s of q in S.

The extremal-search method makes use of 3D geometric primitives that guide the search through a data structure embodying the Dobkin–Kirkpatrick hierarchical representation [22] of the convex hull of S'. Such 3D geometric primitives in turn can be reduced to the following 2D geometric primitives:

- point-to-points distance test for q and a site of S, which has degree 2;
- the identification of suitably defined "extremal edges" of the Delaunay triangulation of a subset of S with respect to q.

The second primitive evaluates the sign of determinants of the type

$$\Delta = \begin{vmatrix} x_a & y_a & x_a^2 + y_a^2 \\ x_b & y_b & x_b^2 + y_b^2 \\ x_q & y_q & x_q^2 + y_q^2 \end{vmatrix},$$

where $a \equiv (x_a, y_a)$ and $b \equiv (x_b, y_b)$ are sites of S. By using the methodology introduced in section 2, we can show that Δ is a degree-4 multivariate polynomial. Thus, we have Theorem 5.

THEOREM 5. *The extremal-search method for the nearest neighbor query problem on a set of point sites in the plane has degree at least* 4.

4.3. Implicit Voronoi diagrams. Let S be a set of n point sites in the plane, and recall our assumption that each site or query point is a primitive point with b-bit integer coordinates. We say that a number s is a *semi-integer* if it is a rational number of the type $s = m/2$ for some integer m. The *implicit Voronoi diagram* $V^*(S)$ of S is a representation of the Voronoi diagram $V(S)$ of S that consists of a topological

[1] K. Mehlhorn suggested that Δ was likely to be reducible.

component and of a geometric component. The topological component of $V^*(S)$ is the planar embedding of $V(S)$, represented by a suitable data structure (e.g., doubly connected edge lists [51] or the data structure of [37]). The geometric component of $V^*(S)$ stores the following geometric information for each vertex and edge of the embedding:

- For each vertex v of $V(S)$, $V^*(S)$ stores semi-integers $x^*(v)$ and $y^*(v)$ that approximate the x- and y-coordinates $y(v)$ of v. We provide the definition of $y^*(v)$ below. The definition of $x^*(v)$ is analogous.

$$y^*(v) = \begin{cases} y(v), & 0 \le y(v) \le 2^b - 1,\ y(v) \text{ integer}, \\ \lfloor y(v) \rfloor + \frac{1}{2}, & 0 \le y(v) \le 2^b - 1,\ y(v) \text{ not integer}, \\ 2^b - \frac{1}{2}, & y(v) > 2^b - 1, \\ 0, & y(v) < 0. \end{cases}$$

  Note that semi-integers $x^*(v)$ and $y^*(v)$ can be stored with $(b+1)$-bits.
- For each nonhorizontal edge e of $V(S)$, $V^*(S)$ stores the pair of sites $\ell(e)$ and $r(e)$ such that e is a portion of the perpendicular bisector of $\ell(e)$ and $r(e)$, and $\ell(e)$ is to the left of $r(e)$.

Equipped with the above information, the three test primitives for point location can be performed in the implicit Voronoi diagram $V^*(S)$ as follows:

above-below(q, v) compare the y-coordinate of q with $y^*(v)$;

left-right(q, v) compare the x-coordinate of q with $x^*(v)$;

left-right(q, e) compare the Euclidean distances of point q from sites $\ell(e)$ and $r(e)$.

Since the query point q is by assumption a primitive point whose coordinates are b-bit integers, we have that $y(q) \le y(v)$ if and only if $y(q) \le y^*(v)$, where testing the latter inequality has degree 1. Similar considerations apply to testing $x(q) \le x(v)$. This proves the correctness of our implementation of above-below(q, v) and left-right(q, v).

The correctness of the above implementation of test left-right(q, e) follows directly from the definition of Voronoi edges. Thus, in an implicit Voronoi diagram, test left-right(q, e) can be implemented with a point-to-points distance test that has degree 2 (Lemma 4).

Hence, we obtain the following lemmas.

LEMMA 10. *Test primitives* above-below(q, v) *and* left-right(q, v) *in an implicit Voronoi diagram of point sites in the plane can be performed in* $O(1)$ *time and with degree* 1.

LEMMA 11. *Test primitive* left-right(q, e) *in an implicit Voronoi diagram of point sites in the plane can be performed in* $O(1)$ *time and with degree* 2.

In order to execute a native point location algorithm in an implicit Voronoi diagram, we only need to redefine the implementation of the three test primitives. By having encapsulated the geometry in the test primitives, no further modifications are needed. Hence, by Lemmas 10–11 we obtain Lemma 12.

LEMMA 12. *For any native method on a class of maps that includes Voronoi diagrams, a point location query in an implicit Voronoi diagram has optimal degree* 2 *and has the same asymptotic time complexity as a point location query in the corresponding explicit Voronoi diagram.*

In order to compute the implicit Voronoi diagram $V^*(S)$, we begin by constructing the Delaunay triangulation of S, denoted $DT(S)$, by means of the $O(n \log n)$-time algorithm of [37], which has degree 4 because its most expensive operation in terms of the degree is the incircle test (see Lemma 5). The topological structure of $V(S)$ and the sites $\ell(e)$ and $r(e)$ for each edge e of $V(S)$ are immediately derived from $DT(S)$

by duality. Next, we compute the approximations $x^*(v)$ and $y^*(v)$ for each vertex v of $V(S)$ by means of integer division. For effective procedures that perform the integer division, see, e.g., *LEDA* [46]. Let a, b, and c be the three sites of S that define vertex v. Adopting the same notation as in the proof of Lemma 9, the y-coordinate $y(v)$ of v is given by the ratio $y(v) = \frac{Y_1}{2W_1}$, where Y_1 is a polynomial of degree 3 and W_1 is a polynomial of degree 2, and similarly for $x(v)$. Hence, the computation of $x^*(v)$ and $y^*(v)$ involves an integer represented by at most $3(b + O(1))$ bits. We summarize the above analysis as follows.

LEMMA 13. *The implicit Voronoi diagram of n point sites in the plane can be computed in $O(n \log n)$ time, $O(n)$ space, and with degree* 4.

THEOREM 6. *Let S be a set of n point sites in the plane. There exists an $O(n)$-space data structure for S, based on the implicit Voronoi diagram $V^*(S)$, that can be computed in $O(n \log n)$ time with degree* 5*, and supports nearest neighbor queries in $O(\log n)$ time with optimal degree* 2.

Proof. We perform point location in the implicit Voronoi $V^*(S)$ diagram of S using a native method for monotone maps with optimal space and query time such as the bridged-chain method or the persistent search tree method. The space requirement and the query degree and time follow from the performance of these methods and from Lemma 12.

Regarding the preprocessing time, by Lemma 13, the construction of the implicit Voronoi $V^*(S)$ takes $O(n \log n)$ time with degree 4. In order to construct the point location data structure, we also need an additional test primitive that consists of comparing the y-coordinates of two Voronoi vertices. For example, this primitive is used to establish bridges in the bridged-chain method (see section 4.1) and to sort the vertices by y-coordinate in the persistent location method. By using the same notation as in Lemma 9, comparing the y-coordinates of the Voronoi vertices is equivalent to evaluating the sign of multivariate polynomials of the form $\frac{Y_i}{2W_i} - \frac{Y_j}{2W_j}$, where $\frac{Y_i}{2W_i}$ and $\frac{Y_j}{2W_j}$ represent the y-coordinates of two different Voronoi vertices. Such multivariate polynomials have degree 5, since they can be rewritten as

$$\frac{\rho_i}{\rho_j} - \frac{\rho_h}{\rho_k} \quad \longrightarrow^{(6)} \quad \rho_i\rho_k - \rho_h\rho_j \quad \longrightarrow^{(1)} \quad \alpha^3\alpha^2 - \alpha^3\alpha^2 \quad \longrightarrow^{(2,3,4)} \quad \alpha^5. \qquad \square$$

Remark 2. It must be pointed that for the problem under consideration similar results could be obtained by carrying out tests with limited accuracy, and therefore risking to mistakenly select a Voronoi site adjacent to the correct one in critical situations (when the query point is very close to the separating edge): such indeterminacy could be remedied by an additional test comparing the distances of the query point from the two competing sites. Although effective, such ad hoc solution would not fit the exact-computation paradigm, whereas our method fully complies with it. $\square$

4.4. Implicit higher-order Voronoi diagrams. In this section, we introduce implicit higher-order Voronoi diagrams for point sites in the plane, and we extend the results of section 4.3 to k-nearest neighbors and circular range search queries.

The definition of the *implicit order-k Voronoi diagram* $V_k^*(S)$ of set S of point sites in the plane is analogous to that given in section 4.3 for Voronoi diagrams. A vertex v of $V_k(S)$ is represented by its approximate coordinates $x^*(v)$ and $y^*(v)$, and a nonhorizontal edge e of $V_k(S)$ stores the pair of sites $\ell(e)$ and $r(e)$ such that e is a portion of the perpendicular bisector of $\ell(e)$ and $r(e)$, and $\ell(e)$ is to the left of $r(e)$.

Lemmas 10–11 immediately hold also for $V_k(S)$, and we obtain Lemma 14.

LEMMA 14. *For any native method for monotone maps, a point-location query in an implicit order-k Voronoi diagram has optimal degree* 2 *and has the same asymptotic time complexity as a point location query in an explicit order-k Voronoi diagram.*

The order-k Voronoi diagram $V_k(S)$ for a set S of n point sites has $O(k(n-k))$ vertices, edges, and faces and can be obtained from the order $k-1$ implicit Voronoi diagram $V_{k-1}(S)$ by intersecting each face of $V_{k-1}(S)$ with the (order-1) Voronoi diagram of a suitable subset of the vertices of S [43]. As shown in [43, 16], $V_k(S)$ can be computed in $O(k(n-k)\sqrt{n}\log n)$ time. Since the construction is based on iteratively computing Voronoi diagrams by using the incircle test, which is the most expensive operation in terms of degree, the overall degree of the preprocessing is 4 (Lemma 5). Hence, we obtain Lemma 15.

LEMMA 15. *The implicit order-k Voronoi diagram of n point sites in the plane can be computed in $O(k(n-k)\sqrt{n}\log n)$ time, $O(k(n-k))$ space, and with degree* 4.

Point location in the order-k Voronoi diagram solves k-nearest neighbors queries. Hence, by Theorem 3 and Lemmas 14–15, we obtain Theorem 7.

THEOREM 7. *Let S be a set of n point sites in the plane and k an integer with $1 \le k \le n-1$. There exists an $O(k(n-k))$-space data structure for S, based on the implicit order-k Voronoi diagram $V_k^*(S)$, that can be computed in $O(k(n-k)\sqrt{n}\log n)$ time with degree* 5 *and supports k-nearest neighbors queries in $O(\log n + k)$ time with optimal degree* 2.

Circular range search queries in a set S of n point sites can be reduced to a sequence of 2^i-nearest neighbors queries in $V_{2^i}(S)$, $i = 0, \ldots, \log n$ [7]. This approach yields a data structure with $O(n^3)$ space and preprocessing time, and $O(\log n \log\log n + k)$ query time, where k is the size of the output. Hence, with analogous reasoning as above, we obtain the following theorem.

THEOREM 8. *Let S be a set of n point sites in the plane. There exists an $O(n^3)$-space data structure for S, based on implicit order-k Voronoi diagrams, that can be computed in $O(n^3)$ time with degree* 5 *and supports circular range search queries in $O(\log n \log\log n + k)$ time with optimal degree* 2.

The space and preprocessing time of Theorems 7–8 and the query time of Theorem 8 can be improved while preserving the same degree bounds by more complicated procedures along the lines suggested in [1, 2, 15].

5. Proximity queries for point sites in 3D space. In this section, we consider the following proximity query on a set S of point sites in 3D space:

nearest neighbor search: given query point q, find a site of S whose Euclidean distance from q is less than or equal to that of any other site.

We recall our assumption that the sites and query points are primitive points represented by b-bit integers.

As for the 2D case, such a query is efficiently answered by performing point location in the 3D Voronoi diagram of S. Test primitives and methods for spatial point location are described in section 5.1. Section 5.2 shows that "conventional" algorithms require degree 8. A degree-3 algorithm based on "implicit" 3D Voronoi diagrams is then given in section 5.3.

5.1. Test primitives and methods for spatial point location. There are only two known efficient spatial point location methods for cell-complexes that are applicable to 3D Voronoi diagrams: the *separating surfaces method* [14, 56], which extends the chain method [44], and the *persistent planar location method* [52], which extends the persistent search tree method [53]. Let N be the number of facets of a cell-complex C. The query time is $O(\log^2 N)$ for both methods. The space used is $O(N)$ for the separating surfaces method and $O(N \log^2 N)$ for the persistent planar location method. Both methods are restricted to convex cell-complexes. The separating surfaces method is further restricted to acyclic convex cell-complexes, where the dominance relation among cells in the z-direction is acyclic.

As in section 4.1, we can separate the primitive operations that access the geometry of the cell-complex from those that access only the topology. We say that a point location method is *native* for a class of 3D cell-complexes if it performs point locations queries in a cell-complex C of the class by accessing the geometry of C exclusively through the following three geometric test primitives that discriminate the query point with respect to the vertices and edges of C:

above-below(q, v) compares the z-coordinate of the query point q with the z-coordinate of vertex v.

left-right(q, v) compares the x-coordinate of the query point q with the x-coordinate of vertex v.

front-rear(q, v) compares the y-coordinate of the query point q with the y-coordinate of vertex v.

left-right(q_{xy}, e_{xy}) compares the xy-projection q_{xy} of the query point q with the xy-projection of edge e_{xy}. This operation assumes that e_{xy} is not parallel to the x-axis and its y-span includes q_{xy}.

above-below(q, f) determines whether query point q is above or below a facet f; this operation assumes that facet f is not parallel to the z-axis and that the xy-projection of f contains the xy-projection of q.

Test primitives above-below(q, v) and left-right(q, v) are used only in degenerate cases (e.g., in the presence of facets parallel to the z-axis and in cases where e_{xy} is horizontal).

Now, we analyze the separating surfaces method for spatial point location [14, 56] in acyclic cell-complexes. *Separating surfaces* are the 3D analogue of separators of monotone maps. Their existence is guaranteed by the acyclicity of the cell-complex. Thus, a point location query consists of traversing a root-to-leaf path in the *separating surface tree*, where at each node we determine whether the query point q is to above or below the separating surface associated with the node. The discrimination of point q with respect to a separating σ is performed in two steps:

1. By means of a planar point location query for the xy-projection q_{xy} of q in the xy projection of σ, we find the facet f of σ whose xy projection contains q_{xy}. If an ordinary point location method is used, this step uses primitives front-rear(q, v), left-right(q, v), and left-right(q_{xy}, e_{xy}).
2. We discriminate q with respect to σ by performing test above-below(q, f).

In the special cases that cell-complex C has facets parallel to the z-axis, the discrimination of point q with respect to σ uses also test primitives above-below(q, v). Thus, the separating surfaces method is native for acyclic convex cell-complexes.

A similar analysis shows that also the persistent planar location method is native for convex cell-complexes. More specifically, we have Lemma 16.

LEMMA 16. *The separating surfaces method is native for acyclic convex cell-complexes. The persistent planar location method is native for convex cell-complexes.*

Hence, all the known spatial point location methods described in the literature are native for 3D Voronoi diagrams.

5.2. Explicit Voronoi diagrams. Let S be a set of n point sites in 3D, where each site is a primitive point with b-bit integer coordinates. The 3D Voronoi diagram $V(S)$ of S is said to be *explicit* if the coordinates of the vertices of $V(S)$ are computed and stored with exact arithmetic, i.e., as rational numbers (pairs of integers).

LEMMA 17. *The* left-right(q_{xy}, e_{xy}) *test primitive in an explicit Voronoi diagram of point sites in* 3D *space has degree* 8.

Proof. The reasoning is the same as in the proof of Lemma 9. Let $e_{x,y} \equiv (v_1, v_2)$, where v_1 and v_2 are the xy-projections of two adjacent vertices u and v of $V(S)$; let u be equidistant from the four primitive sites $a \equiv (x_a, y_a)$, $b \equiv (x_b, y_b)$, $c \equiv (x_c, y_c)$, and

$d \equiv (x_d, y_d)$, and v from $a \equiv (x_a, y_a)$, $b \equiv (x_b, y_b)$, $c \equiv (x_c, y_c)$, and $h \equiv (x_h, y_h)$. In an explicit Voronoi diagram, test primitive $\mathsf{left\text{-}right}(q_{xy}, e_{xy})$ that determines whether query point $q \equiv (x_q, y_q)$ is to the left or to the right of edge $e \equiv (v_1, v_2)$ is equivalent to evaluating the sign of the following determinant:

$$\Delta = \begin{vmatrix} x_q & y_q & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = \begin{vmatrix} x_q & y_q & 1 \\ \frac{X_1}{2W_1} & \frac{Y_1}{2W_1} & 1 \\ \frac{X_2}{2W_2} & \frac{Y_2}{2W_2} & 1 \end{vmatrix} = \frac{1}{4W_1W_2} \begin{vmatrix} x_q & y_q & 1 \\ X_1 & Y_1 & 2W_1 \\ X_2 & Y_2 & 2W_2 \end{vmatrix} = \frac{\Delta'}{4W_1W_2},$$

where

$$X_1 = \begin{vmatrix} x_a^2+y_a^2+z_a^2 & y_a & z_a & 1 \\ x_b^2+y_b^2+z_b^2 & y_b & z_b & 1 \\ x_c^2+y_c^2+z_c^2 & y_c & z_c & 1 \\ x_d^2+y_d^2+z_d^2 & y_d & z_d & 1 \end{vmatrix}, \quad Y_1 = \begin{vmatrix} x_a & x_a^2+y_a^2+z_a^2 & z_a & 1 \\ x_b & x_b^2+y_b^2+z_b^2 & z_b & 1 \\ x_c & x_c^2+y_c^2+z_c^2 & z_c & 1 \\ x_d & x_d^2+y_d^2+z_d^2 & z_d & 1 \end{vmatrix},$$

$$W_1 = \begin{vmatrix} x_a & y_a & z_a & 1 \\ x_b & y_b & z_b & 1 \\ x_c & y_c & z_c & 1 \\ x_d & y_d & z_d & 1 \end{vmatrix},$$

and X_2, Y_2, and W_2 have similar expressions obtained replacing in the above determinants x_d with x_h, y_d with y_h, and z_d with z_h.

Evaluating the sign of Δ is equivalent to evaluating the signs of W_1, W_2 and of Δ'.

By using the notation introduced in section 2, we can rewrite X_i and Y_i as α^4, and W_i as α^3 $(i = 1, 2)$. Hence, Δ' is a degree-8 multivariate polynomial since it can be rewritten as

$$\alpha(\alpha^4\alpha^3 - \alpha^4\alpha^3) - \alpha(\alpha^4\alpha^3 - \alpha^4\alpha^3) + \alpha^4\alpha^3 - \alpha^4\alpha^4 \quad \longrightarrow^{(2,3,4)} \quad \alpha^8 + \alpha^8 \quad \longrightarrow^{(3)} \alpha^8. \qquad \square$$

An algorithm for nearest neighbor queries on a set S of point sites in 3D space is said to be *conventional* if it computes the explicit 3D Voronoi diagram $V(S)$ of S and then performs point location queries on $V(S)$ with a native method. Recall that the class of conventional nearest neighbor query algorithms includes the two efficient algorithms presented in the literature. A conventional proximity query algorithm needs to perform test primitive $\mathsf{left\text{-}right}(q_{xy}, e_{xy})$. Thus, by Lemma 17, we have Theorem 9.

THEOREM 9. *Conventional algorithms for the nearest neighbor query problem on a set of point sites in* 3D *space have degree at least* 8.

5.3. Implicit Voronoi diagrams. The definition of the implicit 3D Voronoi diagram $V^*(S)$ of a set of S of point sites in 3D space is a straightforward extension of the definition for 2D Voronoi diagrams given in section 4.3. Namely, $V^*(S)$ stores the topological structure of the 3D Voronoi diagram $V(S)$ of S (e.g., the data structure of [23]) and the following geometric information for each vertex and facet:

- For each vertex v of $V(S)$, $V^*(S)$ stores the semi-integer $(b+1)$-bit approximations $x^*(v)$, $y^*(v)$, and $z^*(v)$ of the x-, y-, and z-coordinates of v.
- For each facet f of $V(S)$ that is not parallel to any of three Cartesian planes, $V^*(S)$ stores the pair of sites $\ell(f)$ and $r(f)$ such that f is a portion of the perpendicular bisector of $\ell(f)$ and $r(f)$, and $\ell(f)$ is below $r(f)$.

The tests $\mathsf{above\text{-}below}(q, v)$, $\mathsf{left\text{-}right}(q, v)$, $\mathsf{front\text{-}rear}(q, v)$ can be implemented comparing the x-, y-, and z-coordinate of query point q with $x(v)^*$, $y(v)^*$, and $z(v)^*$, respectively. With the same reasoning as for the 2D case (see section 4.3), it is easy to see that such implementations are correct.

LEMMA 18. *Test primitives* above-below(q, v), left-right(q, v), front-rear(q, v) *in an implicit Voronoi diagram of* 3D *point sites can be performed in* $O(1)$ *time and with degree* 1.

Test primitive above-below(q, f) is implemented by comparing the Euclidean distances of point q from the two sites $\ell(e)$ and $r(e)$ of which f is the perpendicular bisector with a point-to-points distance test. The implementation is correct by the definition of Voronoi facet. Thus, by Lemma 4, we have Lemma 19.

LEMMA 19. *Test primitive* above-below(q, f) *in an implicit Voronoi diagram of* 3D *point sites can be performed in* $O(1)$ *time and with degree* 2.

Finally, test left-right(q_{xy}, e_{xy}) is implemented by determining the sign of the equation of the line that contains edge e_{xy} when computed at point q_{xy}.

LEMMA 20. *Test primitive* left-right(q_{xy}, e_{xy}) *in an implicit Voronoi diagram of* 3D *point sites can be performed in* $O(1)$ *time and with degree* 3.

Proof. The line containing the oriented edge e_{xy} is the projection on the xy-plane of the intersection of two planes containing two facets of the 3D Voronoi diagram. Let $a_i x + b_i y + c_i z + d_i = 0$ be the equation of one such plane $(i = 1, 2)$. The projection of their intersection on the xy-plane is

$$\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix} x + \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix} y + \begin{vmatrix} d_1 & c_1 \\ d_2 & c_2 \end{vmatrix} = 0.$$

Test left-right(q_{xy}, e_{xy}) consists of determining the sign of

$$\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix} x_q + \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix} y_q + \begin{vmatrix} d_1 & c_1 \\ d_2 & c_2 \end{vmatrix},$$

which is a multivariate polynomial having arithmetic degree 3, since it can be rewritten as

$$\alpha\alpha^2 + \alpha\alpha^2 + \alpha^3 \longrightarrow^{(2,3)} \alpha^3. \qquad \square$$

In order to execute a native point location algorithm in an implicit 3D Voronoi diagram, we only need to redefine the implementation of the five test primitives. By having encapsulated the geometry in the test primitives, no further modifications are needed. Hence, by Lemmas 18–20 we obtain Lemma 21.

LEMMA 21. *For any native method on a class of cell-complexes that includes* 3D *Voronoi diagrams, a point location query in an implicit* 3D *Voronoi diagram has degree* 3 *and has the same asymptotic time complexity as a point location query in an explicit* 3D *Voronoi diagram.*

The Voronoi diagram of n point sites in 3D space is an acyclic convex cell-complex with $N = O(n^2)$ facets. Hence, using the separating surfaces method on the implicit 3D Voronoi diagram yields the following result.

The implicit Voronoi diagram $V^*(S)$ of a set S of n points in 3D space can be constructed by computing the 3D Delaunay triangulation with the incremental algorithm by Joe [40], whose time complexity and storage is $O(n^2)$ (see also [49]). Since the most demanding operation of the algorithm in terms of degree is the 3D insphere test, from Lemma 6 we have that the degree of the algorithm that computes $V(S)$ is 5. As in the planar case, the topological structure of $V(S)$ and the sites $\ell(f)$ and $r(f)$ for each edge e of $V(S)$ are immediately derived from $DT(S)$ by duality. We then compute the approximations $x^*(v)$, $y^*(v)$, and $z^*(v)$ for each vertex v of $V(S)$ by means of integer division. Let a, b, c, and d be the four sites of S that define vertex v. Adopting the same notation as in the proof of Lemma 17, the x-coordinate $x(v)$ of v is given by the ratio $x(v) = \frac{X_1}{2W_1}$, where X_1 is a variable of arithmetic degree 4

and W_1 is a variable of arithmetic degree 3; this is similar for $y(v)$ and $z(v)$. We summarize the above analysis as follows.

LEMMA 22. *The implicit Voronoi diagram of a set of n point sites in* 3D *space can be computed in $O(n^2)$ time and space and with degree* 5.

Lemmas 21 and 22 lead to the following theorem.

THEOREM 10. *Let S be a set of n point sites in* 3D *space. There exists an $O(n^2)$-space data structure for S that can be computed in $O(n^2)$ time with degree* 7 *and supports nearest neighbor queries in $O(\log^2 n)$ time with degree* 3.

Proof. We perform point location in the implicit Voronoi $V^*(S)$ diagram of S using the separating surfaces method. The space requirement and the query degree and time follow from the performance of these methods and from Lemma 21.

Regarding the preprocessing time, by Lemma 22, the construction of the implicit Voronoi $V^*(S)$ takes $O(n^2)$ time with degree 5. In order to construct the point-location data structure, we also need an additional test primitive that consists of comparing the y-coordinates of two Voronoi vertices. For example, this primitive is used to establish bridges between the vertices of the different separating chains if the bridged-chain method (see section 4.1) is applied to locate the xy-projection of the query point into the xy-projection of a separating surface. Comparing the y-coordinates of the Voronoi vertices is equivalent to evaluating the sign of multivariate polynomials of the form $\frac{Y_i}{2W_i} - \frac{Y_j}{2W_j}$, where $\frac{Y_i}{2W_i}$ and $\frac{Y_j}{2W_j}$ represent the y-coordinates of two different Voronoi vertices (see also the proof of Lemma 17). Such multivariate polynomials have degree 7, since they can be rewritten as

$$\frac{\rho_i}{\rho_j} - \frac{\rho_h}{\rho_k} \quad \longrightarrow^{(6)} \quad \rho_i\rho_k - \rho_h\rho_j \quad \longrightarrow^{(1)} \quad \alpha^4\alpha^3 - \alpha^4\alpha^3 \quad \longrightarrow^{(2,3,4)} \quad \alpha^7. \qquad \square$$

Although the algorithm for nearest neighbor queries proposed in this section has nonoptimal degree 3, it is a practical approach for the important application scenario where the primitive points are pixels on a computer screen. On a typical screen with about $2^{10} \times 2^{10}$ pixels, our nearest neighbor query can be executed with the standard integer arithmetic of a 32-bit processor.

6. Proximity queries for point and segment sites in the plane. In this section, we consider the following proximity query on a set S of point and segment sites in the plane:

nearest neighbor search: given query point q, find a site of S whose Euclidean distance from q is less than or equal to that of any other site.

As for the other queries studied in the previous sections, such a query is efficiently solved by performing point location in the Voronoi diagram of the set of point and segment sites [51].

The test primitives needed by such an approach are described in section 6.1. Section 6.2 shows that the "conventional" approach requires degree 64. A degree-6 algorithm based on "implicit" Voronoi diagrams is then given in section 6.3.

6.1. Test primitives and methods. The Voronoi diagram $V(S)$ of a set S of point and segment sites is a map whose edges are either straight-line segments or arcs of parabolas. Hence, in general $V(S)$ is neither convex nor monotone. In order to perform point location in $V(S)$, we refine $V(S)$ into a map with monotone edges as follows. If edge e of $V(S)$ is an arc of parabola whose point p of maximum (or minimum) y-coordinate is not a vertex, we split e into two edges by inserting a fictitious vertex at point p. We call the resulting map the *extended Voronoi diagram* $V'(S)$ of S.

The persistent search tree method and the trapezoid method can be used as native methods on the extended Voronoi diagram, where the test primitives are the same as

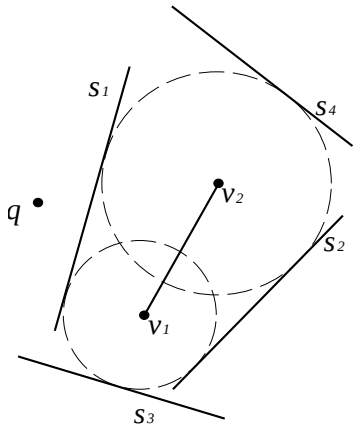


FIG. 2. *Illustration for Lemma* 24.

those defined in section 4.1 for point sites. If we want to use the chain method or the bridged-chain method, we need to do a further refinement that transforms the map into a monotone map by adding vertical fictitious edges emanating from the fictitious vertices previously inserted along the parabolic edges.

LEMMA 23. *The trapezoid method and the persistent search tree method are native, and the chain method and the bridged-chain method are ordinary for extended Voronoi diagrams of point and segment sites.*

6.2. Explicit Voronoi diagrams. Let S be a set of n points and segment sites. The extended Voronoi diagram $V'(S)$ of S is said to be *explicit* if the coordinates of the vertices of $V'(S)$ are computed and stored with exact arithmetic, i.e., as algebraic numbers [10, 59].

In the following lemma, we analyze the degree of test primitive $\mathsf{left\text{-}right}(q, e)$ for a straight-line edge e of an explicit extended Voronoi diagram.

LEMMA 24. *The* $\mathsf{left\text{-}right}(q, e)$ *test primitive for a straight-line edge* e *in an explicit extended Voronoi diagram of point and segment sites in the plane has degree* 64.

Proof. Let $e \equiv (v_1, v_2)$, such that $v_1 \equiv (x_1, y_1)$ is equidistant from three segments s_1, s_2, and s_3 and v_2 is from three segments s_1, s_2, and s_4. See Figure 2.

We show that the test $\mathsf{left\text{-}right}(q, e)$ for determining whether the query point $q \equiv (x_q, y_q)$ is to the left or to the right of (v_1, v_2) has degree 64. Namely, let $a_i x + b_i y + c_i = 0$ $(i = 1, 2, 3, 4)$ be the equation of the line containing segment s_i. In an explicit Voronoi diagram, test primitive $\mathsf{left\text{-}right}(q, e)$, determines whether query point $q \equiv (x_q, y_q)$ is to the left or to the right of edge $e \equiv (v_1, v_2)$, is equivalent to evaluating the sign of the following determinant:

$$\Delta = \begin{vmatrix} x_q & y_q & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = \begin{vmatrix} x_q & y_q & 1 \\ \frac{X_1}{W_1} & \frac{Y_1}{W_1} & 1 \\ \frac{X_2}{W_2} & \frac{Y_2}{W_2} & 1 \end{vmatrix} = \frac{1}{W_1 W_2} \begin{vmatrix} x_q & y_q & 1 \\ X_1 & Y_1 & W_1 \\ X_2 & Y_2 & W_2 \end{vmatrix} = \frac{\Delta'}{W_1 W_2},$$

where

$$X_1 = \begin{vmatrix} b_1 & c_1 & \sqrt{a_1^2 + b_1^2} \\ b_2 & c_2 & \sqrt{a_2^2 + b_2^2} \\ b_3 & c_3 & \sqrt{a_3^2 + b_3^2} \end{vmatrix}, \quad Y_1 = \begin{vmatrix} a_1 & c_1 & \sqrt{a_1^2 + b_1^2} \\ a_2 & c_2 & \sqrt{a_2^2 + b_2^2} \\ a_3 & c_3 & \sqrt{a_3^2 + b_3^2} \end{vmatrix},$$

$$W_1 = \begin{vmatrix} b_1 & a_1 & \sqrt{a_1^2 + b_1^2} \\ b_2 & a_2 & \sqrt{a_2^2 + b_2^2} \\ b_3 & a_3 & \sqrt{a_3^2 + b_3^2} \end{vmatrix},$$

and X_2, Y_2, and W_2 have similar expressions obtained by substituting in the above determinants a_3 with a_4, b_3 with b_4, and c_3 with c_4. Evaluating the sign of Δ is equivalent to evaluating the signs of W_1, W_2 and of Δ'. In the rest of this proof we show that evaluating the sign of Δ' is a computation with degree 64. By using the same technique, one can easily see that evaluating the signs of W_1 and W_2 is a computation with degree 12.

We have

$$\Delta' = x_q(Y_2W_1 - Y_1W_2) - y_q(X_1W_2 - X_2W_1) + (X_2Y_1 - X_1Y_2). \tag{1}$$

By using the notation introduced in section 2, we can rewrite X_1, and Y_1 as $\alpha^3\rho_1 + \alpha^3\rho_2 + \alpha^3\rho_3$, W_1 as $\alpha^2\rho_1 + \alpha^2\rho_2 + \alpha^2\rho_3$, X_2 and Y_2 as $\alpha^3\rho_1 + \alpha^3\rho_2 + \alpha^3\rho_4$, and W_2 as $\alpha^2\rho_1 + \alpha^2\rho_2 + \alpha^2\rho_4$, where $\rho_i = \sqrt{a_i^2 + b_i^2}$ $(i = 1, \ldots, 4)$. Considering that x_q and y_q are expressions of type α and applying repeatedly Rules (1) and (2), we obtain the expression

$$\alpha^8 + \alpha^6\rho_1\rho_2 + \alpha^6\rho_1\rho_3 + \alpha^6\rho_1\rho_4 + \alpha^6\rho_2\rho_3 + \alpha^6\rho_2\rho_4 + \alpha^6\rho_3\rho_4.$$

By means of the rewriting rules of section 2 we have

$$\begin{array}{ll}
\alpha^8 + \alpha^6\rho_1\rho_2 + \alpha^6\rho_1\rho_3 + \alpha^6\rho_1\rho_4 + \alpha^6\rho_2\rho_3 + \alpha^6\rho_2\rho_4 + \alpha^6\rho_3\rho_4 & \longrightarrow^{(7)} \\
(\alpha^8 + \alpha^6\rho_2\rho_3 + \alpha^6\rho_2\rho_4 + \alpha^6\rho_3\rho_4)^2 - (\alpha^6\rho_1\rho_2 + \alpha^6\rho_1\rho_3 + \alpha^6\rho_1\rho_4)^2 & \longrightarrow^{(1,2,3,4)} \\
\alpha^{16} + \alpha^{14}\rho_2\rho_3 + \alpha^{14}\rho_2\rho_4 + \alpha^{14}\rho_3\rho_4 & \longrightarrow^{(7)} \\
(\alpha^{16} + \alpha^{14}\rho_2\rho_3)^2 - (\alpha^{14}\rho_2\rho_4 + \alpha^{14}\rho_3\rho_4)^2 & \longrightarrow^{(1,2,3,4)} \\
\alpha^{32} + \alpha^{30}\rho_2\rho_3 & \longrightarrow^{(7)} \\
\alpha^{64} - \alpha^{64} & \longrightarrow^{(3,4)} \\
\alpha^{64}. &
\end{array}$$

□

An algorithm for proximity queries on a set S of point and segment sites in the plane is said to be *conventional* if it computes the explicit extended Voronoi diagram $V'(S)$ of S and then performs point location queries on $V'(S)$ with a native method. Note that the class of conventional proximity query algorithms includes all the efficient algorithms presented in the literature. A conventional proximity query algorithm needs to perform test primitive left-right(q, e). Thus, by Lemma 24 we conclude Theorem 11.

THEOREM 11. *Conventional algorithms for the nearest neighbor query problem on a set of point and segment sites in the plane have degree at least* 64.

Our analysis shows that performing point location in an explicit Voronoi diagram of points and segments is not practically feasible due to the high degree.

6.3. Implicit Voronoi diagrams. The definition of the implicit Voronoi diagram $V^*(S)$ of a set of S of point and segment sites is a straightforward extension of the definition for Voronoi diagrams of point sites given in section 4.3. Namely, $V^*(S)$ stores the topological structure of the extended Voronoi diagram $V'(S)$ of S (e.g., the data structure of [23]) and the following geometric information for each vertex and edge:

- For each vertex v of $V'(S)$, $V^*(S)$ stores the semi-integer $(b+1)$-bit approximations $x^*(v)$ and $y^*(v)$ of the x- and y-coordinates of v.
- For each nonhorizontal edge e of $V'(S)$, $V^*(S)$ stores the pair of sites $\ell(e)$ and $r(e)$ such that e is a portion of the bisector of $\ell(e)$ and $r(e)$, and $\ell(e)$ is to the left of $r(e)$.

In the implicit Voronoi diagram $V^*(S)$ of S, test left-right(q, e) is implemented by comparing the distances of query point q from sites $\ell(e)$ and $r(e)$ with one of the

following tests, depending on the type (point or line) of sites $\ell(e)$ and $r(e)$: point-to-lines distance test, point-to-point-line distance test, or point-to-points distance test. Thus, by Lemmas 2–4, we have Lemma 25.

LEMMA 25. *For any native method on a class of maps that includes extended Voronoi diagrams of point and segment sites in the plane, a point location query in an implicit Voronoi diagram has degree* 6 *and has the same asymptotic time complexity as a point location query in an explicit Voronoi diagram.*

The implicit Voronoi diagram can be constructed in $O(n \log n)$ expected running time by using the randomized incremental algorithm of [11]. The most demanding operation is incircle test for three segments, which has degree 40 by Lemma 7 (see also [9]). By a similar analysis as the one shown in sections 4 and 5, it is not hard to show that both the y-ordering of the vertices of $V(S)$ and the semi-integer approximation of the vertex coordinates can be performed without affecting the computational cost and the degree of the computation of $V(S)$.

LEMMA 26. *The implicit Voronoi diagram of a set of n point and segment sites in the plane can be computed in $O(n \log n)$ expected time, $O(n)$ space, and degree* 40.

Lemmas 25 and 26 lead to the following theorem.

THEOREM 12. *Let S be a set of n point and segment sites in the plane. There exists an $O(n)$-space data structure for S that can be computed in $O(n \log n)$ expected time with degree* 40 *and supports nearest neighbor queries in $O(\log n)$ time with degree* 6.

7. Simplified implicit Voronoi diagrams. In this section, we describe a modification of implicit Voronoi diagrams of point sites that allows us to reduce the degree of the preprocessing task from 5 to 4 when the sites are in the plane (see Theorems 6–8), and from 7 to 5 when the sites are in 3D space (see Theorem 10). This modification also has a positive impact on the space requirement of the data structure and on the running time of point location queries.

Let $V(S)$ be the Voronoi diagram of a set S of point sites in the plane. We recall our standard assumption that all input parameters — such as coordinates of sites and query points — are represented as b-bit integers.

An *island* of $V(S)$ is a connected component of the map obtained from $V(S)$ by removing all the vertices with integer y-coordinate and all the edges containing a point with integer y-coordinate. Note that for any two vertices v_1 and v_2 of an island, $y^*(v_1) = y^*(v_2) = m + \frac{1}{2}$ for some integer m, where $y^*(v)$ is the semi-integer approximation defined in section 4.3.

The *simplified implicit Voronoi diagram* $V^\circ(S)$ of S is a representation of the Voronoi diagram $V(S)$ of S that consists of a topological component and a geometric component. The topological component of $V^\circ(S)$ is the planar embedding obtained from $V(S)$ by contracting each island of $V(S)$ into an *alias vertex*. The geometric component of $V^\circ(S)$ stores the following geometric information for each vertex and edge of the embedding:

- For each vertex v that is also a vertex of $V(S)$, $V^\circ(S)$ stores the $(b+1)$-bit semi-integers approximations $x^*(v)$ and $y^*(v)$.
- For each alias vertex a, which is associated with an island of $V(S)$, $V^\circ(S)$ stores semi-integer $y^*(a)$ such that $y^*(a) = y^*(v)$ for each vertex v of the island.
- For each nonhorizontal edge e that is also an edge of $V(S)$, $V^\circ(S)$ stores the pair of sites $\ell(e)$ and $r(e)$ such that e is a portion of the perpendicular bisector of $\ell(e)$ and $r(e)$, and $\ell(e)$ is to the left of $r(e)$.

The space requirement of the simplified implicit Voronoi diagram is less than or equal to that of the implicit Voronoi diagram, since each island is represented

by a single alias vertex storing only its semi-integer y-approximation. We can show examples where the simplified implicit Voronoi diagram of n point sites has $O(n)$ fewer vertices and edges than the corresponding implicit Voronoi diagram.

The following lemmas extend Lemmas 12–13 and can be proved with similar techniques.

LEMMA 27. *For any native method on a class of maps that includes monotone maps, a point location query in a simplified implicit Voronoi diagram has optimal degree* 2 *and executes a number of operations less than or equal to a point location query in the corresponding explicit Voronoi diagram.*

LEMMA 28. *The simplified implicit Voronoi diagram of n point sites in the plane can be computed in* $O(n \log n)$ *time,* $O(n)$ *space, and with degree* 4.

The main advantage of the simplified implicit Voronoi diagram with respect to the degree cost measure is that the additional test primitive needed in the preprocessing that consists of comparing the y-coordinates of two Voronoi vertices (see the proof of Theorem 6) is now reduced to the comparison of two $(b + 1)$-bit semi-integers, and thus has degree 1. Hence, the preprocessing for point location using a native method for monotone maps has degree 1.

By the above discussion and Lemmas 27–28, we obtain the following theorem that improves upon Theorem 6.

THEOREM 13. *Let S be a set of n point sites in the plane. There exists an* $O(n)$*-space data structure for S, based on the simplified implicit Voronoi diagram* $V^{\circ}(S)$*, that can be computed in* $O(n \log n)$ *time with degree* 4 *and supports nearest neighbor queries in* $O(\log n)$ *time with optimal degree* 2.

Using a similar approach, we can define simplified implicit order-k Voronoi diagrams for point sites in the plane and simplified implicit Voronoi diagrams for point sites in 3D space. This reduces the degree of the preprocessing from 5 to 4 in Theorems 7–8 and from 7 to 5 in Theorem 10.

8. Further research directions. Within the proposed approach, this paper only addresses the issue of the degree of test computations and illustrates its impact on algorithmic design in relation to a central problem in computational geometry. However, several important related problems need further investigation and will be reported on in the near future.

First, the methodological framework described in section 2 should be extended to the computation of the degree of other classes of geometric primitives. Recently, motivated in part by a preliminary version of this paper [45], Burnikel et al. [13] have presented a new separation bound for arithmetic expressions involving square roots.

Also, since the degree of an algorithm expresses worst-case computational requirement occurring in degenerate or near-degenerate instances, special attention must be devoted to the development of a methodology that reliably computes the sign of an expression with the least expenditure of computational resources. This involves the use of "arithmetic filters," possibly families of filters, of progressively increasing power that, depending upon the values of primitive variables, carefully adjust the computational effort (see, e.g., [4, 11, 29, 41]).

Next, one should carefully analyze the precision adopted in executing constructions, so that the outputs are within the precision bounds required by the application. In addition, each construction algorithm should be accompanied by a verification algorithm, which not only checks for topological compliance of the output with the generic member of its class (as, e.g., a Voronoi diagram must have the topology of a convex map) as illustrated in [54] but more specifically verifies its topological agreement with the structure emerging from the tests executed by the algorithm [47].

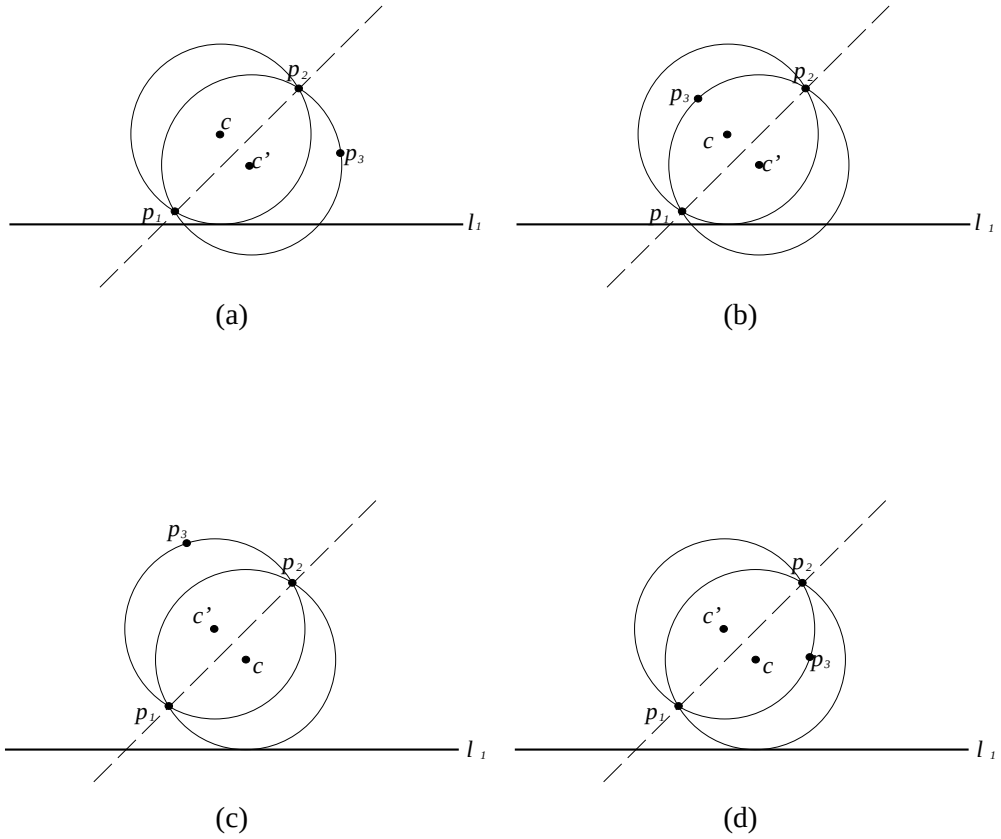


FIG. 3. *Different cases for test* $(p_1, p_2, l_1; p_3)$.

Beyond these general methodological issues, the investigation reported in these pages leaves some interesting open problems such as answering nearest neighbor queries in subquadratic time and optimal degree for a set of points in 3D space or improving the efficiency of the preprocessing stage in computing the implicit Voronoi diagram of a set of sites.

We mention, in this respect, how the degree can impact the design of geometric primitives adopted in existing algorithms for Voronoi diagrams of point and segment sites. Let $(a_1, a_2, a_3; a_4)$, with a_i either a point or a segment, denote the incircle test, where a_4 is tested for intersection with circle(a_1, a_2, a_3). Specifically, consider $(p_1, p_2, l_1; p_3)$, which can be answered with degree 12 [9]. We show that it can be more efficiently executed as follows. First perform the test $(p_1, p_2, p_3; l_1)$. Let c and c' be the centers of circle(p_1, p_2, l_1) and circle(p_1, p_2, p_3), respectively. Two cases are possible: either circle(p_1, p_2, p_3) intersects l_1 or it does not. In the first case, p_3 is inside circle(p_1, p_2, l_1) if and only if c' and p_3 lie on opposite sides of line $\overline{p_1 p_2}$ through p_1 and p_2 (see Figures 3(a) and 3(b)). In the second case the answer to test $(p_1, p_2, l_1; p_3)$ depends on which side of $\overline{p_1 p_2}$ point p_3 lies (see Figures 3(c) and 3(d)). Thus, test $(p_1, p_2, l_1; p_3)$ is reduced to test $(p_1, p_2, p_3; l_1)$ that can be executed with degree 8 (see [9]) and at most two other left-right tests of lower degree.

Finally, an important issue for future research deals with the experimental comparison between point location algorithms in implicit Voronoi diagrams and traditional point location algorithms in explicit Voronoi diagrams. We are currently implementing *GeomLib*, an object-oriented library for robust geometric computing that will

be accessible through the world wide web by using the architectural framework of *Mocha* [6, 5].

Acknowledgment. The authors wish to thank the referees for several useful suggestions.

REFERENCES

[1] A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor, *A linear-time algorithm for computing the Voronoi diagram of a convex polygon*, Discrete Comput. Geom., 4 (1989), pp. 591–604.
[2] A. Aggarwal, M. Hansen, and T. Leighton, *Solving query-retrieval problems by compacting Voronoi diagrams*, in Proc. 22nd Annu. ACM Sympos. Theory Comput., 1990, Association for Computing Machinery, New York, pp. 331–340.
[3] F. Aurenhammer, *Voronoi diagrams: A survey of a fundamental geometric data structure*, ACM Comput. Surv., 23 (1991), pp. 345–405.
[4] F. Avnaim, J.-D. Boissonnat, O. Devillers, F. Preparata, and M. Yvinec, *Evaluating Signs of Determinants Using Single-Precision Arithmetic*, Research Report 2306, INRIA, BP93, 06902 Sophia-Antipolis, France, 1994.
[5] J. E. Baker, I. F. Cruz, G. Liotta, and R. Tamassia, *The Mocha algorithm animation system*, ACM Comput. Surv., 27 (1995), pp. 568–572.
[6] J. E. Baker, I. F. Cruz, G. Liotta, and R. Tamassia, *Animating geometric algorithms over the Web*, in Proc. 12th Annu. ACM Sympos. Comput. Geom., Association for Computing Machinery, New York, 1996, pp. C3–C4.
[7] J. L. Bentley and H. A. Maurer, *A note on Euclidean near neighbor searching in the plane*, Inform. Process. Lett., 8 (1979), pp. 133–136.
[8] M. Bocher, *Introduction to Higher Algebra*, Macmillan, New York, 1907.
[9] C. Burnikel, *Exact Computation of Voronoi Diagrams and Line Segment Intersections*. Ph.D thesis, Universität des Saarlandes, Mar. 1996.
[10] C. Burnikel, J. Könnemann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig, *Exact geometric computation in LEDA*, in Proc. 11th Annu. ACM Sympos. Comput. Geom., Association for Computing Machinery, New York, 1995, pp. C18–C19.
[11] C. Burnikel, K. Mehlhorn, and S. Schirra, *How to compute the Voronoi diagram of line segments: Theoretical and experimental results*, in 2nd Annual European Symp. on Algorithms, Lecture Notes Comput. Sci. 855, Springer-Verlag, Berlin, 1994, pp. 227–239.
[12] C. Burnikel, K. Mehlhorn, and S. Schirra, *On degeneracy in geometric computations*, in Proc. 5th ACM-SIAM Sympos. Discrete Algorithms, 1994, pp. 16–23.
[13] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra, *A strong and easily computable separation bound for arithmetic expressions involving square roots*, in Proc. ACM-SIAM Symposium on Discrete Algorithms, 1997.
[14] B. Chazelle, *How to search in history*, Inform. Control, 64 (1985), pp. 77–99.
[15] B. Chazelle, R. Cole, F. P. Preparata, and C. K. Yap, *New upper bounds for neighbor searching*, Inform. Control, 68 (1986), pp. 105–124.
[16] B. Chazelle and H. Edelsbrunner, *An improved algorithm for constructing kth-order Voronoi diagrams*, IEEE Trans. Comput., C-36 (1987), pp. 1349–1354.
[17] B. Chazelle and L. J. Guibas, *Fractional cascading:* I. *A data structuring technique*, Algorithmica, 1 (1986), pp. 133–162.
[18] B. Chazelle and L. J. Guibas, *Fractional cascading:* II. *Applications*, Algorithmica, 1 (1986), pp. 163–191.
[19] K. L. Clarkson, *Safe and effective determinant evaluation*, in Proc. 33rd Ann. IEEE Sympos. Found. Comput. Sci., IEEE Press, Piscataway, NJ, 1992, pp. 387–395.
[20] T. K. Dey, K. Sugihara, and C. L. Bajaj, *Delaunay triangulations in three dimensions with finite precision arithmetic*, Comput. Aided Geom. Design, 9 (1992), pp. 457–470.
[21] D. P. Dobkin, *Computational geometry and computer graphics*, in Proc. IEEE, 80 (1992), pp. 1400–1411.
[22] D. P. Dobkin and D. G. Kirkpatrick, *Fast detection of polyhedral intersection*, Theoret. Comput. Sci., 27 (1982), pp. 241–253.
[23] D. P. Dobkin and M. J. Laszlo, *Primitives for the manipulation of three-dimensional subdivisions*, Algorithmica, 4 (1989), pp. 3–32.
[24] M. Edahiro, I. Kokubo, and T. Asano, *A new point-location algorithm and its practical efficiency: Comparison with existing algorithms*, ACM Trans. Graph., 3 (1984), pp. 6–109.
[25] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, *Optimal point location in a monotone subdivision*, SIAM J. Comput., 15 (1986), pp. 317–340.

[26] H. Edelsbrunner and H. A. Maurer, *Finding extreme points in three dimensions and solving the post-office problem in the plane*, Inform. Process. Lett., 21 (1985), pp. 39–47.
[27] H. Edelsbrunner and E. P. Mücke, *Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms*, ACM Trans. Graph., 9 (1990), pp. 66–104.
[28] S. Fortune, *Stable maintenance of point set triangulations in two dimensions*, in Proc. 30th Ann. IEEE Sympos. Found. Comput. Sci., IEEE Press, Piscataway, NJ, 1989, pp. 494–505.
[29] S. Fortune, *Numerical stability of algorithms for* 2*-d Delaunay triangulations*, Internat. J. Comput. Geom. Appl., 5 (1995), pp. 193–213.
[30] S. Fortune, *Polyhedral modeling with multiprecision integer arithmetic*, Comput. Aided Design, to appear.
[31] S. Fortune and C. J. Van Wyk, *Efficient exact arithmetic for computational geometry*, in Proc. 9th Annu. ACM Sympos. Comput. Geom., Association for Computing Machinery, New York, 1993, pp. 163–172.
[32] S. Fortune and C. V. Wyk, *Static analysis yields efficient exact integer arithmetic for computational geometry*, ACM Trans. Graphics, 15 (1996), pp. 223–248.
[33] S. J. Fortune, *A sweepline algorithm for Voronoi diagrams*, Algorithmica, 2 (1987), pp. 153–174.
[34] D. H. Greene and F. F. Yao, *Finite-resolution computational geometry*, in Proc. 27th Ann. IEEE Sympos. Found. Comput. Sci., IEEE Press, Piscataway, NJ, 1986, pp. 143–152.
[35] L. J. Guibas, D. E. Knuth, and M. Sharir, *Randomized incremental construction of Delaunay and Voronoi diagrams*, Algorithmica, 7 (1992), pp. 381–413.
[36] L. J. Guibas, D. Salesin, and J. Stolfi, *Epsilon geometry: Building robust algorithms from imprecise computations*, in Proc. 5th Ann. ACM Sympos. Comput. Geom., Association for Computing Machinery, New York, 1989, pp. 208–217.
[37] L. J. Guibas and J. Stolfi, *Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams*, ACM Trans. Graph., 4 (1985), pp. 74–123.
[38] C. M. Hoffmann, *The problems of accuracy and robustness in geometric computation*, IEEE Computer, 22 (1989), pp. 31–41.
[39] C. M. Hoffmann, J. E. Hopcroft, and M. T. Karasick, *Robust set operations on polyhedral solids*, IEEE Comput. Graph. Appl., 9 (1989), pp. 50–59.
[40] B. Joe, *Construction of three-dimensional Delaunay triangulations using local transformations*, Comput. Aided Geom. Design, 8 (1991), pp. 123–142.
[41] M. Karasick, D. Lieber, and L. R. Nackman, *Efficient Delaunay triangulations using rational arithmetic*, ACM Trans. Graph., 10 (1991), pp. 71–91.
[42] D. G. Kirkpatrick, *Optimal search in planar subdivisions*, SIAM J. Comput., 12 (1983), pp. 28–35.
[43] D. T. Lee, *On k-nearest neighbor Voronoi diagrams in the plane*, IEEE Trans. Comput., C-31 (1982), pp. 478–487.
[44] D. T. Lee and F. P. Preparata, *Location of a point in a planar subdivision and its applications*, SIAM J. Comput., 6 (1997), pp. 594–606.
[45] G. Liotta, F. P. Preparata, and R. Tamassia, *Robust Proximity Queries in Implicit Voronoi Diagrams*, Technical Report CS-96-16, Center for Geometric Computing, Comput. Sci. Dept., Brown Univ., Providence, RI, 1996.
[46] K. Mehlhorn and S. Näher, *LEDA: A platform for combinatorial and geometric computing*, Comm. ACM, 38 (1995), pp. 96–102.
[47] K. Mehlhorn, S. Näher, T. Schilz, S. Schirra, M. Seel, R. Seidel, and C. Uhrig, *Checking geometric programs or verification of geometric structures*, in Proc. 12th Ann. ACM Sympos. Comput. Geom., Association for Computing Machinery, New York, 1996, pp. 159–165.
[48] V. J. Milenkovic, *Verifiable implementations of geometric algorithms using finite precision arithmetic*, Artif. Intell., 37 (1988), pp. 377–401.
[49] E. Mücke, *Detri* 2.2: *A robust implementation for* 3d *Triangulations*, manuscript, http://www.geom.umn.edu:80/software/cglist/lowdvod.html (1996).
[50] F. P. Preparata, *A new approach to planar point location*, SIAM J. Comput., 10 (1981), pp. 473–482.
[51] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
[52] F. P. Preparata and R. Tamassia, *Efficient point location in a convex spatial cell-complex*, SIAM J. Comput., 21 (1992), pp. 267–280.
[53] N. Sarnak and R. E. Tarjan, *Planar point location using persistent search trees*, Comm. ACM, 29 (1986), pp. 669–679.
[54] K. Sugihara and M. Iri, *Construction of the Voronoi diagram for 'one million' generators in single-precision arithmetic*, Proc. IEEE, IEEE Press, Piscataway, NJ, 80 (1992), pp. 1471–1484.

[55] K. Sugihara, Y. Ooishi, and T. Imai, *Topology-oriented approach to robustness and its applications to several Voronoi-diagram algorithms*, in Proc. 2nd Canad. Conf. Comput. Geom., 1990, pp. 36–39.
[56] R. Tamassia and J. S. Vitter, *Optimal cooperative search in fractional cascaded data structures*, Algorithmica, 15 (1996), pp. 154–171.
[57] C. Yap and T. Dubé, *A Basis for Implementing Exact Geometric Algorithms*, manuscript, http://simulation.nyu.edu/projects/exact/references.html (1993).
[58] C. K. Yap, *Symbolic treatment of geometric degeneracies*, J. Symbolic Comput., 10 (1990), pp. 349–370.
[59] C. K. Yap, *Toward exact geometric computation*, Comput. Geom., 7 (1997), pp. 3–23.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 890–903

FAILURE DETECTION AND RANDOMIZATION: A HYBRID APPROACH TO SOLVE CONSENSUS*

MARCOS KAWAZOE AGUILERA† AND SAM TOUEG†

Abstract. We present a consensus algorithm that combines unreliable failure detection and randomization, two well-known techniques for solving consensus in asynchronous systems with crash failures. This hybrid algorithm combines advantages from both approaches: it guarantees deterministic termination if the failure detector is accurate, and probabilistic termination otherwise. In executions with no failures or failure detector mistakes, the most likely ones in practice, consensus is reached in only two asynchronous rounds.

Key words. algorithms, reliability, agreement problem, asynchronous systems, Byzantine generals' problem, consensus problem, crash failures, failure detection, fault-tolerance, message passing, processor failures, randomized algorithms

AMS subject classifications. 68Q22, 68M15

PII. S0097539796312915

1. Introduction.

1.1. Motivation.

A well-known result by Fischer, Lynch, and Paterson [14] is that *consensus* cannot be solved in asynchronous systems with failures, even if communication is reliable; at most one process may fail, and it can only fail by crashing. Since this seminal paper, there has been intense research seeking to "circumvent" this negative result (e.g., [4, 5, 6, 7, 10, 13, 22]).

One promising approach is the use of unreliable failure detection [2, 3, 6, 7, 11, 16, 17, 18, 19, 20, 21, 23]. Roughly speaking, this approach assumes that each process has access to a local failure detector module that gives some (possibly inaccurate) information on which processes may have failed. It turns out that consensus can be solved with unreliable failure detectors that make an infinite number of mistakes, provided that they satisfy some minimum properties [6, 7].

In particular, [7] presents a consensus algorithm with the following features. Even if the information provided by the failure detectors is completely wrong, the algorithm never violates safety; i.e., no two processes ever decide differently. During "good" periods, when the failure detectors are reasonably accurate, processes reach consensus within few asynchronous rounds; on the other hand, when a "bad" period occurs, i.e., when failure detectors lose their accuracy, the consensus algorithm may stop making progress until the bad period is over. Such an algorithm is useful because, in practice, good periods tend to be long while bad ones tend to be rare and short. However, long bad periods do occasionally occur, and each time this happens the consensus algorithm of [7] can be delayed for a long time.

In this paper, we seek an algorithm that terminates quickly when failure detection is accurate (i.e., during good periods) and that makes progress and terminates, albeit more slowly, even if failure detection is inaccurate (i.e., during bad periods). We

*Received by the editors November 27, 1996; accepted for publication (in revised form) March 4, 1998; published electronically December 23, 1998. This research was partially supported by NSF grants CCR-9402896 and CCR-9711403, ARPA/ONR grant N00014-96-1-1014, and an Olin Fellowship.

http://www.siam.org/journals/sicomp/28-3/31291.html

†Computer Science Department, Cornell University, Ithaca, NY 14853-7501 (aguilera@cs.cornell.edu, sam@cs.cornell.edu).

achieve this goal by combining failure detection with *randomization*—another technique that was used to solve consensus in asynchronous systems [4]. In this hybrid approach, randomization "kicks in" as a back-up to failure detection when failure detectors are inaccurate. Further discussion of the relative merits of failure detection, randomization, and this hybrid approach is postponed until section 7.

The idea of combining randomization and failure detection to solve consensus in asynchronous systems first appeared in [12]. A related idea, namely, combining randomization and deterministic algorithms to solve consensus in *synchronous* systems, was explored in [15, 25]. A brief comparison with our results is given in section 8.

1.2. Main result. We focus on two of the major techniques to circumvent the impossibility of consensus in asynchronous systems: randomization and unreliable failure detection. The first one assumes that each process has a local random number generator (denoted *R-oracle*) that provides *random bits* [4]. The second technique assumes that each process has a local failure detector module (denoted *FD-oracle*) that provides *a list of processes suspected to have crashed* [7]. Each approach has some advantages over the other, and we seek to combine advantages from both.

With a randomized consensus algorithm, every process can query its local R-oracle and use the oracle's random bit to determine its next step. With such an algorithm, termination is achieved with probability 1, within a finite expected number of steps (for a survey of randomized consensus algorithms see [8]).

With a failure detector based consensus algorithm, every process can query its local FD-oracle (which provides a list of processes that are suspected to have crashed) to determine the process's next step. Consensus can be solved with FD-oracles that make an infinite number of mistakes. In particular, consensus can be solved with FD-oracles that satisfy two properties, *strong completeness* and *eventual weak accuracy*. Roughly speaking, the first property states that every process that crashes is eventually suspected by every correct process, and the second one states that some correct process is eventually not suspected. These properties define the weakest class of failure detectors that can be used to solve consensus [6].

In this paper we describe a hybrid consensus algorithm with the following properties. Every process has access to both an R-oracle and an FD-oracle. If the FD-oracle satisfies the above two properties, the algorithm solves consensus (no matter how the R-oracle behaves). If the FD-oracle loses its accuracy property but the R-oracle works, the algorithm still solves consensus, albeit "only" with probability 1. In executions with no failures or failure detector mistakes, the most likely ones in practice, an optimized version of this algorithm reaches consensus in only two asynchronous rounds.

2. Informal model. Our model of asynchronous computation is patterned after the one in [14] and its extension in [6]. We only sketch its main features here. We consider *asynchronous* distributed systems in which there is no bound on message delay, clock drift, or the time necessary to execute a step. To simplify the presentation of our model, we assume the existence of a discrete global clock. This is merely a fictional device: the processes do not have access to it. We take the range $\mathcal{T}$ of the clock's ticks to be the set of natural numbers $\mathbf{N}$.

The system consists of a set of n *processes*, $\Pi = \{p_0, p_1, \ldots, p_{n-1}\}$. Every pair of processes is connected by a reliable communication channel. Up to f processes can fail by *crashing*. A failure pattern indicates which processes crash, and when they crash during an execution. Formally, a *failure pattern* F is a function from $\mathbf{N}$ to 2^{Π}, where $F(t)$ denotes the set of processes that have crashed through time t.

Once a process crashes, it does not "recover"; i.e., $\forall t : F(t) \subseteq F(t+1)$. We define $\text{crashed}(F) = \bigcup_{t \in \mathbf{N}} F(t)$ and $\text{correct}(F) = \Pi - \text{crashed}(F)$. If $p \in \text{crashed}(F)$ we say *p crashes (in F)* and if $p \in \text{correct}(F)$ we say *p is correct (in F)*.

Each process has access to two oracles: a failure detector, henceforth denoted the FD-oracle, and a random number generator, henceforth denoted the R-oracle. When a process queries its FD-oracle, it obtains a list of processes.[1] When it queries its R-oracle, it obtains a bit. The properties of these oracles are described in the two next sections.

A distributed algorithm $\mathcal{A}$ is a collection of n deterministic automata (one for each process in the system) that communicates by sending messages through reliable channels. The execution of $\mathcal{A}$ occurs in *steps* as follows. For every time $t \in \mathcal{T}$, at most one process takes a step. Each step consists of receiving a message, querying the FD-oracle, querying the R-oracle, changing state, and optionally sending a message to one process. We assume that messages are never lost. That is, if a process does not crash, it eventually receives every message sent to it.

A schedule is a sequence $\{s_j\}_{j \in \mathbf{N}}$ of processes and a sequence $\{t_j\}_{j \in \mathbf{N}}$ of strictly increasing times. A schedule indicates which processes take a step and when they take a step; for each j, process s_j takes a step at time t_j. A schedule is *consistent (with respect to a failure pattern F)* if a process does not take a step after it has crashed (in F). A schedule is *fair (with respect to a failure pattern F)* if each process that is correct (in F) takes an infinite number of steps. We consider only schedules that are consistent and fair.

2.1. FD-oracles. Every process p has access to a local FD-oracle module that outputs a list of processes that are suspected to have crashed. If some process q belongs to such list, we say that p *suspects* q.[2] FD-oracles can make mistakes: it is possible for a process p to be suspected by another even though p did not crash or for a process to crash and never be suspected. FD-oracles can be classified according to properties that limit the extent of such mistakes. We focus on one of the eight classes of FD-oracles defined in [7], namely, the class of *eventually strong* failure detectors, denoted $\diamond S$. An FD-oracle belongs to $\diamond S$ if and only if it satisfies two properties:

Strong completeness. Eventually every process that crashes is permanently suspected by *every* correct process (formally, $\exists t \in \mathcal{T}, \forall p \in \text{crashed}(F), \forall q \in \text{correct}(F), \forall t' \geq t : p \in \text{FD}_q^{t'}$, where $\text{FD}_q^{t'}$ denotes the output of q's FD-oracle module at time t').

Eventual weak accuracy. There is a time after which some correct process is never suspected by any correct process (formally, $\exists t \in \mathcal{T}, \exists p \in \text{correct}(F), \forall t' \geq t, \forall q \in \text{correct}(F) : p \notin \text{FD}_q^{t'}$).

It is known that $\diamond S$ is the weakest class of FD-oracles that can be used to solve consensus [6].

2.2. R-oracles. Each process has access to a local R-oracle module that outputs one bit each time it is queried. We say that the R-oracle is *random* if it outputs an independent random bit for each query. For simplicity, we assume a uniform distribution; i.e., a random R-oracle outputs 0 and 1, each with probability 1/2.

2.3. Adversary power. When designing fault-tolerant algorithms, we often assume that an intelligent adversary has some control on the behavior of the system;

[1] In general, the output of a failure detector is not restricted to be a list of processes [6, 1].

[2] In general, processes do not have to agree on the list of suspects at any one time or ever.

e.g., the adversary may be able to control the occurrence and the timing of process failures, the message delays, and the scheduling of processes. Adversaries may have limitations on their computing power and on the information that they can obtain from the system. Different algorithms are designed to defeat different types of adversaries [8].

We now describe the adversary that our hybrid algorithm defeats. The adversary has unbounded computational power and full knowledge of all process steps that already occurred. In particular, it knows the contents of all past messages, the internal state of all processes in the system,[3] and all the previous outputs of both the R-oracle and FD-oracle. With this information, at any time in the execution, the adversary can dynamically select which process takes the next step, which message this process receives (if any), and which processes (if any) crash. The adversary, however, operates under the following restrictions: the final schedule must be consistent and fair, every message sent to a correct process must be eventually received, and at most f processes may crash over the entire execution.

In addition to the above power, we allow the adversary to initially select *one* of the two oracles to control and possibly corrupt. If the adversary selects to control the R-oracle, it can predict and even determine the bits output by that oracle. For example, the adversary can force some local R-oracle module to always output 0 or it can dynamically adjust the R-oracle's output according to what the processes have done so far.

If the adversary selects to control the FD-oracle, it can ensure that the FD-oracle does not satisfy eventual weak accuracy. In other words, at *any* time the adversary can include *any* process (whether correct or not) in the output of the local FD-oracle module of any process. The adversary, however, does not have the power to disrupt the strong completeness property of the FD-oracle. This is not a limitation in practice; most failure detectors are based on time-outs and eventually detect all process crashes.

If the adversary does not control the R-oracle, then the R-oracle is random. If the adversary does not control the FD-oracle, then the FD-oracle is in $\diamond S$. We stress that the algorithm does *not* know which one of the two oracles (FD-oracle or R-oracle) is controlled by the adversary.

3. The consensus problem. In the *uniform binary consensus* problem, every process p has some *initial value* $v_p \in \{0, 1\}$ and must *decide* on a value such that we have the following:

Uniform agreement. If processes p and p' decide v and v', respectively, then $v = v'$;

Uniform validity. If some process decides v, then v is the initial value of some process;

Termination. Every correct process eventually decides some value.

For probabilistic consensus algorithms, Termination is weakened to the following:

Termination with probability 1. With probability 1, every correct process eventually decides some value.

4. Hybrid consensus algorithm. The hybrid consensus algorithm shown in Figure 4.1 combines Ben-Or's algorithm [4] with failure-detection and the rotating coordinator paradigm used in [7]. With this paradigm, we assume that all processes have a priori knowledge that during phase k, one selected process, namely, $p_{k \bmod n}$,

[3] This is in contrast to the assumptions made by several algorithms, e.g., those that use cryptographic techniques.

Every process p executes the following:

```
0   procedure consensus(v_p)                          {v_p is the initial value of process p}
        x ← v_p                                       {x is p's current estimate of the decision value}
        k ← 0
        while true do
            k ← k + 1                                 {k is the current phase number}
5           c ← p_{k mod n}                           {c is the current coordinator}
            send (R, k, x) to all processes
            wait for messages of the form (R, k, *) from n − f processes      {"*" can be 0 or 1}
            if received more than n/2 (R, k, v) with the same v
            then send (P, k, v) to all processes
10          else send (P, k, ?) to all processes
            wait for messages of the form (P, k, *) from n − f processes      {"*" can be 0, 1 or ?}
            if received at least f + 1 (P, k, v) with the same v ≠ ? then decide v
            if at least one (P, k, v) with v ≠ ? then x ← v else x ← ?
            send (S, k, x) to c
15          if p = c then
                wait for messages of the form (S, k, *) from n − f processes
                if received at least one (S, k, v) with v ≠ ?
                then send (E, k, v) to all processes
                else
20                  random_bit ← R-oracle                                 {query R-oracle}
                    send (E, k, random_bit) to all processes
            wait until receive (E, k, v_coord) from c or c ∈ FD-oracle    {query FD-oracle}
            if received (E, k, v_coord)
            then x ← v_coord
25          else if x = ? then x ← R-oracle                               {query R-oracle}
```

FIG. 4.1. *Hybrid consensus algorithm.*

is the coordinator. The algorithm works under the assumption that a majority of processes are correct (i.e., $n > 2f$). It is easy to see that this requirement is necessary for any algorithm that solves consensus in asynchronous systems with crash failures, even if all processes have access to a random R-oracle and an FD-oracle that belongs to $\diamond S$.

In the hybrid algorithm, every message contains a tag (R, P, S, or E), a phase number, and a value which is either 0 or 1 (for messages tagged P or S, it could also be "?"). Messages tagged R are called *reports*; those tagged with P are called *proposals*; those with tag S are called *suggestions (to the coordinator)*; and those with tag E are called *estimates (from the coordinator)*. When p sends (R, k, v), (P, k, v), or (S, k, v), we say that p *reports*, *proposes*, or *suggests* v in phase k, respectively. When the coordinator sends (E, k, v), we say that the coordinator sends estimate v in phase k.

Each execution of the **while** loop is called a *phase*, and each phase consists of four asynchronous rounds. In the first round (lines 4 to 7), processes report to each other their current estimate (0 or 1) for a decision value.

In the second round (lines 8 to 13), if a process receives a majority of reports for the *same* value then it proposes that value to all processes, otherwise it proposes "?". Note that it is impossible for one process to propose 0 and another process to propose 1 in the same phase. At the end of the second round, if a process receives

$f + 1$ proposals for the same value different than ?, then it decides that value. If it receives at least one value different than ?, then it adopts that value as its new estimate, otherwise it adopts ? for an estimate.

In the third round (lines 14 to 16), processes suggest their estimates to the current coordinator.

In the fourth round (lines 17 to 25), if the coordinator receives a value different than ?, then it sends that value as its estimate. Otherwise, the coordinator queries the R-oracle and sends the random value that it obtains as its estimate. Processes wait until they receive the coordinator's estimate or until their FD-oracle suspects the coordinator. If a process receives the coordinator's estimate, it adopts it. Otherwise, if its current estimate is ?, it adopts a random value obtained from its R-oracle.

To simplify the presentation, the algorithm in Figure 4.1 does not include a halt statement. Moreover, once a correct process decides a value, it will keep deciding the same value in all subsequent phases. However, it is easy to modify the algorithm so that every process decides at most once and halts at most one round after deciding.

5. Proof of correctness. The hybrid algorithm shown in Figure 4.1 always satisfies the safety properties of consensus. This holds no matter how the FD-oracle or the R-oracle behaves, that is, even if these oracles are totally under the control of the adversary. On the other hand, the algorithm satisfies liveness properties only if the FD-oracle satisfies strong completeness. Strong completeness is easy to achieve in practice: most failure detectors use time-out mechanisms, and every process that crashes eventually causes a time-out and, therefore, a permanent suspicion.

Assume that there is a majority of correct processes (i.e., $n > 2f$). We show the following theorem.

THEOREM 5.1.
(Safety) The hybrid algorithm always satisfies uniform validity and uniform agreement.
(Liveness) Suppose that the FD-oracle *satisfies strong completeness.*
- *If the* FD-oracle *satisfies eventual weak accuracy, i.e., it is in* $\diamond S$*, then the algorithm satisfies termination.*
- *If the* R-oracle *is random, then the algorithm satisfies termination with probability* 1.

Proof. We say that *process p starts phase k* if process p completes at least $k - 1$ iterations of the **while** loop. We say that *process p reaches line n in phase k* if process p starts phase k and p executes past line $n - 1$ in that phase. We say that v *is k-locked* if every process that starts phase k does so with its variable x set to v. When ambiguities may arise, a local variable of a process p is subscripted by p; e.g., x_p is the local variable x of process p. □

We first show the safety properties.

LEMMA 5.2. *Suppose* $k > 0$. *Then* (1) *it is impossible for a process to propose* 0 *and another one to propose* 1 *in the same phase k; and* (2) *it is impossible for a process to suggest* 0 *and another to suggest* 1 *in the same phase k.*

Proof. We prove (1) by contradiction: suppose that two processes p and q propose 0 and 1, respectively, in phase k. Thus, p received more than $n/2$ reports for 0 and q received more than $n/2$ reports for 1 in phase k. But then there is a process that reports 0 to p and 1 to q in phase k, and this is impossible. This proves (1).

Now (2) follows from (1) since if a process suggests $v \neq ?$ in phase k, then v was proposed in phase k. □

LEMMA 5.3. *If some process decides v in phase $k > 0$, then v is $(k+1)$-locked.*

Proof. Suppose some process p decides v in phase $k > 0$ (note that $v \neq ?$), and let q be any process that starts phase $k+1$. From the algorithm, p receives at least $f+1$ proposals for v in phase k (line 12). Let r be any process that suggests a value in line 14 of phase k. Before suggesting (line 14), r waits for $n-f$ proposals in line 11. Because p receives $f+1$ proposals for v, r must have received at least one proposal for v. Moreover, by Lemma 5.2, r does not receive any proposals for $\bar{v}$.[4] So r sets x_r to v in line 13 and suggests v in phase k. Thus, (1) q sets x_q to v in line 13, and (2) the coordinator of phase k can only receive suggestions for v. In particular, the coordinator does not receive ?. So, if the coordinator sends an estimate in phase k (line 18), that estimate is also v. If q receives that estimate (line 22), then q resets x_q to v in line 24. Otherwise q does not modify x_q (because x_q is different than ?). In either case, q starts phase $k+1$ with $x_q = v$. □

LEMMA 5.4. *If a value v is k-locked for some $k > 0$, then every process that reaches line* 13 *in phase k decides v in phase k.*

Proof. Suppose v is k-locked for some $k > 0$. Then all reports sent in line 6 of phase k are for v. Since $n-f > n/2$, every process that proposes some value in phase k proposes v in line 9. Consider a process p that reaches line 13 in phase k. Clearly, p receives $n-f$ proposals (line 11) for v in phase k. Since $n-f \geq f+1$, p decides v in phase k. □

COROLLARY 5.5. *If some process decides v in phase $k > 0$, then every process that reaches line* 13 *in phase $k+1$ decides v in phase $k+1$.*

Proof. The corollary is proved by Lemmas 5.3 and 5.4. □

COROLLARY 5.6. (uniform agreement). *If some processes p and p' decide v and v' in phase $k > 0$ and $k' > 0$, respectively, then $v = v'$.*

Proof. For $k = k'$ the result follows from Lemma 5.2 and the fact that a process can decide a value in a phase only if that value was proposed in the same phase. Assume that $k < k'$. Since p' decides in phase k', then p' reaches line 13 in every phase r, $k < r \leq k'$. Since p decides v in phase k, by Corollary 5.5, p' decides v in phase $k+1 \leq k'$. By additional applications of Corollary 5.5, we conclude that p' decides v in phase k'. Each process can decide at most once per phase, so $v = v'$. □

COROLLARY 5.7. (uniform validity). *If some process p decides v, then v is the initial value of some process.*

Proof. Note $v \in \{0, 1\}$. If the initial values of all processes are not identical, then v is clearly the initial value of some process. Now, suppose all processes have the same initial value w. Thus, w is 1-locked. From Lemma 5.4, p decides w, and from Corollary 5.6, $w = v$. □

From now on we assume that the FD-oracle satisfies strong completeness, and proceed to prove the liveness properties.

LEMMA 5.8. *Every correct process starts every phase $k > 0$.*

Proof. The detailed proof is by a simple but tedious induction on k. We describe only the central idea here. In each phase, there are four **wait** statements that can potentially block processes (lines 7, 11, 16, 22). It is not possible for a correct process to be blocked forever in any of the first three **wait** statements because at least $n-f$ processes are correct and send the messages that this process is waiting for. Consider the fourth **wait** statement. Either the coordinator c sends its estimate to all processes or c crashes. In the first case, every correct process receives this estimate. In the

[4] We denote by $\bar{v}$ the binary complement of bit v.

function FavorableToss(r, u): **bit** {evaluated only at time $u \geq \tau_k$ where $k = 2r$}

$k \leftarrow 2r$ {k is the first phase in epoch r}

if some value $v \in \{0, 1\}$ is k-major at time τ_k **then return** v

if by time u no process received $n - f$ proposals in phase $k + 1$ **then return** 0 {$u < \tau_{k+1}$}

if before time τ_{k+1}: {here $u \geq \tau_{k+1}$}
(a) 1 is k-major, and
(b) less than $n/2$ processes R-got a value in phase k, and
(c) the coordinator did not query the R-oracle in line 20 of phase k
then return 1
else return 0

FIG. 5.1. *Favorable coin toss algorithm.*

second case, c eventually appears on the list of suspects; i.e., $c \in$ *FD-oracle* (because the *FD-oracle* satisfies strong completeness). So no correct process waits forever at the fourth **wait** statement of a phase. □

COROLLARY 5.9. *If a value v is k-locked for some $k > 0$, then every correct process decides v in phase k.*

Proof. The corollary is proved immediately from Lemmas 5.4 and 5.8. □

COROLLARY 5.10. *If some process decides v in phase $k > 0$, then every correct process decides v in phase $k + 1$ (and thus in all subsequent phases).*

Proof. The corollary is proved immediately from Corollary 5.5 and Lemma 5.8. □

LEMMA 5.11 (termination). *If the* FD-oracle *satisfies eventual weak accuracy, then every correct process decides.*

Proof. If the FD-oracle satisfies eventual weak accuracy, then there is a time t_0 after which (1) some correct process p_m is never suspected by any correct process and (2) only correct processes take steps (faulty ones crash before t_0). Let k_i be the value of variable k of process p_i at time t_0. Let $\hat{k}$ be the smallest phase after $\max_i\{k_i\}$ such that p_m is the coordinator of phase $\hat{k}$. Let q and r be arbitrary processes that start phase $\hat{k} + 1$. Note that this occurs after time t_0, so neither q nor r suspect the coordinator p_m in phase $\hat{k}$. Thus, q and r set x_q and x_r to p_m's estimate in line 24. Since this estimate is different from ? and unique for phase $\hat{k}$, we have $x_q = x_r = v$ for some $v \neq ?$ at the beginning of phase $\hat{k} + 1$. So v is $(\hat{k} + 1)$-locked. Therefore, by Corollary 5.9, all correct processes decide v in phase $\hat{k} + 1$. □

We now proceed to show that if the R-oracle is random, then the algorithm satisfies termination with probability 1. For $k > 0$, let τ_k be the first time that any process receives $n - f$ proposals in phase k. From Lemma 5.8, for every $k > 0$, some process receives $n - f$ proposals in phase k, so τ_k is well defined. Note that in our algorithm no process queries the R-oracle in phase k before time τ_k.

A process starts a phase with its variable x set to either 0 or 1 (never to ?). For each $k > 0$, we say that a value $v \in \{0, 1\}$ *is k-major at time t* if by time t more than $n/2$ processes have started phase k with their variable x set to v. Clearly, for each $k > 0$ and all times t and t', it is impossible for 0 to be k-major at t and 1 to be k-major at t'.

We say that *a process p R-gets v in phase k at time t* if either

1. in phase k at time t, p obtains v from the R-oracle in line 25 and sets x_p to v; or

2. in phase k, the coordinator obtains v from the R-oracle in line 20 and sends v as its estimate to all processes, and p receives this estimate and sets x_p to v in line 24 at time t.

Intuitively, a process p R-gets v if p sets x_p to v, and p obtained v from an R-oracle query (directly, or indirectly through the coordinator).

LEMMA 5.12. *For every $k \geq 1$, if at time t a process p starts phase $k+1$ with x_p set to some value $v \in \{0,1\}$, then v is k-major at time t or p R-gets v in phase k.*

Proof. Consider phase k. Suppose p did not R-get v. Let t' be the last time p updates x_p in phase k. Note that $t' < t$. Then, at time t', either (a) p receives the estimate from the coordinator and the coordinator obtained that estimate from one of its non-? suggestions; or (b) p sets x_p in line 13. In both cases, more than $n/2$ processes must have reported v in phase k before time t'. Therefore, more than $n/2$ processes have started phase k by time t' (and thus by time t) with their variable x set to v. □

An immediate consequence of Lemma 5.12 is that for every $k \geq 1$, if v is never k-major and no process R-gets v in phase k, then $\bar{v}$ is $(k+1)$-locked.

For the rest of the proof, we group pairs of phases into *epochs* as follows: *epoch* r consists of phases $2r$ and $2r+1$.[5] We will define the concept of a "lucky" epoch—one in which processes toss coins that cause the termination of the algorithm (no matter what the adversary does). To do so, we first define function *FavorableToss*(r, u), given in Figure 5.1. We say that *epoch r is lucky* if, for every process p and any time u, if p queries the R-oracle in epoch r at time u, then p obtains *FavorableToss*(r, u) from the R-oracle. Note that if p queries the R-oracle in epoch r at time u, this occurs after at least one process receives $n - f$ proposals in phase $2r$. Thus, $\tau_{2r} \leq u$, so the value of *FavorableToss*(r, u) depends only on what occurred in the system up to time u.

LEMMA 5.13. *If the R-oracle is random, then the probability that some epoch is lucky is* 1.

Proof. The result is immediate from the following observation: for every $r \geq 1$, (a) the probability that epoch r is lucky is at least $2^{-(2n+2)}$ (because in each phase there are at most $n+1$ queries to the R-oracle, and the R-oracle is random), and (b) for any $r' \neq r$, the events "epoch r is lucky" and "epoch r' is lucky" are independent (because epochs r and r' consist of disjoint sets of phases). □

LEMMA 5.14. *For every $r \geq 1$, if epoch r is lucky, then some value is $(2r+1)$-locked or $(2r+2)$-locked.*

Proof. Throughout the proof of this lemma, fix some arbitrary $r \geq 1$ and assume that epoch r is lucky. Let $k = 2r$; recall that epoch r consists of phases k and $k+1$. Since epoch r is lucky, if any process R-gets a value v at time t and in phase $j = k$ or $j = k+1$, then $v =$ *FavorableToss*(r, u) for some time u, $\tau_j \leq u \leq t$ (value v was obtained either directly from the R-oracle or indirectly through the coordinator).

Case 1. Suppose some value v is k-major at time τ_k. By the definition of *FavorableToss*, for any u such that $\tau_k \leq u$, *FavorableToss*$(r, u) = v$. So, if a process R-gets a value in phase k, that value is v. Note that $\bar{v}$ is not k-major at any time. By Lemma 5.12, v is $(k+1)$-locked.

Case 2. Now assume that no value is k-major at time τ_k.

Case 2.1. Suppose that no value is k-major before time τ_{k+1}. Then for any u, $\tau_k \leq u$, we have *FavorableToss*$(r, u) = 0$. By Lemma 5.12, every process p that starts phase $k+1$ before time τ_{k+1} does so with x_p set to some value that p R-got in phase k, and such value can only be 0. So all reports (and thus all proposals) sent in phase $k+1$

[5] Phase 1 is not part of any epoch.

before time τ_{k+1} are for 0. From the definition of τ_{k+1}, there are at least $n-f$ such proposals for 0 in phase $k+1$. By an argument similar to the one in the proof of Lemma 5.3, value 0 is $(k+2)$-locked.

Case 2.2. Now assume some value v is k-major before time τ_{k+1}.

Case 2.2.1. Suppose $v=0$. Since 1 is never k-major, then for any time u such that $\tau_k \le u$, we have $\mathit{FavorableToss}(r,u) = 0$. So all processes that R-get a value in phase k R-get 0. By Lemma 5.12, value 0 is $(k+1)$-locked.

Case 2.2.2. Now assume $v=1$. For any time u, $\tau_k \le u < \tau_{k+1}$, we have $\mathit{FavorableToss}(r,u)=0$. Let S be the processes that R-get a value in phase k before time τ_{k+1}; clearly, all processes in S R-get 0.

Case 2.2.2.1. Suppose $|S| \ge n/2$. Then, for any time u, $\tau_k \le u$, $\mathit{FavorableToss}(r,u)=0$. So, all processes that R-get in phase $k+1$ R-get 0. Note that $|S| \ge n/2$ implies that 1 can never be $(k+1)$-major. By Lemma 5.12, value 0 is $(k+2)$-locked.

Case 2.2.2.2. Now assume that $|S| < n/2$.

Case 2.2.2.2.1. Suppose that the coordinator of phase k does not query the R-oracle in line 20 of phase k before time τ_{k+1}. Then, for any u such that $\tau_{k+1} \le u$, we have $\mathit{FavorableToss}(r,u)=1$. So, if the coordinator queries the R-oracle in line 20 of phase k, it obtains 1 from the R-oracle. Therefore, all processes that R-get a value at or after time τ_{k+1} in phase k R-get 1. Thus, exactly $|S| < n/2$ processes R-get 0 in phase k. Since 1 is k-major, from Lemma 5.12 we conclude that value 0 can never be $(k+1)$-major. Since no process queries the R-oracle in phase $k+1$ before time τ_{k+1}, all processes that R-get a value in phase $k+1$ R-get 1. By Lemma 5.12, value 1 is $(k+2)$-locked.

Case 2.2.2.2.2. Now assume that the coordinator of phase k queries the R-oracle in line 20 of phase k before time τ_{k+1}. Then the coordinator obtains 0 from the R-oracle. So, for any $u \ge \tau_k$, we have $\mathit{FavorableToss}(r,u)=0$. Since the coordinator queries the R-oracle in line 20, it received $n-f$ suggestions for ? in line 16, and this occurred before time τ_{k+1}. Thus, $n-f$ processes have set their variable x to ? in line 13 in phase k before time τ_{k+1}. Note that if any such process starts phase $k+1$, then it R-gets a value in phase k and that value is 0, and thus, any such process starts phase $k+1$ with its variable x set to 0. Therefore, at most $n-(n-f)=f<n/2$ processes start phase $k+1$ with their variable x set to 1. So 1 can never be $(k+1)$-major. All processes that R-get in phase $k+1$ R-get 0. By Lemma 5.12, value 0 is $(k+2)$-locked. □

LEMMA 5.15 (termination with probability 1). *If the R-oracle is random, then the probability that all correct processes decide is* 1.

Proof. The lemma is proved immediately from Lemmas 5.13 and 5.14 and Corollary 5.9. □

The proof of Theorem 5.1 is now complete: Uniform validity and uniform agreement were shown in Corollaries 5.7 and 5.6, respectively. Termination was proved in Lemma 5.11, and termination with probability 1 was shown in Lemma 5.15. □

From the proof of Lemma 5.13, it is easy to see that the expected number of rounds for termination is $O(2^{2n})$. However, it can be shown that, as in [4], termination is reached in constant expected number of rounds if $f = O(\sqrt{n})$. In section 7, we outline a similar hybrid algorithm that terminates in constant expected number of rounds, even for $f = O(n)$.

$c \leftarrow p_0$ {p_0 is the first coordinator}
if $p = c$ **then send** $(E, 0, v_p)$ to all processes {if p is the first coordinator}

wait until receive $(E, 0, v\_coord)$ **from** c **or** $c \in$*FD-oracle* {query FD-oracle}
if received $(E, 0, v\_coord)$
then send $(P, 0, v\_coord)$ **to** all processes
else send $(P, 0, ?)$ **to** all processes

wait for messages of the form $(P, 0, *)$ **from** $n - f$ processes {"$*$" can be 0, 1 or ?}
if received at least $f + 1$ $(P, 0, v)$ with the same $v \neq ?$ **then** decide v
if received at least one $(P, 0, v)$ with $v \neq ?$ **then** $x \leftarrow v$

FIG. 6.1. *Optimization for the hybrid algorithm.*

6. An optimization. The algorithm in Figure 4.1 was designed to be simple rather than efficient, because our main goal here is to demonstrate the viability of a "robust" hybrid approach (one in which termination can occur in more than one way: by "good" failure detection or by "good" random draws). The following optimization suggests that such hybrid algorithms can also be efficient in practice.

In many systems, failures are rare, and failure detectors can be tuned to seldom make mistakes (i.e., erroneous suspicions). The algorithm in Figure 4.1 can be optimized to perform particularly well in such systems. The optimized version ensures that all correct processes decide by the end of two asynchronous rounds when the first coordinator does not crash and no process erroneously suspects it.[6]

This optimization is obtained by inserting some extra code between lines 2 and 3 of the hybrid algorithm. This code, given in Figure 6.1, consists of a phase (phase 0) with two asynchronous rounds. In the first round, p_0 sends a message to all processes; in the second round, every process sends a message to all processes. We claim that (1) the optimization code preserves the correctness of the original algorithm; and (2) processes decide quickly in the absence of failures and erroneous suspicions. To see (1), note the following:

1. No correct process blocks during the execution of the optimization code (phase 0); i.e., all correct processes start phase 1.

2. Any process p that starts phase 1 does so with x_p set to the initial value of some process.

3. If some process decides v in phase 0, then v is 1-locked. Thus, (by Corollary 5.9) all correct processes decide v in phase 1.

To see (2), note that if p_0 is correct and no process suspects p_0, then all processes wait for its estimate v and propose v in phase 0; so every process receives $n - f$ proposals for v and, thus, decides v in phase 0. Thus, we have the following theorem.

THEOREM 6.1. *Theorem* 5.1 *holds for the optimized hybrid algorithm. Moreover, in executions with no crashes or false suspicions, all processes decide in two rounds.*

7. Discussion. In practice, many systems are well behaved most of the time: few failures actually occur, and most messages are received within some predictable time. Failure detector based algorithms (whether "pure" ones like in [7] or hybrid ones like in this paper) are particularly well suited to take advantage of this: (time-out based) failure detectors can be tuned so that the algorithms perform optimally

[6] Actually, decision occurs in two rounds even if up to $n - 2f - 1$ processes erroneously suspect it.

when the system behaves as predicted, and performance degrades gracefully as the system deviates from its "normal" behavior (i.e., if failures occur or messages take longer than expected). For example, the optimized version of our hybrid algorithm solves consensus in only two asynchronous rounds in the executions that are most likely to occur in practice, namely, runs with no failures or erroneous suspicions.

The above discussion suggests that using this hybrid approach is better than using the randomized approach alone. In fact, randomized consensus algorithms for asynchronous systems tend to be inefficient in practical settings.[7] Typically, their performance depends more on "luck" (e.g., many processes happen to start with the same initial value or happen to draw the same random bit) than on how "well behaved" the underlying system is (e.g., on the number of failures that actually occur during execution). The fact that randomized algorithms are extremely "robust," i.e., they do not depend on how the system behaves, may also be an inherent source of inefficiency.

Note that our hybrid algorithm terminates with probability 1 even if the FD-oracle is completely inaccurate (in fact, even if every process suspects every other process all the time). So it is more robust than algorithms that are simply failure detector based.

An important remark is now in order about the expected termination time of our hybrid algorithm. We developed this algorithm by combining Ben-Or's randomized algorithm [4] with the failure detection ideas in [7]. We selected Ben-Or's algorithm because it is the simplest and thus the most appropriate to illustrate this approach, even though its expected number of rounds is exponential in n for $f = O(n)$. By starting from an efficient randomized algorithm, due to Chor, Merritt, and Shmoys [9], we can obtain a hybrid algorithm that terminates in constant expected number of rounds, as we now briefly explain.

Roughly speaking, the randomized asynchronous consensus algorithm in [9] is obtained from Ben-Or's algorithm by replacing each coin toss with the toss of a "weakly global coin" computed by a *coin_toss* procedure. We can do exactly the same: replace the coin tosses of the algorithm in Figure 4.1 with those obtained by using the *coin_toss* procedure. More precisely, in each phase, every process (a) invokes this procedure between the second and third rounds (i.e., between lines 13 and 14) to obtain a random bit, and (b) uses this random bit rather than querying the R-oracle (in lines 20 and 25).[8]

As in [9], this modified hybrid algorithm terminates[9] in constant expected number of rounds for $f \leq n\ (3 - \sqrt{5})\ /\ 2 \approx 0.38n$. But also as in [9], and in contrast to the algorithm in section 4, it assumes that the adversary cannot see the internal state of processes or the content of messages. With the optimization of Figure 6.1, this modified hybrid algorithm also terminates in two rounds in failure-free and suspicion-free runs.

8. Related work. The idea of combining randomization with a deterministic consensus algorithm appeared in [15] and was further developed in [25]. These works, however, assume that the system is *synchronous* and do not use failure detectors.

Dolev and Malki were the first to combine randomization and unreliable failure

[7] Algorithms that assume that processes a priori agree on a long sequence of random bits [22, 24] are more efficient than others. But this assumption may be too strong for some systems.

[8] As in [9], another simple modification is necessary: the addition of a "synchronization round" just before the *coin_toss* procedure. In this round, processes broadcast "wait" messages, then wait until $n - f$ such messages are received.

[9] Provided, of course, that the FD-oracle satisfies strong completeness.

detection to solve consensus in asynchronous systems with process crashes [12]. That work differs from ours in many respects:

1. In contrast to our algorithm, those in [12] require that both R-oracle and FD-oracle always work correctly.

2. In our hybrid algorithm, safety is always preserved; even if the failure detector continuously misbehaves, no two processes ever decide differently. In contrast, with the hybrid algorithms given in [12], if at any point the failure detector loses its accuracy property, processes may decide differently.

3. Our goal is to use randomization to improve failure detector based algorithms. We use randomization as a "back-up" to ensure termination in the occasional "bad" periods when the failure detector loses its accuracy property.

Two goals of [12] are to use failure detection to increase the resiliency of randomized consensus algorithms, and to ensure their deterministic termination. The hybrid consensus algorithms given in [12] achieve the first goal by increasing the resiliency from $f < n/2$ to $f < n$, but not the second one. It is stated, however, that a future version of the paper will give an algorithm that achieves both goals.

4. The two hybrid algorithms in [12] use failure detectors that are stronger than $\diamond S$ (the failure detector that we use). The first algorithm—which supposes that the same sequence of random bits is shared by all the processes, as in [22]—assumes that some correct process is *never* suspected by any process. The second algorithm—which drops the assumption of a common sequence of bits—assumes that $\Omega(n)$ correct processes are never suspected by any process. Both algorithms reach consensus in constant expected time.

Acknowledgments. We are grateful to Vassos Hadzilacos; some of our proofs are based on his lecture notes. We would also like to thank the anonymous referees for their valuable comments.

REFERENCES

[1] M. K. Aguilera, W. Chen, and S. Toueg, *Heartbeat: A timeout-free failure detector for quiescent reliable communication*, in Proc. of the 11th International Workshop on Distributed Algorithms, Lecture Notes in Comput. Sci., Springer-Verlag, New York, 1997, pp. 126–140. A full version is also available as Tech. report 97-1631, Department of Computer Science, Cornell University, Ithaca, NY, 1997.

[2] M. K. Aguilera, W. Chen, and S. Toueg, *Quiescent Reliable Communication and Quiescent Consensus in Partitionable Networks*, Tech. report 97-1632, Department of Computer Science, Cornell University, Ithaca, NY, 1997; revised version to appear in Theoret. Comput. Sci. as *Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks.*

[3] O. Babaoğlu, R. Davoli, and A. Montresor, *Failure Detectors, Group Membership and View-synchronous Communication in Partitionable Asynchronous Systems (Preliminary Version)*, Tech. report UBLCS-95-18, Department of Computer Science, University of Bologna, Bologna, Italy, 1995.

[4] M. Ben-Or, *Another advantage of free choice: Completely asynchronous agreement protocols*, in Proc. of the Second ACM Symposium on Principles of Distributed Computing, ACM, New York, 1983, pp. 27–30.

[5] G. Bracha and S. Toueg, *Resilient consensus protocols*, in Proc. of the Second ACM Symposium on Principles of Distributed Computing, ACM, New York, 1983, pp. 12–26. An extended and revised version appeared as *Asynchronous consensus and broadcast protocols*, J. Assoc. Comput. Mach., 32 (1985), pp. 824–840.

[6] T. D. Chandra, V. Hadzilacos, and S. Toueg, *The weakest failure detector for solving consensus*, J. Assoc. Comput. Mach., 43 (1996), pp. 685–722.

[7] T. D. Chandra and S. Toueg, *Unreliable failure detectors for reliable distributed systems*, J. Assoc. Comput. Mach., 43 (1996), pp. 225–267.

[8] B. Chor and C. Dwork, *Randomization in Byzantine agreement*, Adv. Comput. Res., 4 (1989), pp. 443–497.

[9] B. Chor, M. Merritt, and D. B. Shmoys, *Simple constant-time consensus protocols in realistic failure models*, J. Assoc. Comput. Mach., 36 (1989), pp. 591–614.

[10] D. Dolev, C. Dwork, and L. Stockmeyer, *On the minimal synchronism needed for distributed consensus*, J. Assoc. Comput. Mach., 34 (1987), pp. 77–97.

[11] D. Dolev, R. Friedman, I. Keidar, and D. Malkhi, *Failure Detectors in Omission Failure Environments*, Tech. report TR96-1608, Department of Computer Science, Cornell University, Ithaca, NY, 1996.

[12] D. Dolev and D. Malki, *Consensus Made Practical*, Tech. report CS94-7, The Hebrew University of Jerusalem, 1994.

[13] C. Dwork, N. A. Lynch, and L. Stockmeyer, *Consensus in the presence of partial synchrony*, J. Assoc. Comput. Mach., 35 (1988), pp. 288–323.

[14] M. J. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. Assoc. Comput. Mach., 32 (1985), pp. 374–382.

[15] O. Goldreich and E. Petrank, *The best of both worlds: Guaranteeing termination in fast randomized Byzantine agreement protocols*, Inform. Process. Lett., 36 (1990), pp. 45–49.

[16] R. Guerraoui and A. Schiper, *Non blocking atomic commitment with an unreliable failure detector*, in Proc. of the 14th IEEE Symposium on Reliable Distributed Systems, Bad Neuenahr, Germany, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 41–50.

[17] R. Guerraoui and A. Schiper, *Consensus service: A modular approach for building agreement protocols in distributed systems*, in Proc. of the 26th IEEE International Symposium on Fault-Tolerant Computing, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 168–177.

[18] M. Hurfin, A. Mostefaoui, and M. Raynal, *Consensus in Asynchronous Systems Where Processes Can Crash and Recover*, Tech. report 1144, Institut de Recherche en Informatique et Systèmes Aléatoires, Université de Rennes, 1997; Proc. 17th IEEE Symposium on Reliable Distributed Systems (SRDS '98), IEEE Computer Society Press, Washington, DC, to appear.

[19] W.-K. Lo and V. Hadzilacos, *Using failure detectors to solve consensus in asynchronous shared-memory systems*, in Proc. of the Eighth International Workshop on Distributed Algorithms, Springer-Verlag, New York, 1994, pp. 284–295.

[20] D. Malkhi and M. Reiter, *Unreliable intrusion detection in distributed computations*, in Proc. of the 10th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 116–124.

[21] R. Oliveira, R. Guerraoui, and A. Schiper, *Consensus in the Crash-Recover Model*, Tech. report 97-239, Département d'Informatique, Ecole Polytechnique Fédérale, Lausanne, Switzerland, 1997.

[22] M. Rabin, *Randomized Byzantine generals*, in Proc. of the 24th Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1983, pp. 403–409.

[23] A. Schiper, *Early consensus in an asynchronous system with a weak failure detector*, Distrib. Comput., 10 (1997), pp. 149–157.

[24] S. Toueg, *Randomized Byzantine agreements*, in Proc. of the Third ACM Symposium on Principles of Distributed Computing, ACM, New York, 1984, pp. 163–178.

[25] A. Zamsky, *A randomized Byzantine agreement protocol with constant expected time and guaranteed termination in optimal (deterministic) time*, in Proc. of the Fifteenth ACM Symposium on Principles of Distributed Computing, ACM, New York, 1996, pp. 201–208.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 904–934

AVERAGE PROFILE OF THE GENERALIZED DIGITAL SEARCH TREE AND THE GENERALIZED LEMPEL–ZIV ALGORITHM*

GUY LOUCHARD†, WOJCIECH SZPANKOWSKI‡, AND JING TANG§

Abstract. The goal of this research is threefold: (i) to analyze generalized digital search trees, (ii) to derive the average profile (i.e., phrase length) of a generalization of the well-known parsing algorithm due to Lempel and Ziv, and (iii) to provide analytic tools to analyze asymptotically certain partial differential functional equations often arising in the analysis of digital trees. In the generalized Lempel–Ziv parsing scheme, one partitions a sequence of symbols from a finite alphabet into phrases such that the new phrase is the shortest substring seen in the past by at most $b-1$ phrases ($b=1$ corresponds to the original Lempel–Ziv scheme). Such a scheme can be analyzed through a generalized digital search tree in which every node is capable of storing up to b strings. In this paper, we investigate the depth of a randomly selected node in such a tree and the length of a randomly selected phrase in the generalized Lempel–Ziv scheme. These findings and some recent results allow us to compute the average redundancy of the generalized Lempel–Ziv code and compare it to the ordinary Lempel–Ziv code, leading to an optimal value of b. Analytic techniques of (precise) analyses of algorithms are used to establish most of these conclusions.

Key words. generalized Lempel–Ziv parsing scheme, generalized digital search trees, average redundancy, partial differential functional equations, singularity analysis, asymptotic expansions, depoissonization, Mellin transform

AMS subject classifications. 68Q25, 68P05

PII. S0097539796301811

1. Introduction.

The heart of several universal data compression schemes is the parsing algorithm due to Lempel and Ziv [23] (e.g., it is used in the UNIX `compress` command and in a CCITT standard for data compression for modems). It is a dictionary-based algorithm that partitions a sequence into phrases (blocks) of variable sizes such that a new block is the shortest substring not seen in the past as a phrase. Such a new phrase is coded by giving the location of the prefix (that occurred before as a phrase) and the value of the last symbol; that is, the Lempel–Ziv code consists of pairs $(\texttt{pointer}, \texttt{symbol})$ (details can be found in many textbooks, e.g., [3, 36]). For example, the sequence `ababbababaaaaaaaaac` is parsed into (`a`)(`b`)(`ab`)(`ba`)(`bab`)(`aa`)(`aaa`)(`aaaa`)(`c`), and its code becomes `0a0b1b2a4b1a6a7a0c` (e.g., the pair `6a` indicates that this phrase (`aaa`) consists of the sixth phrase as a prefix that occurred before, appended by `a`). Observe that there is no need for a separator between phrases. Let us compute the length of this code in bits, assuming a ternary alphabet $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$. There are nine phrases; thus we need up to four $(= \lceil \log_2 9 \rceil)$ bits to code each pointer. Every terminal symbol requires two $(= \lceil \log_2 3 \rceil)$ bits and there are nine phrases in the code; hence the total length of the code is $9 \cdot 4 + 9 \cdot 2 = 54$ bits.

*Received by the editors April 2, 1996; accepted for publication (in revised form) May 7, 1997; published electronically January 29, 1999. This research was partially supported by NSF grant NCR-9206315 and NATO collaborative grant CRG.950060.

http://www.siam.org/journals/sicomp/28-3/30181.html

†Département d'Informatique, Université Libre de Bruxelles, B-1050 Brussels, Belgium (louchard@ulb.ac.be).

‡Department of Computer Science, Purdue University, W. Lafayette, IN 47907 (spa@cs.purdue.edu). The research of this author was additionally supported by NSF grants NCR-9415491, CCR-9201078, and C-CR-9804760.

§Microsoft Co., One Microsoft Way, 1/2061 Redmond, WA 98052 (gcneta@microsoft.com).

It is known that the original Lempel–Ziv scheme does not cope very well with some sequences; e.g., those containing a long string of repeated symbols. To somewhat remedy this situation, Louchard and Szpankowski [27] introduced a generalization of the Lempel–Ziv parsing scheme: It parses a sequence into phrases such that the next phrase is the shortest phrase seen in the past by *at most* $b-1$ phrases ($b=1$ corresponds to the original Lempel–Ziv algorithm). For example, the sequence above is parsed with $b=2$ as follows: (a)(b)(a)(b)(ba)(ba)(baa)(aa)(aa)(aaa)(c), which has seven *distinct* phrases and eleven phrases. The code for this new algorithm consists of either (i) (pointer, symbol) when pointer refers to the *first* previous occurrence of the prefix of the phrase and symbol is the value of the last symbol of this phrase or (ii) just (pointer) if the phrase has occurred as a whole previously (i.e., it is the second, third, ..., or bth occurrence of this phrase). For example, the code for the previously parsed sequence becomes, for $b=2$, 0a0b122a33a1a55a0c (e.g., the phrase 2a occurs for the first time as a new phrase hence 2 refers to the second distinct phrase appended by a, while code 5 represents a phrase that has its second occurrence as the fifth distinct phrase). Observe that this code is of length 47 bits since there are 11 phrases, each requiring up to $\lceil \log_2 7 \rceil = 3$ bits, and seven symbols needing 14 additional bits (i.e., $47 = 11 \cdot 3 + 7 \cdot 2 = 47$). We saved 7 bits! But the reader may verify that the same sequence requires only 46 bits for $b=3$ (so only 1 additional bit is saved), while for $b=4$ the bit count increases again to 52. This example suggests that $b=3$ is (at least local) optimum for the above sequence. Can one draw similar conclusions "on average" for a typical sequence (i.e., generated randomly)? We shall provide an answer to this and other questions in this paper.

Our goal is to investigate the probabilistic behavior of a *typical phrase length*, that is, the length of a randomly selected phrase. As already observed in Louchard and Szpankowski [26] (cf. [15]), the Lempel–Ziv algorithm can be modeled in two ways, namely, as a *digital tree model* or a *Lempel–Ziv model*. In the former, one constructs the Lempel–Ziv sequence from m (probabilistically) independent strings (of possibly infinite lengths). For example, let $m=4$ sequences be given: $X_1 = 0000\ldots$, $X_2 = 1010\ldots$, $X_3 = 1111\ldots$, and $X_4 = 0101\ldots$. Then, for $b=1$ the Lempel–Ziv sequence (0)(1)(11)(01) is of length $L_4 = 6$ and a typical (i.e., randomly selected) phrase is of average length $1\frac{1}{2}$. In the Lempel–Ziv model there is only *one sequence* of fixed length, say, n, and one partitions the sequences according to the Lempel–Ziv algorithm as described above. Clearly, these models are related as already observed in [15, 26]. We shall study both models in this paper.

Let us have a closer look at the *digital tree model* (cf. [26] for a more detailed description). To justify its name we shall show how the Lempel–Ziv parsing (of m strings) can be constructed by building an associated digital search tree (cf. [6, 20, 29]). In this case, we consider an extension of digital search trees called *b-digital search tree*, or (for short) b-DST (cf. [8, 29]) which is built from a fixed number, say, m, of strings. Hereafter, we consider only the binary alphabet $\Sigma = \{0, 1\}$, but an extension to any finite alphabet is straightforward. This tree is constructed as follows: In addition to m given strings, we consider b empty strings that are stored in the root of the tree. The remaining m strings are stored in an available space in a node which is not full, i.e., containing less than b strings. The search for an available space follows the prefix structure of a string. The rule is simple: if the next symbol in a string is 1 we move to the left; otherwise we move to the right until either we find a node with less than b strings or, if all nodes are full on this path, we create a new node. The details of such a construction can be found in [8, 20, 29].

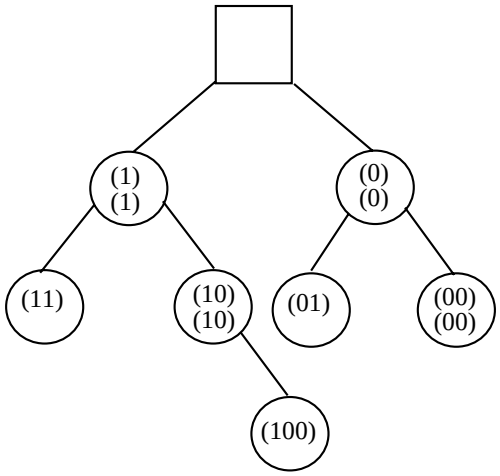


FIG. 1. *A* 2*-digital search tree representation of the generalized Lempel–Ziv parsing for the string* 1100101000100010011.

Now, we briefly discuss the *Lempel–Ziv model*. We recall that now we deal with a single sequence of fixed length n which is partitioned according to the Lempel–Ziv algorithm. As already discussed in [1, 26, 15], digital trees also can be used to construct the Lempel–Ziv parsing. Indeed, let us first append the string with b empty phrases that are stored in the root of the associated b-digital search tree. All other phrases are stored in internal nodes and they are constructed on-line in the course of building the associated b-DST. When a new phrase is created, the search starts at the root and proceeds down the tree as directed by the input symbols of the string exactly in the same manner as in the b-digital tree construction until either we find a node with less than b phrases or we create a new node. In Figure 1 we show the 2-DST constructed from the sequence 1100101000100010011. Observe that for a fixed length string, the number of nonroot nodes in the associated digital tree is a random variable that is equal to the number of distinct phrases of the generalized Lempel–Ziv scheme.

In this paper, we study both models in a probabilistic framework in which every string is generated according to the Bernoulli model; that is, *symbols are generated in an independent manner with* 0 *and* 1 *occurring, respectively, with probability* p *and* $q = 1-p$. If $p = q = 0.5$, the Bernoulli model is *symmetric*; otherwise it is *asymmetric*.

Digital trees appear in a variety of computer and communications applications including searching, sorting, dynamic hashing, codes, conflict resolution protocols for multiaccess communications, and data compression (cf. [6, 8, 20, 29, 26, 27, 15, 39]). Thus, better understanding of their behavior is desirable and could lead to some algorithmic improvements. One parameter that is of interest to these applications is the depth of a randomly (uniformly) selected node (i.e., the length of the path from the root to the chosen node). It can represent the search time for a key word or the length of a phrase in the generalized Lempel–Ziv algorithm (cf. Figure 1). In this paper, we fully characterize the probabilistic behavior of the depth in a b-digital search tree under the digital tree model. We derive asymptotic expansions for the mean and the variance, as well as for large deviations and the limiting distribution of the depth. In particular, we prove that the depth appropriately normalized is asymptotically normally distributed in the asymmetric Bernoulli model.

The *Lempel–Ziv model* is somewhat more intricate since there is some unpleasant dependency between consecutive phrases. Fortunately, Louchard and Szpankowski [26] proved that this dependency is not too strong (cf. (24) in section 2 of this paper), and one can infer the probabilistic behavior of the length of a randomly selected phrase from the depth of a randomly selected node in a b-DST built from a fixed number of nodes (i.e., in the digital tree model). In addition, using another recent finding of Louchard and Szpankowski [28] (concerning redundancy of the ordinary Lempel–Ziv code, i.e., for $b = 1$; cf. Savari [35]), we are able to compute the average redundancy of the generalized Lempel–Ziv code. The average redundancy measures how far the code is from being optimal for a given source of information (thus it requires a precise asymptotic expansion for the average length of the Lempel–Ziv sequence in the digital tree model). This allows us to determine theoretically the optimal value of b that minimizes the average length of the generalized Lempel–Ziv code (cf. Theorem 2.3 and the discussion thereafter).

We believe our contribution is also of a methodological nature: We establish our results in a consistent manner by a technique that belongs to the toolkit of the analytic analysis of algorithms. More precisely, Flajolet and Richmond [8] had already observed that b-digital trees are harder to analyze than ordinary ($b = 1$) digital search trees. The difficulty boils down ultimately to a solution of the following general recurrence in x_n. Let $x_1, \ldots, x_b$ be given. Then, for a given sequence a_n and a constant u,

$$x_{n+b} = a_n + u \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} (x_k + x_{n-k}), \qquad n \geq 0 \tag{1}$$

(cf. recurrence (4) in section 2), which can be reduced to the following partial differential functional equation in terms of the Poisson generating function of x_n defined as $\widetilde{X}(z) = \sum_{n\geq 0} x_n \frac{z^n}{n!} e^{-z}$:

$$\sum_{i=0}^{b} \binom{b}{i} \frac{\partial^i \widetilde{X}(z)}{\partial z^i} = \widetilde{A}(z) + u(\widetilde{X}(pz) + \widetilde{X}(qz)), \tag{2}$$

where $q = 1 - p$ (cf. (6) in section 2 and (30) and(31) in section 3).

The above recurrence can be solved exactly for $b = 1$ (cf. [38]), but attempts at extensions to $b > 1$ have partially failed. Flajolet and Richmond [8] (cf. also [12]) used a new technique to solve this recurrence for $p = 1/2$ (i.e., symmetric Bernoulli model). Unfortunately, this technique seems to be restricted to the symmetric Bernoulli model since some sums involved in the asymmetric Bernoulli model (i.e., $p \neq 1/2$) cease to be harmonic sums. To circumvent this difficulty we devise another approach that is asymptotic in nature. In order to accomplish this, we use some other techniques such as analytical poissonization and depoissonization, singularity analysis, and Mellin transforms.

In passing, we should mention that differential functional equations such as (2) were discussed as early as 1924 by Flamant [10], who provided an iterative solution. Our approach is completely different, and we present an asymptotic solution as $z \to \infty$ (which suffices to obtain an asymptotic solution of the original recurrence). Finally, during the course of the analysis we face a problem of numerical evaluation of some constants involving Mellin transforms. These constants are somewhat important since they carry the information about the dependence of b on the final solution. We propose here a method to evaluate numerically such constants (cf. section 3.3) that is of its

own interest and can be applied to other problems. We should mention that similar numerical problems can be encountered in other analyses (cf. [18]).

Digital search trees for $b = 1$ have been analyzed in the past in the case of a *fixed* number of independent strings (cf. [6, 14, 19, 20, 21, 24, 31, 32, 33, 38, 39]). Much less is known about b-digital search trees. As mentioned, the first nontrivial analysis of the size of such trees for the symmetric Bernoulli model was proposed by Flajolet and Richmond [8]. The variance of the size and the internal path length—still for the symmetric model—was discussed by Hubalek [12]. To the best of our knowledge, b-DST have not yet been analyzed for the asymmetric Bernoulli model. In a companion paper, Louchard [25] presents an alternative probabilistic approach to obtain some of our results. He gets the limiting distribution (without the rate of convergence) but not the large deviation results and precise evaluation of the moments (see section 4.2 for the derivation of the asymptotic distribution in the symmetric case using this approach). In [25] Louchard also evaluates the average number of nodes in a b-digital search tree, thus directly extending the Flajolet and Richmond result [8] to the asymmetric Bernoulli model.

For the original Lempel–Ziv parsing algorithm ($b = 1$) mostly only first-order asymptotics (e.g., leading terms in almost sure convergence) have been analyzed (cf. [41, 22, 23]), with exceptions being the work by Aldous and Shields [1] and recent works of Louchard and Szpankowski [26] and Jacquet and Szpankowski [14] (see also [30, 35, 39]). Finally, Gilbert and Kadota [11] analyzed numerically the number of possible messages composed of m parsed phrases, as well as the length of a phrase in the digital tree model.

The paper is organized as follows: In the next section, we present our main results concerning the digital tree model and the generalized Lempel–Ziv scheme. Proofs are deferred to sections 3 and 4, where in the former we treat the asymmetric Bernoulli model and in the latter the symmetric case. The proofs are analytic with the exception of the distribution in the symmetric Bernoulli model discussed in section 4.2.

2. Main results. We consider a b-digital tree built over m statistically independent words. Let $D_m(i) = D_i(i)$ be the depth of the ith string (of infinite length) in such a tree, that is, the length of a path from the root to a node containing the ith string. In a variety of applications, one is interested in the *typical* depth D_m, defined as the depth of a randomly selected string; that is,

$$\Pr\{D_m = k\} = \frac{1}{m}\sum_{i=1}^{m} P\{D_m(i) = k\}. \tag{3}$$

As argued in Louchard and Szpankowski [26], the depth D_m is closely related to the length of a randomly selected phrase in the generalized Lempel–Ziv parsing scheme. We denote it D_n^{LZ}, where n is the length of the string to be parsed. Our goal is to study moments and distribution of D_m and D_n^{LZ} and their dependence upon parameter b.

2.1. Digital tree model. We now concentrate on the depth D_m in a b-DST built over a *fixed* number, say, m, of independent strings generated according to an asymmetric Bernoulli model (with 0 and 1 occurring, respectively, with probability p and $q = 1-p$). Let B_m^k be the *expected* number of strings on level k of a randomly built b-digital search tree. From the above we immediately obtain $\Pr\{D_m = k\} = B_m^k/m$; thus one can alternatively study the average B_m^k, which is further called *the average profile*. Let $B_m(u) = \sum_{k\geq 0} B_m^k u^k$ be the generating function of the average profile.

A digital tree is a recursive structure. Suppose that there are $m+b$ strings to store. The root of such a tree contains b strings, and the remaining m strings are split between the left subtree and the right subtree. If i strings go to the left subtree, then its average profile is characterized by $uB_i(u)$, while $uB_{m-i}(u)$ is the generating function for the right subtree. Finally, the probability that i strings end up in the left subtree is equal to the probability that i out of m strings start with 0, and this is equal to $\binom{m}{i}p^iq^{m-i}$. Thus we have the following recurrence for $m \geq 0$:

$$B_{m+b}(u) = b + u\sum_{i=0}^{m}\binom{m}{i}p^iq^{m-i}\left(B_i(u) + B_{m-i}(u)\right) \tag{4}$$

with initial conditions

$$B_i(u) = i \qquad \text{for} \qquad i = 0, 1, \ldots, b-1. \tag{5}$$

For $b = 1$ the above recurrence can be solved exactly as discussed in [37] (cf. [26]). Unfortunately, for $b > 1$ the recurrence becomes too complicated and no exact solution is known. This had been noted by Flajolet and Richmond [8] who developed a special technique to deal with such recurrences for $b > 1$. Unfortunately again, the technique of [8] was designed for the symmetric Bernoulli model and becomes very intricate for the asymmetric Bernoulli models. The reason is that some sums occurring in the solution of (4) cease to become harmonic sums in the asymmetric case.

In view of this, we approach the general recurrence (4) from a different "angle." First of all, we "poissonize" the model; that is, we introduce the Poisson transform (or Poisson generating function) as

$$\widetilde{B}(u,z) = \sum_{i=0}^{\infty} B_i(u)\frac{z^i}{i!}e^{-z}.$$

Then, the recurrence becomes a slightly more manageable differential functional equation, namely,

$$\left(1+\frac{\partial}{\partial z}\right)^b \widetilde{B}(u,z) = b + u\left(\widetilde{B}(u,pz) + \widetilde{B}(u,qz)\right), \tag{6}$$

where $(1+\frac{\partial}{\partial z})^b f(z) \stackrel{\text{def}}{=} \sum_{i=0}^{b}\binom{b}{i}\frac{\partial^i f(z)}{\partial z^i}$. We shall study $\widetilde{B}(u,z)$ for $z \to \infty$ in a cone around the real axis and u in a compact set around $u = 1$. This will suffice to recover asymptotics of $B_m(u)$, as discussed in section 3.2.

In passing, we should point out that $\widetilde{B}(u,z)$ represents the average profile in the so-called Poisson model in which the fixed number of strings is replaced by a random number of strings according to a Poisson distribution with mean z. To take full advantage of this new model, however, we shall postulate that z is a complex variable. After "depoissonization" (cf. section 3.2) we expect that $B_m(u) \sim \widetilde{B}(u,m)$.

In the next section, we use the Mellin transform [9], singularity analysis [7], and the depoissonization lemma [16, 34] to solve (6) and to prove the following main result. Below, we write log for natural logarithm.

THEOREM 2.1 (asymmetric Bernoulli model).

(i) *Let D_m be the typical depth in a b-digital tree built over m statistically independent strings under the* asymmetric *Bernoulli model. Then*

$$ED_m = \frac{1}{h_1}\log m + \frac{1}{h_1}\left(\frac{h_2}{2h_1} + \gamma - 1 - H_{b-1} - \Delta(b,p) + \delta_1(m,b)\right) + O\left(\frac{\log m}{m}\right), \tag{7}$$

$$\text{Var } D_m = \frac{h_2 - h_1^2}{h_1^3}\log m + O(1), \tag{8}$$

where $h_1 = -p\log p - q\log q$ is the entropy of the Bernoulli(p) distribution, $h_2 = p\log^2 p + q\log^2 q$, and $\gamma = 0.577\ldots$ is the Euler constant, while $H_{b-1} = \sum_{i=1}^{b-1}\frac{1}{i}$, $H_0 = 0$ are harmonic numbers. The constant $\Delta(b,p)$ can be computed as follows (see Table 1, *section* 3.3, *for numerical values):*

$$\Delta(b,p) = \sum_{n=2b+1}^{\infty} \bar{f}_n \sum_{i=1}^{b} \frac{(i+1)b!}{(b-i)!n(n-1)\ldots(n-i-1)} < \infty, \tag{9}$$

where $\bar{f}_n$ is given recursively by

$$\begin{cases} f_{m+b} = m + \displaystyle\sum_{i=0}^{m}\binom{m}{i}p^i q^{m-i}(f_i + f_{m-i}), & m > 0, \\ f_0 = f_1 = \cdots = f_b = 0, & \\ \bar{f}_{m+b} = f_{m+b} - m > 0, & m \geq 1. \end{cases}$$

Finally, $\delta_1(x,b)$ is a fluctuating function with a small amplitude (cf. (48)*) when $(\log p)/(\log q)$ is rational, and for any fixed b $\lim_{x\to\infty}\delta_1(x,b) = 0$ otherwise.*

(ii) *Let $G_m(u)$ be the probability generating function of D_m (i.e., $G_m(u) = Eu^{D_m}$), $\mu_m = ED_m$, and $\sigma_m = \sqrt{\text{Var } D_m}$. Then, for complex τ,*

$$e^{-\tau\mu_m/\sigma_m}G_m(e^{\tau/\sigma_m}) = e^{\frac{\tau^2}{2}}\left(1 + O\left(\frac{1}{\sqrt{\log m}}\right)\right). \tag{10}$$

Thus, the limiting distribution of $\frac{D_m - \mu_m}{\sigma_m}$ is normal, and it converges in moments (i.e., in mean of any order) to the appropriate moments of the standard normal distribution. Also, there exist positive constants A and $\alpha < 1$ (that may depend on p and b) such that, uniformly in k for large m,

$$\Pr\left\{\left|\frac{D_m - c_1\log m}{\sqrt{c_2\log m}}\right| > k\right\} \leq A\alpha^k, \tag{11}$$

where $c_1 = 1/h_1$ and $c_2 = (h_2 - h_1^2)/h_1^3$.

The *symmetric* Bernoulli model must be treated differently since we shall prove below that Var $D_m = O(1)$, and hence a central limit theorem may not hold. We use the Flajolet and Richmond [8] technique to establish an asymptotic distribution in this case (cf. section 4.1). Using a probabilistic approach we also establish the exact distribution of D_m (cf. section 4.2). Both results are summarized in Theorem 2.2 below.

Before we present our findings, we must introduce some additional notation. Let

$$Q(t) = \prod_{k=0}^{\infty}(1 + t2^{-k}), \tag{12}$$

and for integer s and complex z we define

$$H(s) = \frac{\partial^s}{\partial z^s}\left(\frac{1}{Q^b(-z)}\right)\Big|_{z=1}; \tag{13}$$

$$R_i(s) = -\frac{\partial^s}{\partial z^s}\left(\prod_{k=1}^{i}(1 - z2^k)^{-b}\right)\Big|_{z=1}, \qquad R_0(s) = -1. \tag{14}$$

THEOREM 2.2 (symmetric Bernoulli model).

(i) *Let us consider the symmetric Bernoulli model (with $p = q = 1/2$). The mean value ED_m is given by* (7)*, while the variance becomes*

$$\begin{aligned}\operatorname{Var} D_m = {} & \frac{1}{12} + \frac{1}{L^2}\left(1 + \frac{\pi^2}{6}\right) + \frac{1}{L^2}\left(J''(0) - (J'(0))^2\right) \\ & + \frac{1}{L}\delta_2(\log_2 m) - [\delta_1^2]_0 + O\left(\frac{\log^2 m}{m}\right),\end{aligned} \tag{15}$$

where $L = h_1 = \log 2$. Then

$$J'(0) = \int_0^1 \left(\frac{1}{Q(t)^b} - 1\right)\frac{dt}{t} + \int_0^\infty \frac{1}{Q(t)^b}\frac{dt}{t} \tag{16}$$

and

$$J''(0) = -\frac{\pi^2}{3} + 2\int_0^1 \left(\frac{1}{Q(t)^b} - 1\right)\frac{\log t}{t}dt + 2\int_0^\infty \frac{1}{Q(t)^b}\frac{\log t}{t}dt, \tag{17}$$

where $J(s)$ is defined in (68) *of section* 4.1*, $\delta_2(\cdot)$ is a periodic function with mean* 0 *and period* 1*, and $[\delta_1^2]_0$ is a very small constant (e.g., $[\delta_1^2]_0 \le 10^{-10}$ for $b = 1$). More precisely, as in Hubalek* [12] *with $\chi_k = 2k\pi i/L$ for $k = \pm 1, \pm 2, \ldots,$*

$$[\delta_1^2]_0 = \frac{1}{L^2}\sum_{k\ne 0}\frac{I(\chi_k)I(-\chi_k)}{\Gamma(2 + \chi_k)\Gamma(2 - \chi_k)},$$

where $\Gamma(\cdot)$ is Euler's gamma function, and

$$I(\chi_k) = \frac{1}{\chi_k} + \int_0^1 \left(Q^{-b}(t) - 1\right)t^{\chi_k - 1}dt + \int_1^\infty Q^{-b}(t)t^{\chi_k}dt,$$

where $I(s)$ is defined in (67) *of section* 4.1.

(ii) *The exact distribution of D_m is given by*

$$\begin{aligned}m\Pr\{D_m \le j\} = {} & b - \frac{1}{(b-1)!}\sum_{k=1}^{j}(1 - 2^k)^{-b} \\ & \times \frac{\partial^{b-1}}{\partial z^{b-1}}\left(\frac{z^{2b}}{(z-1)^2}(z^{-b} - z^{-m})\prod_{1\le\ell\le j:\ell\ne k}\left(\frac{2^{-\ell}z}{1 - (1 - 2^{-\ell})z}\right)^b\right)\Bigg|_{z=(1-2^{-k})^{-1}}\end{aligned} \tag{18}$$

for any positive integer j.

(iii) *Now let* $j = \lfloor \log_2 m \rfloor + \kappa$ *for a fixed integer* κ*, and define* $\{\log_2 m\} = \log_2 m - \lfloor \log_2 m \rfloor$*. Then the "asymptotic distribution" of* D_m *can be expressed as*

$$\lim_{m\to\infty} \Big| \Pr\{D_m \le \lfloor \log_2 m \rfloor + \kappa\} - \sum_{\substack{\ell+s+t=b-1 \\ \ell,s,t \,\ge\, 0}} \frac{(s+1)}{\ell!} \sum_{i=0}^{\infty} \frac{K_i(t) e^{-2^{-(\kappa-\{\log_2 m\}-i)}}}{2^{-(\ell-1)(i-(\kappa-\{\log_2 m\}))}}$$

$$- \sum_{s+t=b-1} (s+1) \sum_{i=0}^{\infty} K_i(t) \left(e^{-2^{-(\kappa-\{\log_2 m\}-i)}} - 1 \right) 2^{\kappa-\{\log_2 m\}-i} \Big| = 0,$$

where $K_i(t) = \sum_{s_1+s_2=t} \frac{(-1)^t}{s_1! s_2!} R_i(s_2) H(s_1)$*. Observe that the limiting distribution of* D_m *does not exist due to the term* $\{\log_2 n\}$*.*

In passing it should be noted that the "asymptotic distribution" established in part (iii) above resembles a "mixture" of double exponential distributions (i.e., $e^{-2^{-x}}$), as in the case $b = 1$. An intuitive explanation for different behavior in the symmetric case is given in [26], but this follows basically from the fact that Var $D_m = O(1)$. We should also point out that numerical values of $J'(0)$ and $J''(0)$ can be found in Hubalek [12].

2.2. Lempel–Ziv model. The situation is similar, but *not* the same, in the *Lempel–Ziv model*, in which a sequence of *fixed length* n is parsed into random number of phrases. Let M_n denote the number of *full* phrases produced by the algorithm (the last incomplete phrase is ignored). We should mention that for $b > 1$ the number of full *distinct* phrases, M'_n, is not equal to the total number of full phrases M_n. Also let $D_n^{LZ}(i)$ be the length of the ith phrase in the Lempel–Ziv model, where $1 \le i \le M_n$. By the *typical* phrase length D_n^{LZ} we mean the length of a randomly selected phrase (i.e., conditioned on M_n each phrase has probability $1/M_n$ of being selected).

The typical depth D_n^{LZ} in the Lempel–Ziv model can be estimated as follows (cf. [26]):

$$\Pr\{D_n^{LZ} = k\} = \sum_{m=m_L}^{m_U} \Pr\{D_n^{LZ} = k | M_n = m\} \Pr\{M_n = m\}, \tag{19}$$

where m_L and m_U are lower and upper bounds for the number of phrases. It is easy to see that there exist constants $\alpha_1 > 0$ and $\alpha_2 < \infty$ such that $m_L = \alpha_1 \sqrt{n/b} \le M_n \le \alpha_2 (n/b)/\log_2(n/b) = m_U$. Indeed, the minimum number of phrases occurs only for two strings: either all 0s or all 1s, and then $n = \sum_{i=1}^{M_n} D_n^{LZ}(i) \le b \sum_{i=1}^{M_n} i$, whence the lower bound $m_L = \Omega(\sqrt{n/b})$. For the upper bound, we consider a complete binary tree with the internal path length equal to n. Naturally, the number of nodes in such a tree is $O((n/b)/\log_2(n/b))$.

To estimate the probabilities appearing in (19) one seeks the limiting distribution of M_n. This is a difficult problem even for $b = 1$, and only recently Jacquet and Szpankowski [14] "cracked" it by showing that M_n appropriately normalized weakly converges to the standard normal distribution. The case $b > 1$ is still unsolved; however, the technique of [14] can handle this case, too. To see this, we first reduce the problem to another one on the digital tree model. Indeed, observe that the following relationship between M_n (Lempel–Ziv model) and $L_m = \sum_{i=1}^{m} D_m(i)$ (digital tree model) takes place:

$$M_n = \max \left\{ m : \; L_m = \sum_{i=1}^{m} D_m(i) \le n \right\},$$

which immediately implies

$$\Pr\{M_n \geq m\} = \Pr\{L_m \leq n\}. \tag{20}$$

The above relationship is known as the *renewal equation*, and from Theorem 17.3 of [2], we conclude the central limit theorem for M_n knowing it holds for L_m. The latter is easier to handle but far from trivial; see [14] for details.

One finds a similar situation for the case $b > 1$; thus a central limit theorem for the internal path length L_m should hold. The exponential generating function $L(z,u) = \sum_{m=0}^{\infty} Eu^{L_m} \frac{z^m}{m!}$ of the probability generating function of L_m satisfies the following partial-functional differential equation:

$$\frac{\partial^b L(z,u)}{\partial z^b} = L(pzu,u)L(qzu,u). \tag{21}$$

The arguments from [14] can be extended to $b > 1$, after some tedious labor, and one can solve (21) asymptotically . We formulate our conclusions in a form of a fact that follows from [14] but without providing any detailed derivation.

FACT 1. *Consider the asymmetric Bernoulli model. Let* $c_1 = 1/h_1$ *and* $c_2 = (h_2 - h_1^2)/h_1^3$.
(i) *The path length* L_m *in a* b*-digital search tree possesses the following limiting distribution*

$$\frac{L_m - EL_m}{\sqrt{\text{Var } L_m}} \to N(0,1), \tag{22}$$

where $N(0,1)$ *denotes the standard normal distribution,* $EL_m = mED_m$, Var $L_m = c_2 m \log m + O(m)$, *and the convergence is also in moments.*

(ii) *The number of phrases* M_n *of the generalized Lempel–Ziv parsing scheme satisfies the following*

$$\frac{M_n - EM_n}{\sqrt{\text{Var } M_n}} \to N(0,1), \tag{23}$$

where $EM_n \sim nh_1/\log n$ *and* Var $M_n \sim c_2 h_1^3 n/\log^2 n$. *Moreover, all moments of* M_n *converge to the appropriate moments of the normal distribution.*

Having settled this, we can return to evaluating the limiting distribution of the phrase length D_n^{LZ}. According to (19), one needs to estimate the conditional probability $\Pr\{D_n^{LZ} = k | M_n = m\}$. It is tempting to assume that it is equal to $\Pr\{D_m = k\}$ (the latter refers to the probability distribution of the depth in the digital tree model). But this is *untrue* due to the fact that in the Lempel–Ziv model we consider *only* those digital search trees whose internal path length is fixed and equal to n. Clearly, this restriction affects the depth of a randomly selected phrase. A mathematical form of this dependency is actually written down in (20). We can use exactly the same arguments as in Louchard and Szpankowski [26] (cf. section III-B of [26]) to show that, for $b > 1$,

$$\Pr\{D_n^{LZ} = k | M_n = m\} = \left(1 + O\left(\sqrt{(\log n)/n}\right)\right) \Pr\{D_m = k\} \tag{24}$$

holds as long as $k = O(ED_m) = O(\log m)$. This would particularly imply (using Fact 1(ii)) that, for complex ϑ,

$$Ee^{\vartheta D_n^{LZ}} \sim Ee^{\vartheta D_{nh_1/\log n}}$$

asymptotically as $n \to \infty$ (cf. [26] for details).

In summary, the second main result concerning the Lempel–Ziv model is presented below (for simplicity we formulate it only for the asymmetric case).

THEOREM 2.3. *Consider the asymmetric Bernoulli model. Let D_n^{LZ} be the length of a randomly selected phrase in the generalized Lempel–Ziv scheme that partitions a string of length n. Then*

$$\frac{D_n^{LZ} - c_1 \log(nh_1/\log n)}{\sqrt{c_2 \log(nh_1/\log n)}} \to N(0,1). \tag{25}$$

More precisely, for complex ϑ,

$$e^{-\vartheta c_1 \sqrt{\log(nh_1/\log n)}} E\left(e^{\vartheta D_n^{LZ}/\sqrt{\log(nh_1/\log n)}}\right) = e^{c_2\vartheta^2/2}\left(1 + O\left(1/\sqrt{\log(n/\log n)}\right)\right). \tag{26}$$

Furthermore, there exist two positive constants A' and $\alpha_1 < 1$ such that

$$\Pr\left\{\left|\frac{D_m^{LZ} - c_1 \log(nh_1/\log n)}{\sqrt{c_2 \log(nh_1/\log n)}}\right| > k\right\} \le A'\alpha_1^k \tag{27}$$

uniformly in k for large m, where $c_1 = 1/h_1$ and $c_2 = (h_2 - h_1^2)/h_1^3$ as before.

The symmetric Bernoulli model can be handled in a similar manner, but its formulation is too complicated to be presented in a compact form. It is described by a similar formula as for the digital tree model with m replaced by $n/\log_2 n$. Naturally, the limiting distribution does not exist as such but some limiting theorem can be formulated, as in the case of the digital tree model (cf. Theorem 2.2(iii)).

Finally, in order to find an optimal b that possibly asymptotically minimizes the generalized Lempel–Ziv code length, we shall deal with the *average redundancy* $\bar{r}_n$ defined as

$$\bar{r}_n = \frac{E\ell_n - nh_1}{n},$$

where ℓ_n is the length of the generalized Lempel–Ziv code and the expectation is taken with respect to the underlying probability measure. As explained in section 1, the data compression code for the generalized Lempel–Ziv scheme consists of pairs of numbers, one being a pointer to the previous occurrence of the prefix of the phrase and the second number either containing the last symbol of a new phrase in the case it is the first phrase among b identical phrases or otherwise being empty. Clearly, the length ℓ_n of such a code depends on two parameters, namely, the number of phrases M_n and the number of *distinct* phrases M_n', and it can be computed (for a binary alphabet) as

$$\ell_n = M_n \log M_n' + \log 2 \cdot \sum_{i=1}^{M_n} I_i, \tag{28}$$

where I_i is equal to 1 if the ith phrase consists of a previously occurring prefix and an additional symbol and 0 otherwise (in the case of a general alphabet Σ of size $|\Sigma|$ the $\log 2$ factor in the second term of (28) should be replaced by $\log|\Sigma|$). In the above, for simplicity of presentation, the length ℓ_n is expressed in nats instead of bits since we use natural logarithm $\log M_n'$ instead of $\log_2 M_n'$. We should also point out that a particular implementation of the algorithm may lead to a slightly different formula

for ℓ_n but here we ignore these differences, concentrating only on the mathematical analysis.

It is not difficult to see (especially, if one considers the associated digital tree, as discussed in section 1), that

$$E\sum_{i=1}^{M_n} I_i = EM'_n.$$

Thus, $\ell_n = E(M_n \log M'_n) + \log 2 \cdot EM'_n$. But, as in [28], we notice that $E(M_n \log M'_n) = (EM_n) \log EM'_n + O(1/\log n)$. To estimate EM_n we observe that it is related to the internal path length in the associated digital tree, and $EL_m = mED_m$ (cf. (20)). As in Louchard and Szpankowski [28], we conclude that $EM_n \sim x_n$, where x_n is a solution of $EL_{x_n} = n$; that is,

$$x_n = \frac{nh_1}{\log n}\left(1 + \frac{\log\log n}{\log n} + \frac{A - \log h_1}{\log n} + O\left(\frac{(\log\log n)^2}{\log^2 n}\right)\right).$$

Here $-A$ is h_1 times the $\Theta(1)$ term in (7) of Theorem 2.1(i); that is,

$$A = 1 + H_{b-1} + \Delta(b,p) - \frac{h_2}{2h_1} - \gamma - \delta_1(m,b).$$

In a similar fashion we can estimate EM'_n; however, one should observe that M'_n is related to the size of the associated b-DST. More precisely, if S_m is the size of a b-DST built from m strings, then $M'_n = S_{M_n} - 1$ (since we count only nonroot nodes). However, according to Flajolet and Richmond [8] (symmetric case) and Louchard [25] (asymmetric case),

$$ES_m = m\,(q_0(b) + \delta_2(m,b)) + O(1),$$

where $q_0(b)$ is a constant that can be computed explicitly. For example, Flajolet and Richmond [8] proved that

$$q_0(b) = \frac{1}{\log 2}\int_0^\infty \left(\frac{1+t}{Q(t)}\right)^b \frac{dt}{1+t},$$

where, as in (12), $Q(t) = \prod_{j=0}^\infty (1 + t2^{-j})$. In particular, $q_0(1) = 1$, and the authors of [8] computed $q_0(2) = 0.5747$, $q_0(3) = 0.4069$, and so on. For large b, one derives that $q_0(b) \sim 1/(b \log 2)$. In summary, $EM'_n \sim x_n(q_0(b) + \delta_2(n,b))$.

Putting everything together, and using the approach from [28], we finally arrive at the following formula for the average redundancy of the generalized Lempel–Ziv code:

$$\bar{r}_n(b) = h_1 \frac{1 - \gamma - \frac{h_2}{2h_1} + \Delta(b,p) + H_{b-1} + q_0(b)\log 2 + \log q_0(b) - \delta(n,b)}{\log n} + O\left(\frac{\log\log n}{\log^2 n}\right),$$

where $\delta(\cdot)$ is a fluctuating function with a small amplitude, and the other quantities are defined as above.

It may be interesting to compare the average redundancy for different values of b hoping that there exists an optimal value of b. For example, for the symmetric Bernoulli model with a binary alphabet, we obtain

$$\bar{r}_n(1) = \frac{2.27 + \delta(n)}{\log_2 n} + O\left(\frac{\log\log n}{\log^2 n}\right),$$
$$\bar{r}_n(2) = \frac{1.98 + \delta(n)}{\log_2 n} + O\left(\frac{\log\log n}{\log^2 n}\right),$$
$$\bar{r}_n(3) = \frac{1.89 + \delta(n)}{\log_2 n} + O\left(\frac{\log\log n}{\log^2 n}\right),$$
$$\bar{r}_n(\infty) = \frac{1.71 + \delta(n)}{\log_2 n} + O\left(\frac{\log\log n}{\log^2 n}\right).$$

Furthermore, some recent preliminary experimental results (cf. [13]) carried out on structured ASCII files indicate that a practical saving can be achieved for $b > 1$, and this is particularly true for large alphabets (e.g., image), as already indicated in section 1. We should point out that these experimental findings are very sensitive to implementation issues. For example, a particular implementation can add $\Omega(M_n)$ bits which contribute $\Omega(1/\log n)$ to the expected redundancy $\bar{r}_n$. Observe that the leading term of $\bar{r}_n$ is only of order $\Omega(1/\log n)$.

3. Analysis of the asymmetric Bernoulli model. In this section, we prove Theorem 2.1 concerning the digital tree model in the asymmetric Bernoulli model. After establishing recurrences for the mean and variance, we proceed to derive the asymptotics of these quantities. We first deal with the Poisson model (section 3.1), and then depoissonize the results (section 3.2). Special attention is devoted to computing a certain constant arising in the analysis (section 3.3). Finally, we show how to derive the limiting distribution for D_m (section 3.4).

3.1. Analysis of moments in the Poisson model. As defined in section 2.1, $B_m(u)$ is the generating function of the average profile B_m^k. Observe that $B_m(1) = m$, and $ED_m = B_m'(1)/m$, and $B_m''(1)/m = E\{D_m(D_m-1)\} = \mathrm{Var}D_m - ED_m + (ED_m)^2$. Thus,

$$\text{Var } D_m = \frac{B_m''(1)}{m} + \frac{B_m'(1)}{m} - \left(\frac{B_m'(1)}{m}\right)^2. \tag{29}$$

We will use the above formulas to derive asymptotics of ED_m and Var D_m as $m \to \infty$.

Our approach is analytic, and as mentioned in the previous section, we first derive the mean and the second factorial moment of the average profile in the Poisson model, that is, the first and the second derivative with respect to u at $u = 1$ of $\widetilde{B}(u, z)$. Since $\widetilde{B}(u, z)$ is analytic (jointly in u and z), one immediately obtains

$$\left(1 + \frac{\partial}{\partial z}\right)^b \widetilde{B}_u(u, z) = \left(\widetilde{B}(u, pz) + \widetilde{B}(u, qz)\right) + u\left(\widetilde{B}_u(u, pz) + \widetilde{B}_u(u, qz)\right),$$
$$\left(1 + \frac{\partial}{\partial z}\right)^b \widetilde{B}_{uu}(u, z) = 2\left(\widetilde{B}_u(u, pz) + \widetilde{B}_u(u, qz)\right) + u\left(\widetilde{B}_{uu}(u, pz) + \widetilde{B}_{uu}(u, qz)\right).$$

Let $\widetilde{B}_u(1, z) = \widetilde{X}(z)$, $\widetilde{B}_{uu}(1, z) = \widetilde{W}(z)$, which suffice to compute the mean and the variance of D_m, as previously indicated. Then

$$\left(1+\frac{\partial}{\partial z}\right)^b \widetilde{X}(z) = z + \widetilde{X}(pz) + \widetilde{X}(qz), \tag{30}$$

$$\left(1+\frac{\partial}{\partial z}\right)^b \widetilde{W}(z) = 2\left(\widetilde{X}(pz) + \widetilde{X}(qz)\right) + \left(\widetilde{W}(pz) + \widetilde{W}(qz)\right). \tag{31}$$

Our goal is now to solve (30) and (31) asymptotically (as $z \to \infty$ in a cone around $\Re(z) > 0$). It is well known that equations like these are amiable to attack by the Mellin transform. To recall, for a function $f(x)$ of real x, we define its Mellin transform $F(s)$ as

$$F(s) = \mathcal{M}[f(t); s] = \int_0^\infty f(t) t^{s-1} dt.$$

In some of our arguments (e.g., depoissonization in section 3.2 and singularity analysis in section 4.1), we could use either Mellin transform of a complex variable function $f(z)$ or an analytic continuation argument. It is known (cf. [5, 16]) that as long as $\arg(z)$ belongs to some cone around the real axis, the Mellin transform $F(s)$ of a function $f(x)$ of a real argument and its corresponding function of a complex argument is the same. Therefore, we work most of the time with the Mellin transform of a function of real variable as defined above.

Now let

$$X(s) = \mathcal{M}[\widetilde{X}(t); s] = \Gamma(s)\gamma(s), \tag{32}$$

$$Y(s) = \mathcal{M}[\widetilde{W}(t); s] = \Gamma(s)\beta(s), \tag{33}$$

where $\Gamma(s)$ is the classical gamma function, and we aim to compute $\gamma(s)$ and $\beta(s)$. They exist in a proper strip as proved in the following lemma.

LEMMA 3.1. *The following is true:* (i) $X(s)$ *exists for* $\Re(s) \in (-b-1, -1)$, *and* $Y(s)$ *is defined for* $\Re(s) \in (-2b-1, -1)$.

(ii) *Furthermore,* $\gamma(-1-i) = 0$ *for* $i = 1, \ldots, b-1$, $\gamma(-1-b) = (-1)^{b+1}$, *and* $\beta(-1-i) = 0$ *for* $i = 1, \ldots, b$, *and* $\gamma(s)$ *has simple poles at* $s = -1, 0, 1, \ldots$

Proof. By recurrence (4), we have $B_i(u) = i$ for $i = 0, 1, \ldots, b$ and thus $B_i(u) = b + (i-b)u$ for $i = 1+b, \ldots, 2b$. Taking derivatives, we obtain $\frac{\partial B_i(u)}{\partial u} = 0$ for $i = 0, 1, \ldots, b$ and $\frac{\partial B_i(u)}{\partial u} = i - b$ for $i = b, 1+b, \ldots, 2b$. Furthermore, the second derivative becomes $\frac{\partial^2 B_i(u)}{\partial u^2} = 0$ for $i = 0, 1, \ldots, 2b$. Hence, for $z \to 0$,

$$\begin{aligned}
\widetilde{X}(z) &= \left(z^{(b+1)}/(b+1)! + 2z^{(b+2)}/(b+2)! + 3z^{(b+3)}/(b+3)! + O(z^{b+4})\right) e^{-z} \\
&= z^{(b+1)}/(b+1)! + O(z^{b+2}) \quad \text{as} \quad z \to 0, \\
\widetilde{W}(z) &= O(z^{2b+1}) \quad \text{as} \quad z \to 0.
\end{aligned}$$

However, for $z \to \infty$ we conclude from (30) and (31) that $\widetilde{X}(z) = O(z \log z)$ and $\widetilde{W}(z) = O(z \log^2 z)$. Thus, the first part of the lemma is proven. Part (ii) follows directly from the lemma below and (38). ∎

LEMMA 3.2. *Let* $\{f_n\}_{n=0}^\infty$ *be a sequence of real numbers, and suppose that its Poisson generating function* $\widetilde{F}(z) = \sum_{n=0}^\infty f_n \frac{z^n}{n!} e^{-z}$ *is an entire function. Furthermore, let its Mellin transform* $F(s)$ *have the factorization* $F(s) = \mathcal{M}[\widetilde{F}(z); s] = \Gamma(s)\gamma(s)$, *and assume that* $F(s)$ *exists for* $\Re(s) \in (-2, -1)$ *while* $\gamma(s)$ *is analytic for* $\Re(s) \in (-\infty, -1)$. *Then*

$$\gamma(-n) = \sum_{k=0}^{n} \binom{n}{k} (-1)^k f_k \quad \textit{for} \quad n \geq 2. \tag{34}$$

Proof. Let a sequence $\{g_n\}_{n=0}^{\infty}$ be such that $\widetilde{F}(z) = \sum_{n=0}^{\infty} g_n \frac{z^n}{n!}$, that is (cf. [8]),

$$g_n = \sum_{k=0}^{n} \binom{n}{k} (-1)^{n-k} f_k, \qquad n \geq 0.$$

Now define, for some fixed $N \geq 2$, the function $\widetilde{F}_N(z) = \sum_{n=0}^{N-1} g_n \frac{z^n}{n!}$. Due to our assumptions, we can analytically continue $F(s)$ to the whole complex plane except $s = -2, -3, \ldots$ In particular, for $\Re(s) \in (-N, -N+1)$, we have

$$F(s) = \mathcal{M}[\widetilde{F}(z) - \widetilde{F}_N(z); s].$$

(The above is true since a polynomial in z such as $\widetilde{F}_N(z)$ can only shift the fundamental strip of the Mellin transform but cannot change its value [9].) As $s \to -N$, due to the assumed factorization $F(s) = \Gamma(s)\gamma(s)$, we have

$$F(s) = \frac{1}{s+N} \frac{(-1)^N}{N!} \gamma(-N) + O(1);$$

thus, by the inverse Mellin transform, we have

$$\widetilde{F}(z) - \widetilde{F}_N(z) = \frac{(-1)^N}{N!} \gamma(-N) z^N + O(z^{N+1}) \quad \text{as} \quad z \to 0. \tag{35}$$

But

$$\widetilde{F}(z) - \widetilde{F}_N(z) = \sum_{i=N}^{\infty} g_n \frac{z^n}{n!} = g_N \frac{z^N}{N!} + O(z^{N+1}). \tag{36}$$

Thus, by comparing (35) and (36), we prove that

$$\gamma(-N) = (-1)^N g_N = \sum_{k=0}^{N} \binom{N}{k} (-1)^k f_k \quad \text{for} \quad N \geq 2. \qquad \square$$

Now we are in a position to compute the Mellin transforms of $\widetilde{X}(z)$ and $\widetilde{W}(z)$. From (30) and (31), after taking Mellin transforms and using (32) and (33), we obtain

$$\sum_{i=0}^{b} \binom{b}{i} (-1)^i \gamma(s-i) = (p^{-s} + q^{-s})\gamma(s),$$

$$\sum_{i=0}^{b} \binom{b}{i} (-1)^i \beta(s-i) = 2(p^{-s} + q^{-s})\gamma(s) + (p^{-s} + q^{-s})\beta(s),$$

and, by Lemma 3.1, $\gamma(s)$ exists at least in $\Re(s) \in (-b-1, -1)$, while $\beta(s)$ is well defined in the strip $\Re(s) \in (-2b-1, -1)$. To simplify the above equations, we define, for any function $g(s)$,

$$\widehat{g}(s) = \sum_{i=1}^{b} \binom{b}{i} (-1)^{i+1} g(s-i), \tag{37}$$

provided $g(s-1),\dots,g(s-b)$ exist. Then

$$\gamma(s)=\frac{1}{1-p^{-s}-q^{-s}}\sum_{i=1}^{b}\binom{b}{i}(-1)^{i+1}\gamma(s-i)=\frac{1}{1-p^{-s}-q^{-s}}\widehat{\gamma}(s), \tag{38}$$

$$\begin{aligned}\beta(s)&=\frac{1}{1-p^{-s}-q^{-s}}\sum_{i=1}^{b}\binom{b}{i}(-1)^{i+1}\beta(s-i)+\frac{2(p^{-s}+q^{-s})}{1-p^{-s}-q^{-s}}\gamma(s)\\&=\frac{1}{1-p^{-s}-q^{-s}}\widehat{\beta}(s)+\frac{2(p^{-s}+q^{-s})}{(1-p^{-s}-q^{-s})^2}\widehat{\gamma}(s).\end{aligned} \tag{39}$$

Now let s_k, $k=0,\pm1,\pm2,\dots$, be roots of $1-p^{-s}-q^{-s}=0$. Observe that $s_0=-1$. Actually, s_k were studied quite intensively in the past, and the following is well known (e.g., see [15] for further references).

LEMMA 3.3. *Let s_k for $k=\dots,-2,-1,0,1,2,\dots$ all be solutions of $1-p^{-s}-q^{-s}=0$.*

(i) *Then*

$$-1\le\Re(s_k)\le\sigma_0,$$

where σ_0 is a positive solution of $1+q^{-s}=p^{-s}$.

(ii) *If $\Re(s_k)=-1$ and $\Im(s_k)\neq0$, then $(\log p)/(\log q)$ must be rational. Furthermore, if $\frac{\log p}{\log q}=\frac{r}{t}$, where $\gcd(r,t)=1$ for integers r,t, then*

$$s_k=-1+\frac{2kr\pi i}{\log p}$$

for all integers k.

Observe now that at $s=s_k$ we have

$$\frac{1}{1-p^{-s}-q^{-s}}=-\frac{1}{h(s_k)}\frac{1}{s-s_k}+\frac{h_2(s_k)}{2h^2(s_k)}+O(s-s_k), \tag{40}$$

where

$$h(t)=-p^{-t}\log p-q^{-t}\log q, \tag{41}$$

$$h_2(t)=p^{-t}\log^2 p+q^{-t}\log^2 q. \tag{42}$$

Expanding $\Gamma(s)\widehat{\gamma}(s)$ around $s=s_k$, we find

$$\Gamma(s)\widehat{\gamma}(s)=\Gamma(s_k)\widehat{\gamma}(s_k)+\left(\Gamma(s_k)\widehat{\gamma}'(s_k)+\Gamma'(s_k)\widehat{\gamma}(s_k)\right)(s-s_k)+O((s-s_k)^2).$$

Therefore, since $X(s)=\Gamma(s)\gamma(s)=\frac{1}{1-p^{-s}-q^{-s}}\Gamma(s)\widehat{\gamma}(s)$, we derive around $s=s_k\neq-1$ to obtain

$$\begin{aligned}X(s)=&-\frac{1}{s-s_k}\frac{\Gamma(s_k)}{h(s_k)}\widehat{\gamma}(s_k)+\frac{h_2(s_k)}{2h^2(s_k)}\Gamma(s_k)\widehat{\gamma}(s_k)\\&-\frac{1}{h(s_k)}\left(\Gamma(s_k)\widehat{\gamma}'(s_k)+\Gamma'(s_k)\widehat{\gamma}(s_k)\right)+O(s-s_k).\end{aligned} \tag{43}$$

In a similar manner, from (39) we have

$$\begin{aligned} Y(s) &= -\frac{1}{s-s_k}\frac{\Gamma(s_k)}{h(s_k)}\widehat{\beta}(s_k) + 2\Gamma(s_k)\left(\frac{1}{h^2(s_k)}\frac{1}{(s-s_k)^2} - \frac{h_2(s_k)-h^2(s_k)}{h^3(s_k)}\frac{1}{s-s_k}\right) \\ &\quad \times\ (\widehat{\gamma}(s_k) + (s-s_k)\widehat{\gamma}'(s_k)) + \frac{2\Gamma'(s_k)\widehat{\gamma}(s_k)}{h^2(s_k)}\frac{1}{s-s_k} + O(1) \\ &= \frac{2\Gamma(s_k)\widehat{\gamma}(s_k)}{h^2(s_k)}\frac{1}{(s-s_k)^2} + \left(\frac{2\Gamma'(s_k)\widehat{\gamma}(s_k)}{h^2(s_k)} - \frac{\Gamma(s_k)}{h(s_k)}\widehat{\beta}(s_k)\right. \\ &\quad \left. - 2\Gamma(s_k)\frac{h_2(s_k)-h^2(s_k)}{h^3(s_k)}\widehat{\gamma}(s_k) - \frac{2\Gamma(s_k)\widehat{\gamma}'(s_k)}{h^2(s_k)}\right)\frac{1}{s-s_k} + O(1). \end{aligned} \tag{44}$$

However, from (40) and (38) at $s = s_0 = -1$, we find

$$\begin{aligned} \gamma(s) &= \left(-\frac{1}{h_1}\frac{1}{s+1} + \frac{h_2}{2h_1^2}\right)(\widehat{\gamma}(-1) + (s+1)\widehat{\gamma}'(-1)) + O(s+1) \\ &= -\frac{1}{h_1}\frac{1}{s+1} + \frac{h_2}{2h_1^2} - \frac{\widehat{\gamma}'(-1)}{h_1} + O(s+1), \end{aligned} \tag{45}$$

$$\begin{aligned} \beta(s) &= \left(-\frac{1}{h_1}\frac{1}{s+1} + \frac{h_2}{2h_1^2}\right)\left(\widehat{\beta}(-1) + (s+1)\widehat{\beta}'(-1)\right) \\ &\quad + 2\left(\frac{1}{h_1^2}\frac{1}{(s+1)^2} - \frac{h_2-h_1^2}{h_1^3}\frac{1}{s+1}\right)(\widehat{\gamma}(-1) + (s+1)\widehat{\gamma}'(-1)) + O(1) \\ &= \frac{2}{h_1^2}\frac{1}{(s+1)^2} + \left(-2\frac{h_2-h_1^2}{h_1^3} + 2\widehat{\gamma}'(-1)\frac{1}{h_1^2}\right)\frac{1}{s+1} + O(1). \end{aligned} \tag{46}$$

In the above equation, we used the fact that $\widehat{\gamma}(-1) = 1$ and $\widehat{\beta}(-1) = 0$, which follows directly from Lemma 3.1. Observe now that $\Gamma(s) = -\frac{1}{s+1} + (\gamma - 1) + O(s+1)$; hence the Laurent expansion of $X(s)$ at $s = -1$ is

$$X(s) = \Gamma(s)\gamma(s) = \frac{1}{h_1}\frac{1}{(s+1)^2} - \left(\frac{h_2}{2h_1^2} - \frac{1}{h_1}\widehat{\gamma}'(-1) + \frac{\gamma-1}{h_1}\right)\frac{1}{s+1} + O(1). \tag{47}$$

In order to derive the asymptotic expansion of $\widetilde{X}(z)$ for large z, we use well-known arguments (cf. [4, 6, 9, 14, 26, 29]) of the inverse Mellin transform; that is,

$$\widetilde{X}(z) = \frac{1}{2\pi i}\int_{-\frac{3}{2}-i\infty}^{-\frac{3}{2}+i\infty} X(s)z^{-s}ds.$$

(The evaluation of this integral is quite standard (e.g., see [29]): we extend the line of integration to a big rectangle right to the integration line, and observe that bottom and top lines contribute negligibly because the gamma function decreases exponentially with the increase in the magnitude of the imaginary part, and the right side positioned at, say, d, contributes x^{-d} for $d \to \infty$.) However, to estimate the error term we must note, as observed in Lemma 3.1, that $\gamma(s)$ has additional simple poles at $s = 0, 1, \ldots$. The pole at $s = 0$ is a double pole of $X(s) = \Gamma(s)\gamma(s)$, and thus its contribution to $\widetilde{X}(z)$ is $O(\log z)$. Putting everything together, we finally arrive at

$$\widetilde{X}(z) = \frac{1}{h_1} z \log z + \left(\frac{h_2}{2h_1^2} - \frac{1}{h_1} \widehat{\gamma}'(-1) + \frac{\gamma - 1}{h_1} \right) z + \sum_{k \neq 0} \frac{\Gamma(s_k) \widehat{\gamma}(s_k)}{h(s_k)} z^{-s_k} + O(\log z). \tag{48}$$

Similarly, at $s = -1$,

$$Y(s) = -\frac{2}{h_1^2} \frac{1}{(s+1)^3} + \frac{2}{h_1} \left(\frac{h_2 - h_1^2}{h_1^2} - \frac{1}{h_1} \widehat{\gamma}'(-1) + \frac{\gamma - 1}{h_1} \right) \frac{1}{(s+1)^2} + O\left(\frac{1}{s+1} \right).$$

In addition, there is a double pole at $s = 0$; hence, by the inverse Mellin transform and Lemma 3.3, we obtain

$$\widetilde{W}(z) = \frac{1}{h_1^2} z \log^2 z + \frac{2}{h_1} \left(\frac{h_2 - h_1^2}{h_1^2} - \frac{1}{h_1} \widehat{\gamma}'(-1) + \frac{\gamma - 1}{h_1} \right) z \log z + 2 \sum_{k \neq 0} \frac{\Gamma(s_k) \widehat{\gamma}(s_k)}{h^2(s_k)} z^{-s_k} \log z + O(z)$$

for $z \to \infty$, where $O(z)$ comes from the term $O((s+1)^{-1})$. This formula will allow us to infer asymptotics of the variance of D_m.

3.2. Depoissonization. The above asymptotic formulas concern the behavior of the Poisson mean and the second factorial moment as $z \to \infty$. More precisely, we must restrict the growth of z to a cone $S_\theta = \{z : \ |\arg(z)| \leq \theta\}$ for some $|\theta| < \pi/2$. But our original goal was to derive asymptotics of the mean ED_m and the variance $\operatorname{Var} D_m$ in the Bernoulli model. To infer Bernoulli model behavior from its Poisson model asymptotics, we must apply the so-called *depoissonization lemma*. This lemma basically says that $mED_m \sim \widetilde{X}(m)$ and $mED_m(D_m - 1) \sim \widetilde{W}(m)$ under some weak conditions that are easy to verify in our case. The reader is referred to [16, 34] for more details about the depoissonization lemma. For completeness, however, we review some depoissonization results that are useful for our problem.

Let us consider a more general problem: For a random variable X_n, we define g_n as a functional of the distribution of X_n (e.g., $g_n = EX_n$ or $g_n = EX_n^2$, etc.) or, in general, we assume that g_n is a sequence of n. We may also need to consider the generating function $G_n(u) = Eu^{X_n}$ for a complex u, which can be viewed as such a g_n (with a parameter u belonging to a compact set). Define the Poisson transform of g_n as $\widetilde{G}(z) = \sum_{n=0}^{\infty} g_n \frac{z^n}{n!} e^{-z}$ (or, more generally, $\widetilde{G}(z,u) = \sum_{n=0}^{\infty} G_n(u) \frac{z^n}{n!} e^{-z}$ for u in a compact set). Assume that we know the asymptotics of $\widetilde{G}(z)$ for z large and belonging to a cone $S_\theta = \{z : |\arg(z)| \leq \theta\}$ for some $|\theta| < \pi/2$. How can we infer asymptotics of g_n from $\widetilde{G}(z)$? One possible answer is given in the depoissonization lemma below (cf. [16, 34]):

LEMMA 3.4 (depoissonization lemma).

(i) *Let $\widetilde{G}(z)$ be the Poisson transform of a sequence g_n that is assumed to be an entire function of z. We postulate that for $0 < |\theta| < \pi/2$ the following two conditions simultaneously hold for some numbers $A, B, \xi > 0$, β, and $\alpha < 1$:*

(I) *For $z \in S_\theta$,*

$$|z| > \xi \Rightarrow |\widetilde{G}(z)| \leq B|z|^\beta \Psi(|z|), \tag{49}$$

where $\Psi(z)$ is a slowly varying function (e.g., $\Psi(z) = \log^d z$ for some $d > 0$).

(O) *For $z \notin S_\theta$,*

$$|z| > \xi \Rightarrow |\widetilde{G}(z)e^z| \le A\exp(\alpha|z|). \tag{50}$$

Then, for large n,

$$g_n = \widetilde{G}(n) + O(n^{\beta-1}\Psi(n)) \tag{51}$$

or, more precisely,

$$g_n = \widetilde{G}(n) - \frac{1}{2}\widetilde{G}''(n) + O(n^{\beta-2}\Psi(n)).$$

(ii) *If conditions* (I) *and* (O) *hold for $\widetilde{G}(z,u)$ for u belonging to a compact set $\mathcal{U}$, then*

$$G_n(u) = \widetilde{G}(n,u) + O(n^{\beta-1}\Psi(n)) \tag{52}$$

for large n and uniformly in $u \in \mathcal{U}$.

To apply the above lemma to $\widetilde{X}(z)$ and $\widetilde{W}(z)$, one must check conditions (I) and (O). But condition (I) inside the cone S_θ is automatically satisfied due to the asymptotics of $\widetilde{X}(z)$ and $\widetilde{W}(z)$ just derived. Formally, we must use either complex variable Mellin transform or analytic continuation to establish $\widetilde{X}(z) = O(z\log z)$ and $\widetilde{W}(z) = O(z\log^2 z)$. Thus, it suffices to check condition (O) outside the cone[1] (in fact, the arguments below work fine also for verifying condition (I)).

We consider only $\widetilde{X}(z)$ since $\widetilde{W}(z)$ can be treated in a similar manner. Let $X(z) = \widetilde{X}(z)e^z$. Then, functional equation (30) transforms into

$$X^{(b)}(z) = X(zp)e^{zq} + X(zq)e^{zp} + ze^z,$$

where $X^{(b)}(z)$ denotes the bth derivative of $X(z)$. Observe that the above equation can be represented alternatively as

$$X(z) = \underbrace{\int_0^z \int_0^{w_1} \cdots \int_0^{w_{b-1}}}_{b \text{ times}} + \left(X(w_1 p))e^{w_1 q} + X(w_1 q)e^{w_1 p} + w_1 e^{w_1}\right) dw_b \cdots dw_2 dw_1, \tag{53}$$

where the integration is along lines in the complex plane.

We now prove $|X(z)| \le e^{\alpha|z|}$ for $z \notin S_\theta$ for $\alpha < 1$. We use induction over the so-called *increasing domains* defined as follows (cf. [16, 29]): For all positive integers $m \ge 1$ and constants $\xi, \delta > 0$, let

$$\mathcal{F}_m = \{z = \rho e^{i\vartheta} : \ \rho \in [\xi\delta, \xi\nu^{-m}], \ \ 0 \le \vartheta < 2\pi\},$$

where $\max\{p,q\} \le \nu < 1$ and $\delta \le \min\{p,q\}$. The point to observe is that if $z \in \mathcal{F}_{m+1} - \mathcal{F}_m$, then $zp, zq \in \mathcal{F}_m$, provided $|z| \ge \xi$, which is assumed to hold.

To carry out the induction, we first define $\bar{\mathcal{F}}_m = \mathcal{F}_m \cap \bar{S}_\theta$, where $\bar{S}_\theta$ denotes points in the complex plane outside S_θ. Since $X(z)$ is bounded for $z \in \bar{\mathcal{F}}_1$, the initial step of induction holds. Let us now assume that for some $m > 1$ and for $z \in \bar{\mathcal{F}}_m$ we have

[1]Recently, Jacquet and Szpankowski [17] proved that if an analytic continuation of g_n has a polynomial growth, then condition (O) from Lemma 3.4 is automatically satisfied.

$|X(z)| \le e^{\alpha|z|}$ with $\alpha < 1$. We intend to prove that $|X(z)| \le e^{\alpha|z|}$ for $z \in \bar{\mathcal{F}}_{m+1}$. Indeed, let $z \in \bar{\mathcal{F}}_{m+1}$. If also $z \in \bar{\mathcal{F}}_m$, then the proof is completed. So let us now assume that $z \in \bar{\mathcal{F}}_{m+1} - \bar{\mathcal{F}}_m$. Then since $zp, zq \in \bar{\mathcal{F}}_m$, we can use our induction hypothesis together with the integral equation (53) to obtain the following estimate for $|z| > \xi$ where ξ is large enough:

$$|X(z)| \le |z|^{b+1} \left(e^{|z|(p\alpha + q\cos\theta)} + e^{|z|(q\alpha + p\cos\theta)} + e^{|z|\cos\theta} \right).$$

Let us now define $1 > \alpha > \cos\theta$ such that the following three inequalities are simultaneously fulfilled:

$$\begin{aligned} |z|^b e^{|z|(p\alpha + q\cos\theta)} &\le \frac{1}{3} e^{\alpha|z|}, \\ |z|^b e^{|z|(q\alpha + p\cos\theta)} &\le \frac{1}{3} e^{\alpha|z|}, \\ |z|^{b+1} e^{|z|\cos\theta} &\le \frac{1}{3} e^{\alpha|z|}. \end{aligned}$$

Then for $z \in \bar{\mathcal{F}}_{m+1}$ we have $|X(z)| \le e^{\alpha|z|}$, as needed to verify condition (O) of the depoissonization lemma.

Hence we can apply the depoissonization lemma to $\widetilde{X}(z)$, and using our previous asymptotics on $\widetilde{X}(z)$, we immediately obtain

$$\begin{aligned} ED_m &= \frac{\widetilde{X}(m)}{m} + O\left(\frac{\log m}{m}\right) \\ &= \frac{1}{h_1}\log m + \frac{h_2}{2h_1^2} - \frac{1}{h_1}\widehat{\gamma}'(-1) + \frac{\gamma - 1}{h_1} + \sum_{k \ne 0} \frac{\Gamma(s_k)\widehat{\gamma}(s_k)}{h(s_k)} m^{-1-s_k} + O\left(\frac{\log m}{m}\right). \end{aligned}$$

Another application of the depoissonization lemma leads to a formula on the second factorial moment:

$$\begin{aligned} ED_m(D_m - 1) = \frac{\widetilde{W}(m)}{m} + O(1) &= \frac{1}{h_1^2}\log^2 m \\ &\quad + 2\frac{1}{h_1}\left(\frac{h_2 - h_1^2}{h_1^2} - \frac{1}{h_1}\widehat{\gamma}'(-1) + \frac{\gamma - 1}{h_1}\right)\log m \\ &\quad + 2\sum_{k \ne 0} \frac{\Gamma(s_k)\widehat{\gamma}(s_k)}{h^2(s_k)} m^{-1-s_k}\log m + O(1). \end{aligned}$$

Finally, after computing $(ED_m)^2$ we arrive at

$$\begin{aligned} \operatorname{Var} D_m &= ED_m(D_m - 1) + ED_m - (ED_m)^2 \\ &= \frac{h_2 - h_1^2}{h_1^3}\log m + 2\sum_{k \ne 0} \frac{\Gamma(s_k)\widehat{\gamma}(s_k)}{h(s_k)} \left(\frac{1}{h(s_k)} - \frac{1}{h_1}\right) m^{-1-s_k}\log m + O(1). \end{aligned}$$

If $\Re(s_k) = -1$ for all k, then by Lemma 3.3 one can prove that $h(s_k) = h_1$ (cf. [15]). If $\Re(s_k) > -1$, then $m^{-1-s_k}\log m = o(1)$. Therefore, $\operatorname{Var} D_m = \frac{h_2 - h_1^2}{h_1^3}\log m + O(1)$. From Lemma 3.3 we know that $\Re(s_k) = -1$ whenever $(\log p)/(\log q)$ is rational; otherwise $\Re(s_k) > -1$. In summary, to complete the proof of Theorem 2.1(i) we must evaluate the constant $\widehat{\gamma}'(-1)$, which we discuss next.

3.3. Evaluation of some constants. In several applications (e.g., the computation of the average code redundancy discussed at the end of section 2) the second-order term of ED_m (i.e., the leading term of $ED_m - h_1^{-1}\log m$) plays a very important role. Therefore, knowing its value, or providing a numerical algorithm to compute it, is of prime interest. In the previous subsection, we showed that the value of this term depends on $\widehat{\gamma}'(-1)$, which can be also expressed as

$$\widehat{\gamma}'(-1) = \sum_{i=1}^{b} \binom{b}{i} (-1)^{i+1} \gamma'(-1-i),$$

where $\gamma(s)\Gamma(s) = \mathcal{M}[\widetilde{X}(t); s]$ and $\widetilde{X}(z) = \sum_{n=0}^{\infty} f_n \frac{z^n}{n!} e^{-z}$. We recall from Theorem 2.1 that f_n is defined as

$$\begin{cases} f_{m+b} = m + \displaystyle\sum_{i=0}^{m} \binom{m}{i} p^i q^{m-i} (f_i + f_{m-i}), \quad m \geq 0, \\ f_0 = f_1 = \cdots = f_b = 0, \\ \bar{f}_{m+b} = f_{m+b} - m, \quad m \geq 1. \end{cases}$$

Clearly, $\bar{f}_i > 0$ for any $i > b+1$ since $f_i \geq i - b$ for $i \geq b$.

To compute $\widehat{\gamma}'(-1)$ we must find $\gamma(s)$ in terms of computable quantities such as f_n. We proceed as follows:

$$\gamma(s) = \frac{1}{\Gamma(s)} \mathcal{M}\left[\sum_{n=b+1}^{\infty} f_n \frac{z^n}{n!} e^{-z}; s\right] = \sum_{n=b+1}^{\infty} \frac{f_n}{n!} \frac{\Gamma(s+n)}{\Gamma(s)}$$

$$= \sum_{n=b+1}^{\infty} \frac{f_n}{n!} s(s+1)\dots(s+n-1). \tag{54}$$

We assume above that $\Re(s) \in (-b-1, -1)$ to ensure the existence of the Mellin transform and the convergence of the series. Then one easily derives

$$\gamma'(s) = \sum_{n=b+1}^{\infty} \frac{f_n}{n!} s(s+1)\cdots(s+n-1) \sum_{i=0}^{n-1} \frac{1}{s+i}, \qquad s \notin \{-2, -3, \dots, -b\}. \tag{55}$$

After some algebra, we arrive at the following:

$$\gamma'(-k) = (-1)^k \sum_{n=b+1}^{\infty} \frac{f_n}{n!} k!(n-k-1)! \quad \text{for} \quad k = 2, \dots, b,$$

$$\gamma'(-b-1) = (-1)^b H_{b+1} + (-1)^{b+1} \sum_{n=b+2}^{\infty} \frac{f_n}{n!} (b+1)!(n-b-2)!.$$

Let us first assume that $b > 1$. Then, to estimate $\widehat{\gamma}'(-1)$, we proceed as follows:

$$\widehat{\gamma}'(-1) = \sum_{i=1}^{b} \binom{b}{i} (-1)^{i+1} \gamma'(-i-1)$$

$$= -H_{b+1} + \frac{1}{b+1} \sum_{i=1}^{b-1} \frac{i+1}{b-i} + \sum_{i=1}^{b} \binom{b}{i} \sum_{n=b+2}^{\infty} \frac{f_n}{n!} (i+1)!(n-i-2)!$$

$$= -H_{b+1} - H_{b-1} - \frac{b-1}{b+1} + \sum_{n=b+2}^{\infty} (n-b+\bar{f}_n) \sum_{i=1}^{b} \frac{(i+1)b!}{(b-i)!n(n-1)\dots(n-i-1)}$$
$$= -\frac{1}{b} - \frac{b}{b+1} + A + \Delta(b,p),$$

where

$$\Delta(b,p) = \sum_{n=b+2}^{\infty} \bar{f}_n \sum_{i=1}^{b} \frac{(i+1)b!}{(b-i)!n(n-1)\dots(n-i-1)},$$
$$A = \sum_{n=b+2}^{\infty} (n-b) \sum_{i=1}^{b} \frac{(i+1)b!}{(b-i)!n(n-1)\dots(n-i-1)}.$$

The above series converge since the summands are $O(\log n/n^2)$. Finally, observe that $\bar{f}_{m+b} = 0$ for $m = 1, 2, \dots, b$ and $\bar{f}_i > 0$ for $i > 2b$; hence

$$\Delta(b,p) = \sum_{n=b+2}^{\infty} \bar{f}_n \sum_{i=1}^{b} \frac{(i+1)b!}{(b-i)!n(n-1)\dots(n-i-1)}$$
$$= \sum_{n=2b+1}^{\infty} \bar{f}_n \sum_{i=1}^{b} \frac{(i+1)b!}{(b-i)!n(n-1)\dots(n-i-1)}.$$

After a long and tedious algebra (cf. [40]), we can prove that $A = H_b + b(1+b)^{-1}$. Hence $\widehat{\gamma}'(-1) = H_{b-1} + \Delta(b,p)$ as presented in Theorem 2.1, and this completes the proof of part (i) of Theorem 2.1 for $b > 1$.

For $b = 1$ we have

$$\widehat{\gamma}'(-1) = \gamma'(-2) = -H_2 + \Delta(1,p) + \sum_{n=3} \frac{2}{n(n-2)} = \Delta(1,p),$$

since the above series is equal to $3/2$, which is canceled by $-H_2 = -3/2$. Thus Theorem 2.1 is also proved for $b = 1$. Actually, in this case we may also conclude from [38] that

$$\Delta(1,p) = -\sum_{k=1}^{\infty} \frac{p^{k+1}\log p + q^{k+1}\log q}{1 - p^{k+1} - q^{k+1}}.$$

In Table 1 we present numerical values of $\Delta(b,p)$ and $ED_m - \frac{1}{h_1}(\log m - \delta(m,b))$ as a function of b. While $\Delta(b,p)$ is relatively easy to compute numerically, we must point out that the rate of convergence for this series is only $O(\log N/N)$, where N is the cutoff value of the series computation.

3.4. Limiting distribution. In this section, we will prove part (ii) of Theorem 2.1; that is, we establish the central limit theorem for D_m. We recall that $\widetilde{B}(u,z) = \sum_{i=0}^{\infty} B_i(u)\frac{z^i}{i!}e^{-z}$ and that

$$\left(1 + \frac{\partial}{\partial z}\right)^b \widetilde{B}(u,z) = b + u\left(\widetilde{B}(u,pz) + \widetilde{B}(u,qz)\right). \tag{56}$$

TABLE 1
Numerical values of $\Delta(b,p)$ *and* $ED_m - \frac{1}{h_1}\log m$ *for* $p = 0.3$

b	$\Delta(b,p)$	$ED_m - \frac{1}{h_1}(\log m - \delta(m,b))$
1	1.25	− 2.04
2	0.96	− 3.20
3	0.91	− 3.94
5	0.83	− 4.76
8	0.76	− 5.48
20	0.60	− 6.78
50	0.36	− 7.91
90	0.12	− 8.49

For some function $\omega(u,s)$, let the Mellin transform of $\widetilde{B}(u,z)$ be given by

$$Z(u,s) = \mathcal{M}\left(\widetilde{B}(u,z) - z; s\right) = \Gamma(s)\omega(u,s). \tag{57}$$

The existence of the Mellin transform $Z(u,s)$ is proved in the lemma below.

LEMMA 3.5. (i) *The Mellin* $Z(u,s)$ *exists for* $\Re(s) \in (-b-1,-1)$.
(ii) *For* $i = 1,\dots,b-1$ *we have* $\omega(u,-1-i) = 0$ *and* $\omega(u,-1-b) = (-1)^{b+1}(u-1)$.

Proof. The proof uses the same arguments as in Lemma 3.1. In particular,

$$\begin{aligned}\widetilde{B}(u,z) &= \left(z + z^2 + z^3/2! + \cdots + z^b/(b-1)! + (u+b)z^{b+1}/(b+1)! + O(z^{b+2})\right)e^{-z}\\ &= z + (u-1)z^{b+1}/(b+1)! + O(z^{b+2}).\end{aligned}$$

Thus as $z \to 0$ one obtains $\widetilde{B}(u,z) - z = O(z^{(b+1)})$. For fixed u, we also have $\widetilde{B}(u,z) = O(z\log z)$ for $z \to \infty$. Therefore, part (i) is proved. Part (ii) follows from Lemma 3.2. □

The plan for this section is similar to the previous one. We first use the Mellin transform technique to derive asymptotics of $\widetilde{B}(z,u) - z$ for $z \to \infty$ in a cone S_θ and then depoissonize this result by Lemma 3.4. We start with taking the Mellin transform to (56). After some algebra, we obtain

$$\sum_{i=0}^{b}\binom{b}{i}(-1)^i\omega(u,s-i) = u(p^{-s}+q^{-s})\omega(u,s),$$

which further leads to

$$\omega(u,s) = \frac{1}{1-u(p^{-s}+q^{-s})}\widehat{\omega}(u,s).$$

Now let $s_k(u), k = 0,\pm1,\pm2,\dots,$ be the roots of the equation $1-u(p^{-s}+q^{-s}) = 0$ for fixed u. Then, for $s = s_k(u)$,

$$\frac{1}{1-u(p^{-s}+q^{-s})} = \frac{1}{s-s_k(u)}\frac{u^{-1}}{-h(s_k(u))}.$$

In addition, one must consider two poles of the gamma function $\Gamma(s)$ at $s_{-1} = -1$ and $s_0 = 0$. The latter pole contributes $O(1)$, and the former $-z\omega(u,-1)$. But, by Lemma 3.5, we know that $\omega(u,-1) = 1$; thus the total contribution of these two poles is $-z + O(1)$.

Summing up, we have

$$\widetilde{B}(u,z)7 = \frac{u^{-1}}{h(s_0(u))}\Gamma(s_0(u))\widehat{\omega}(u,s_0(u))z^{-s_0(u)} + \sum_{k\neq 0}\frac{u^{-1}}{h(s_k(u))}\Gamma(s_k(u))\widehat{\omega}(u,s_k(u))z^{-s_k(u)} + O(1).$$

We now set $u = e^t$ for complex t in the vicinity of 0. Algebra similar to that in [14, 26] leads to the following for $t \to 0$:

$$\begin{aligned} s_0(t) &= -1 - \frac{t}{h_1} - \frac{\alpha t^2}{2} + O(t^3), \\ \Gamma(s_0(t)) &= \frac{h_1}{t} + O(t^2), \\ \frac{e^{-t}}{h(s_0(t))} &= \frac{1}{h_1} + O(t), \\ \widehat{\omega}(t,s_0(t)) &= e^t - 1 + O(t^2) = t + O(t^2). \end{aligned} \tag{58}$$

The rest is a matter of depoissonization. But the depoissonization conditions (I) and (O) of Lemma 3.4 are easy to verify for u belonging to a compact set around $u = 1$, as we already showed in the case of $\widetilde{X}(z)$. Thus, an application of (52) provides the following estimate:

$$B_m(t) = \frac{1}{h_1}\frac{h}{t}\widehat{\omega}(t,s_0(t))m^{-s_0(t)} + e^{-t}\sum_{k\neq 0}\frac{1}{h(s_k(t))}\Gamma(s_k(t))\widehat{\omega}(u,s_k(t))m^{-s_k(t)} + O(\log m),$$

since $B(z,u) = O(z\log z)$. Then the generating function $G_m(t) = Ee^{tD_m}$ becomes

$$\begin{aligned} G_m(t) &= \frac{B_m(t)}{m} \\ &= \frac{1}{t}\widehat{\omega}(t,s_0(t))m^{-1-s_0(t)} + e^{-t}\sum_{k\neq 0}\frac{1}{h(s_k(t))}\Gamma(s_k(t))\widehat{\omega}(u,s_k(t))m^{-1-s_k(t)} \\ &\quad + O\left(\frac{\log m}{m}\right) \\ &= \frac{1}{t}(t-1)m^{-1-s_0(t)} + e^{-t}\sum_{k\neq 0}\frac{1}{h(s_k(t))}\Gamma(s_k(t))\widehat{\omega}(u,s_k(t))m^{-1-s_k(t)} \\ &\quad + O\left(\frac{\log m}{m}\right) \\ &= m^{-1-s_0(t)} + e^{-t}\sum_{k\neq 0}\frac{1}{h(s_k(t))}\Gamma(s_k(t))\widehat{\omega}(u,s_k(t))m^{-1-s_k(t)} + O\left(\frac{\log m}{m}\right). \end{aligned}$$

As the final step, we set $t = \frac{\tau}{\sigma_m}$ for some fixed τ and $\sigma_m = \operatorname{Var} D_m$. Then, using (58) $m^{-1-s_0(t)} = e^{\tau\mu_m/\sigma_m + \tau^2/2}$, as well as

$$e^{-\tau\mu_m/\sigma_m}G_m(e^{\tau/\sigma_m}) = e^{-\tau\mu_m/\sigma_m}$$
$$\times \left(\frac{1}{t}te^{\tau\mu_m/\sigma_m+\frac{\tau^2}{2}} + e^{-t}m^{-1-s_0(t)}\sum_{k\neq 0}\frac{1}{h(s_k(t))}\Gamma(s_k(t))\widehat{\omega}(u,s_k(t))m^{s_0(t)-s_k(t)} + O\left(\frac{\log m}{m}\right)\right)$$
$$= e^{\frac{\tau^2}{2}}\left(1+O\left(\sum_{k\neq 0}\frac{1}{h(s_k(t))}\Gamma(s_k(t))\widehat{\omega}(u,s_k(t))m^{s_0(t)-s_k(t)}\right)\right)$$
$$= e^{\frac{\tau^2}{2}}\left(1+O\left(\frac{1}{\sqrt{\log m}}\right)\right)$$

since, as in [14], we prove that (cf. [40])

$$\sum_{k\neq 0}\frac{1}{h(s_k(t))}\Gamma(s_k(t))\widehat{\omega}(u,s_k(t))m^{s_0(t)-s_k(t)} = O(t) = O\left(\frac{1}{\sqrt{\log m}}\right)$$

for $t = \tau/\sigma_m = O(1/\sqrt{\log m})$.

To complete the proof of part (ii) of Theorem 2.1, we must show that the above expression implies the convergence of moments. But this is standard (cf. [16]) and can be argued as follows: Let $D'_m = (D_m - \mu_n)/\sigma_m$. We just proved that $G_m(t) = E(e^{tD'_m}) \to e^{t^2/2}$ on t belonging to a real interval around $t = 0$. Hence $G_m(t)$ is bounded in the vicinity of $t = 0$ since $|E(e^{tD'_m})| \le E(e^{\Re(t)D'_m})$, which further implies that $G_m(t)$ is uniformly bounded around $t = 0$. By *Ascoli's theorem* we can select a subsequence from $G_m(t)$ that converges to a continuous function which must be analytic and equal to e^{t^2} inside the real interval (by uniqueness of analytic continuation). Since analytic functions that are uniformly bounded on a compact set must have all derivatives, we conclude the convergence in moments of D'_m, as desired. In summary, part (ii) of Theorem 2.1 and hence the theorem as a whole is proven.

4. Analysis of the symmetric Bernoulli model. In this section we prove Theorem 2.2 concerning the asymptotic behavior of a b-digital search tree in the unbiased (symmetric) Bernoulli model.

4.1. The variance. The average value ED_m follows directly from (7). But, in the symmetric case, $h_2 = h_1^2 = \log 2$, and therefore from (8) we deduce that Var $D_m = O(1)$. Our goal is to compute it precisely. In this case, an extension of a Flajolet and Richmond technique [8] works fine, and we apply it in this subsection. We follow Hubalek [12] to derive our results. We omit most of the detailed calculations, which can be found in [8, 12].

First, we observe that (6), our differential functional equation, in this case becomes

$$\left(1+\frac{\partial}{\partial z}\right)^b \widetilde{B}(u,z) = b + 2u\widetilde{B}(u,z/2).$$

The coefficients of $\widetilde{B}(u,z)$ can be computed by solving a linear recurrence of type (1). Unfortunately, there is no easy way to solve such a recurrence unless $b = 1$ (cf. [20, 38]). To circumvent this difficulty, Flajolet and Richmond [8] reduced it to a certain functional equation on an *ordinary* generating function that is easier to solve. We proceed along this path.

Let $\widetilde{B}(u,z) = \sum_{k=0}^{\infty} g_k(u)\frac{z^k}{k!}$, and $G(u,z) = \sum_{k=0}^{\infty} g_k(u)z^k$. We also define an ordinary generating function of $B_k(u)$ as $F(u,z) = \sum_{k=0}^{\infty} B_k(u)z^k$. Observe that $B_n(u) = \sum_{k=0}^{n} \binom{n}{k} g_k(u)$; hence as in [8] we obtain

$$F(u,z) = \frac{1}{1-z} G\left(u, \frac{z}{1-z}\right). \tag{59}$$

Indeed,

$$\begin{aligned}\frac{1}{1-z} G\left(u, \frac{z}{1-z}\right) &= \sum_{m=0}^{\infty} g_m(u) z^m \frac{1}{(1-z)^{m+1}} = \sum_{m=0}^{\infty} g_m(u) z^m \sum_{j=0}^{\infty} \binom{m+j}{j} z^j \\ &= \sum_{n=0}^{\infty} z^n \sum_{k=0}^{n} \binom{n}{k} g_k(u) = F(u,z).\end{aligned}$$

Certainly, (59) further implies that

$$F_u^{(n)}(u,z) = \frac{1}{1-z} G_u^{(n)}\left(u, \frac{z}{1-z}\right),$$

where $f_u^{(k)}(z,u)$ denotes the kth derivative of $f(z,u)$ with respect to u. Then

$$G(u,z)(1+z)^b = z(1+z)^b - z^{b+1} + 2uz^b G\left(u, \frac{z}{2}\right), \tag{60}$$

$$G'_u(u,z)(1+z)^b = 2z^b G\left(u, \frac{z}{2}\right) + 2uz^b G'_u\left(u, \frac{z}{2}\right), \tag{61}$$

$$G''_u(u,z)(1+z)^b = 4z^b G'_u\left(u, \frac{z}{2}\right) + 2uz^b G''_u\left(u, \frac{z}{2}\right). \tag{62}$$

In order to compute the variance, we compute $L^{\underline{1}}(z) := G'_u(u,z)|_{u=1}$ and $L^{\underline{2}}(z) := G''_u(u,z)|_{u=1}$ and then use (59). From (61) and (62) we immediately obtain

$$\begin{aligned}L^{\underline{1}}(z)(1+z)^b &= z^{b+1} + 2z^b L^{\underline{1}}\left(\frac{z}{2}\right), \\ L^{\underline{2}}(z)(1+z)^b &= 4z^b L^{\underline{1}}\left(\frac{z}{2}\right) + 2z^b L^{\underline{2}}\left(\frac{z}{2}\right).\end{aligned}$$

Iterating these equations we easily find (cf. [8, 12])

$$L^{\underline{1}}(z) = \sum_{k=0}^{\infty} \frac{(2z^b)(2(\frac{z}{2})^b)\cdots(2(\frac{z}{2^k})^b)}{\left((1+z)(1+\frac{z}{2})\cdots(1+\frac{z}{2^k})\right)^b} \frac{z}{2^{k+1}}, \tag{63}$$

$$L^{\underline{2}}(z) = \sum_{k=0}^{\infty} \frac{(2z^b)(2(\frac{z}{2})^b)\cdots(2(\frac{z}{2^k})^b)}{\left((1+z)(1+\frac{z}{2})\cdots(1+\frac{z}{2^k})\right)^b} 2L^{\underline{1}}(\frac{z}{2^{k+1}}). \tag{64}$$

The next step is to transform the above sums (63) and (64) into certain harmonic sums (cf. [9]). For this, we set $z = 1/t$ and define $Q(t) = \prod_{k=0}^{\infty}(1+\frac{t}{2^k})$. Then (63) and (64) become

$$\frac{tL^{\underline{1}}(\frac{1}{t})}{\left(Q(\frac{t}{2})\right)^b} = \sum_{k=0}^{\infty} \frac{1}{\left(Q(2^k t)\right)^b}, \tag{65}$$

$$\frac{tL^{\underline{2}}(\frac{1}{t})}{\left(Q(\frac{t}{2})\right)^b} = 2\sum_{k=0}^{\infty} \frac{2^{k+1} tL^{\underline{1}}(\frac{1}{2^{k+1}t})}{\left(Q(\frac{2^{k+1}t}{2})\right)^b}. \tag{66}$$

Both sums are of the form $\sum_{k\geq 0}\lambda_k f(\mu_k x)$ for some function $f(\cdot)$ and sequences λ_k, μ_k; that is, they are the so-called *harmonic sums* (cf. [9]). It is well known that the Mellin transform of such a sum is $F(s)\sum_{k\geq 0}\lambda_k\mu_k^{-s}$ (where $F(s)$ is the Mellin transform of f). In our case, we have

$$\mathcal{M}\left[\frac{tL^1(\frac{1}{t})}{Q^b(\frac{t}{2})};s\right] = \frac{1}{1-2^{-s}}I(s),$$
$$\mathcal{M}\left[\frac{tL^2(\frac{1}{t})}{Q^b(\frac{t}{2})};s\right] = \frac{2^{1-s}}{(1-2^{-s})^2}I(s),$$

where

$$I(s) = \int_0^\infty \frac{t^{s-1}}{Q^b(t)}dt = \frac{\pi}{\sin\pi s}J(s), \tag{67}$$

$$J(s) = \frac{1}{2\pi i}\int_{\mathcal{H}} \frac{(-t)^{s-1}}{Q^b(t)}dt \tag{68}$$

with $\mathcal{H}$ being the Hankel contour (cf. [9, 12]).

The rest is easy. Applying standard arguments of the inverse Mellin transform we can derive asymptotic expansions of $L^1(\frac{1}{t})$ and $L^2(\frac{1}{t})$ as $t\to 0$. We find

$$L^1(\frac{1}{t}) = \frac{1}{t}k(t) + bk(t) + O(t\log t^{-1}),$$
$$L^2(\frac{1}{t}) = \frac{1}{t}K(t) + bK(t) + O(t\log^2 t^{-1}),$$

where

$$k(t) = \frac{1}{L}\log\frac{1}{t} + \frac{1}{2} + \frac{J'(0)}{L} - \frac{1}{L}\sum_{k=0}\frac{I(s_k)}{s_k}t^{-s_k},$$
$$K(t) = \frac{1}{L^2}\log^2\frac{1}{t} + \frac{2J'(0)}{L^2}\log\frac{1}{t} - \left(\frac{1}{6} + \frac{J''(0)}{L^2} - \frac{\pi^2}{3L^2}\right) + 8bt$$
$$- \frac{2}{L^2}\sum_{k=0}\frac{I(s_k)}{s_k}t^{-s_k}\log\frac{1}{t} + \frac{2}{L^2}\sum_{k=0}\left(\frac{I(s_k)}{s_k^2} - \frac{I'(s_k)}{s_k}\right)t^{-s_k},$$

with $s_k = 2\pi ik/\log 2$ for $k = 0, \pm 1, \ldots$ being the roots of $1-2^{-s}=0$, and $L = \log 2$. Finally, applying the *singularity analysis* of Flajolet and Odlyzko [7], after somewhat tedious algebra we prove (15).

4.2. Exact and limiting distribution. We need another approach to establish exact and asymptotic distributions in the symmetric case since, as shown above, Var $D_m = O(1)$. We also point out that—even if it is possible in principle—using recurrence (5) or functional equation (6) may be quite troublesome. Therefore we devised another, more combinatorial and probabilistic approach.

Let us fix $j \geq 1$, and consider a *particular path*, say, $\mathcal{P}$, from the root to a node at level j on $\mathcal{P}$. Let $T_{j,r}$ be the number of strings needed to be added to the tree (after the first b) to ensure that a node at level j contains exactly r strings ($1 \leq r \leq b$). Since the first b strings are stored in the root, we observe the following:

$\Pr\{T_{j,r} \leq m-b\} =$

$\Pr\{$node at level j contains at least r strings when m strings are in the tree$\}$.

Note that $P[j,r] := \Pr\{\text{exactly } r \text{ strings are in a node at level } j \text{ when } m \text{ strings are added}\} = \Pr\{T_{j,r} \le m-b\} - \Pr\{T_{j,r+1} \le m-b\}$.

Then the distribution of D_m can be computed as

$$\Pr\{D_m = j\} = \frac{2^j}{m} \sum_{r=1}^{b} P[j,r] \cdot r = \frac{2^j}{m} \sum_{r=1}^{b} \Pr\{T_{j,r} \le m-b\}. \tag{69}$$

In view of the above equation, to compute the exact distribution of D_m one needs the distribution of $T_{r,j}$. But the number of strings, say, X_i, that one must insert into the tree in order to fill up a node at level $i < j$ on the path $\mathcal{P}$ (when the node on $\mathcal{P}$ at level $i-1$ is full) is distributed as the sum of b independent random variables geometrically distributed with success probability $\pi(i) = 2^{-i}$. Let $X_i(z) = Ez^{X_i}$ be the probability generating function. Then

$$X_i(z) = \left(\frac{\pi(i)z}{1-(1-\pi(i))z} \right)^b \quad \text{for} \quad i < j.$$

Similarly, the probability generating function for the number of strings needed to get exactly r strings in a given node at level j (when the node on $\mathcal{P}$ at level $j-1$ is full) is given by

$$X_j(z) = \left(\frac{\pi(j)z}{1-(1-\pi(j))z} \right)^r.$$

Summing up, the probability generating function $T_{j,r}(z)$ of $T_{j,r}$ is

$$T_{j,r}(z) = X_j(z) \prod_{i=1}^{j-1} X_i(z). \tag{70}$$

To compute the required probabilities, we first use the Cauchy formula,

$$\Pr\{T_{j,r} = \ell\} = \frac{1}{2\pi i} \oint \frac{T_{j,r}(z)}{z^{\ell+1}}\, dz,$$

and then the residue theorem. The calculations are rather straightforward but quite tedious. We find

$$\begin{aligned} m\Pr\{D_m \le j\} = b - \frac{1}{(b-1)!} \sum_{k=1}^{j} \left(\frac{\pi(k)}{\pi(k)-1} \right)^b \frac{\partial^{b-1}}{\partial z^{(b-1)}} \Bigg\{ \frac{z^{2b}}{(z-1)^2} (z^{-b} - z^{-m}) \\ \times \prod_{v=1, v\ne k}^{j} \left(\frac{\pi(v)z}{1-(1-\pi(v))z} \right)^b \Bigg\}_{z=z^*(k)}, \end{aligned} \tag{71}$$

where $z^*(k) = (1-\pi(k))^{-1}$. This leads to (18) for the exact distribution for D_m.

The asymptotic formula of part (iii) of Theorem 2.2 follows from the above after some algebra that we summarize below. We set throughout this derivation $j = \log_2 m + \eta$, with $\eta = O(1)$, and $k = j + O(1)$, which we justify. After substituting $\eta = \kappa - \{\log_2 m\}$, we prove part (iii) of Theorem 2.2.

Let us now analyze (71). The term involving z^{-m} in (71) becomes

$$H_1 := \frac{\pi^b(k)}{m(b-1)!(\pi(k)-1)^b} \frac{\partial^{b-1}}{\partial z^{b-1}} \left(z^{-(m-2b)} (z-1)^{-2} \varphi_1(z) \varphi_2(z) \right) |_{z=z^*(k)},$$

where

$$\varphi_1(z) = \prod_{v=1}^{k-1} \left(\frac{\pi(v)z}{1-(1-\pi(v))z} \right)^b,$$
$$\varphi_2(z) = \prod_{v=k+1}^{j} \left(\frac{\pi(v)z}{1-(1-\pi(v))z} \right)^b.$$

After using Leibniz's rule for differentiation, we obtain

$$\sum_{\ell+s+s_1+s_2=b-1} \binom{b-1}{\ell, s, s_1, s_2} \left(\frac{(-1)^{\ell+s-b}\pi^b(k)(m-2b)_\ell(-1)^s(s+1)!(1-\pi(k))^{m-3b+\ell+2+s}}{(m(b-1)!\pi(k))^{2+s}} \right)$$

$$\times \left(\varphi_1^{(s_1)}(z)\varphi_2^{(s_2)}(z) \right) |_{z^*(k)}, \tag{72}$$

where $f^{(k)}(z)$ denotes the kth derivative of $f(z)$, and $(m)_\ell = m(m-1)\cdots(m-\ell+1)$.

Now let $j = \log_2 m + \eta$ and $i = j - k$ where $i = O(1)$. We obtain

$$\frac{(\pi(k))^{\ell-1}(m-2b)_\ell}{m} \sim (m\pi(k))^{\ell-1} = \frac{2^{i(\ell-1)}}{2^{\eta(\ell-1)}} \tag{73}$$

and

$$(1-\pi(k))^{m-3b+\ell+2+s} \sim e^{-2^{-(\eta-i)}}.$$

To compute the derivatives of $\varphi_1(z)$ and $\varphi_2(z)$, we observe, for example, that for any integer r,

$$Y := \frac{\partial^r}{\partial z^r} \left(\frac{\pi(v)}{1-(1-\pi(v))z} \right)^b = \frac{(\pi(v))^b(1-\pi(v))^r(b)_r}{1-(1-\pi(v)z)^{b+r}}.$$

Setting $z = z^*(k)$ and $v = k + O(1)$, we find

$$Y \sim \frac{(b)_r}{(\pi(k))^r(1-2^{-u})^{b+r}2^{ur}} \qquad \text{in the } \varphi_1 \text{ case,} \tag{74}$$

where $u = k - v > 0$, and

$$Y \sim \frac{(b)_r 2^{ur}}{(\pi(k))^r(1-2^u)^{b+r}} \qquad \text{in the } \varphi_2 \text{ case.} \tag{75}$$

To deal with expressions like (74) or (75), we define

$$H(s) = \frac{\partial^s}{\partial z^s} \prod_{k=1}^{\infty} \left(\frac{1}{1-\pi(k)z} \right)^b |_{z=1},$$

and with $R(0,s) = -1$,

$$R(i,s) = -\frac{\partial^s}{\partial z^s} \prod_{k=1}^{i} \left(\frac{1}{1-\frac{z}{\pi(k)}} \right)^b \Bigg|_{z=1},$$

which are (13) and (14) from section 2.

These expressions are the b-equivalent of $Q^{-1}(t)$ (cf. (12)) and function $|R_i|$ used in [24, 26] (cf. (30) of [26]) parametrized by s. Clearly $R(i,s)$ decreases exponentially with i and $H(s)$ is uniformly bounded, which justifies our choice $k = j + O(1)$ for asymptotic analysis. Moreover, any term $(1-\pi(v))^r$ $(v < k)$ leads to a contribution $(1-2^{u-k})^r 2^{-ur}(1-2^{-u})^{-r}$. The sum of all these contributions is $O(1)$, which shows that we can asymptotically take $(1-\pi(v)) \sim 1$.

Let us return to (71). We can extract a term $(\pi(k))^{b-2-s-s_1-s_2} = (\pi(k))^{\ell-1}$ and, with (73), after summing over k we obtain

$$H_1 \sim \sum_{l+s+s_1+s_2=b-1} \frac{(-1)^{s_1+s_2}}{l!s_1!s_2!} \sum_{i=0}^{\infty} (s+1) R(i,s_2) H(s_1) \frac{2^{i(\ell-1)}}{2^{\eta(\ell-1)}} e^{-2^{-(\eta-i)}}.$$

Similar analysis is valid for the term at z^{-b} of (71). Finally, after substituting $\eta = \kappa - \{\log_2 m\}$, we prove part (iii) of Theorem 2.2, which completes the proof of Theorem 2.2.

Acknowledgment. We thank Philippe Jacquet (INRIA, France) and Helmut Prodinger (TU Wien) for many valuable comments regarding this research. We are particularly obliged to one of the referees whose very careful reading of the paper allowed us to eliminate some inaccuracies and led to a better presentation of our results.

REFERENCES

[1] D. Aldous and P. Shields, *A diffusion limit for a class of random-growing binary trees*, Probab. Theory Related Fields, 79 (1988), pp. 509–542.
[2] P. Billingsley, *Convergence of Probability Measures*, John Wiley & Sons, New York, 1968.
[3] T.M. Cover and J.A. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, 1991.
[4] B. Davies, *Integral Transforms and Their Applications*, Springer-Verlag, New York, 1985.
[5] G. Doetsch, *Handbuch der Laplace Transformation*, Birkhäuser Verlag, Basel, 1950.
[6] P. Flajolet and R. Sedgewick, *Digital search trees revisited*, SIAM J. Comput., 15 (1986), pp. 748–767.
[7] P. Flajolet and A. Odlyzko, *Singularity analysis of generating functions*, SIAM J. Disc. Meth., 3 (1990), pp. 216–240.
[8] P. Flajolet and B. Richmond, *Generalized digital trees and their difference—differential equations*, Random Structures Algorithms, 3 (1992), pp. 305–320.
[9] P. Flajolet, X. Gourdon, and P. Dumas, *Mellin transforms and asymptotics: Harmonic sums*, Theoret. Comput. Sci., 144 (1995), pp. 3–58.
[10] P. Flamant, *Sur une equation différentielle fonctionelle linéaire*, Rend. Circ. Mat. Palermo, XLVIII (1924), pp. 135–208.
[11] E. Gilbert and T. Kadota, *The Lempel–Ziv algorithm and message complexity*, IEEE Trans. Inform. Theory, 38 (1992), pp. 1839–1842.
[12] F. Hubalek, *Beiträge zur Analyse Verallgemeinerter Digitaler Suchbäume*, Ph.D. thesis, Technische Universität Wien, Vienna, 1994.
[13] K. Hummelsheim and C. Kleiner, *Project in CS* 543: *Analysis of a data compression algorithm*, Department of Computer Science, Purdue University, West Lafayette, IN, 1996.
[14] P. Jacquet and W. Szpankowski, *Analysis of digital trees with Markovian dependency*, IEEE Trans. Inform. Theory, 37 (1991), pp. 1470–1475.
[15] P. Jacquet and W. Szpankowski, *Asymptotic behavior of the Lempel–Ziv parsing scheme and digital search trees*, Theoret. Comput. Sci., 144 (1995), pp. 161–197.
[16] P. Jacquet and W. Szpankowski, *Analytical depoissonization lemma and its applications*, Theoret. Comput. Sci., 201 (1998), pp. 1–62.
[17] P. Jacquet and W. Szpankowski, *Entropy computations via analytic depoissonization,*, IEEE Trans. Inform. Theory, 1999, to appear.
[18] S. Janson and W. Szpankowski, *Analysis of an asymmetric leader election algorithm*, Electron. J. Combin., 4 (1997), Research Paper 17, (electronic).

[19] P. Kirschenhofer, H. Prodinger, and W. Szpankowski, *Digital search trees again revisited: The internal path length perspective*, SIAM J. Comput., 23 (1994), pp. 598–616.
[20] D. Knuth, *The Art of Computer Programming. Sorting and Searching.* Vol. 3, Addison-Wesley, Reading, MA, 1973.
[21] A. Konheim and D.J. Newman, *A note on growing binary trees*, Discrete Math., 4 (1973), pp. 57–63.
[22] A. Lempel and J. Ziv, *A universal algorithm for sequential data compression*, IEEE Trans. Inform. Theory, 23 (1977), pp. 337–343.
[23] A. Lempel and J. Ziv, *Compression of individual sequences via variable-rate coding*, IEEE Trans. Inform. Theory, 24 (1978), pp. 530–536.
[24] G. Louchard, *Exact and asymptotic distributions in digital and binary search trees*, RAIRO Theoretical Inform. Appl., 21 (1987), pp. 479–495.
[25] G. Louchard, *Digital search trees revisited*, Cahiers Centre Études Rech. Oper., 36 (1995), pp. 259–278.
[26] G. Louchard and W. Szpankowski, *Average profile and limiting distribution for a phrase size in the Lempel–Ziv parsing algorithm*, IEEE Trans. Inform. Theory, 41 (1995), pp. 478–488.
[27] G. Louchard and W. Szpankowski, *Generalized Lempel–Ziv parsing scheme and its preliminary analysis of the average profile*, in Proc. Data Compression Conference, Snowbird, UT, 1995, pp. 262–271.
[28] G. Louchard and W. Szpankowski, *On the average redundancy rate of the Lempel–Ziv code*, IEEE Trans. Inform. Theory, 43 (1997), pp. 2–8.
[29] H. Mahmoud, *Evolution of Random Search Trees*, John Wiley & Sons, New York, 1992.
[30] D. Ornstein and B. Weiss, *Entropy and data compression schemes*, IEEE Trans. Inform. Theory, 39 (1993), pp. 78–83.
[31] B. Pittel, *Asymptotic growth of a class of random trees*, Ann. Probab., 13 (1985), pp. 414–427.
[32] H. Prodinger, *Approximate counting via Euler transform*, Math. Slovaca, 44 (1994), pp. 569–574.
[33] H. Prodinger, *Digital search trees and basic hypergeometric functions*, EATCS Bulletin, 56 (1995), pp. 112–115.
[34] B. Rais, P. Jacquet, and W. Szpankowski, *A limiting distribution for the depth in PATRICIA tries*, SIAM J. Discrete Math., 6 (1993), pp. 197–213.
[35] S. Savari, *Redundancy of the Lempel–Ziv incremental parsing rule*, IEEE Trans. Information Theory, 43 (1997), pp. 9–21.
[36] J. Storer, *Data Compression: Methods and Theory*, Computer Science Press, Rockville, MD, 1988.
[37] W. Szpankowski, *The evaluation of an alternating sum with applications to the analysis of some data structures*, Inform. Process. Lett., 28 (1988), pp. 13–19.
[38] W. Szpankowski, *A characterization of digital search trees from the successful search viewpoint*, Theoret. Comput. Sci., 85 (1991), pp. 117–134.
[39] W. Szpankowski, *A generalized suffix tree and its (un)expected asymptotic behaviors*, SIAM J. Comput., 22 (1993), pp. 1176–1198.
[40] J. Tang, *Probabilistic Analysis of Digital Search Trees*, Ph.D. thesis, Purdue University, West Lafayette, IN, 1996.
[41] A. Wyner and J. Ziv, *Some asymptotic properties of the entropy of a stationary ergodic data source with applications to data compression*, IEEE Trans. Inform. Theory, 35 (1989), pp. 1250–1258.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 935–954

THE MAXIMUM PARTITION MATCHING PROBLEM WITH APPLICATIONS*

CHI-CHANG CHEN† AND JIANER CHEN‡

Abstract. Let $\mathcal{S} = \{C_1, C_2, \ldots, C_k\}$ be a collection of pairwise disjoint subsets of $U = \{1, 2, \ldots, n\}$ such that $\bigcup_{i=1}^{k} C_i = U$. A *partition matching* of $\mathcal{S}$ consists of two subsets $\{a_1, \ldots, a_m\}$ and $\{b_1, \ldots, b_m\}$ of U together with a sequence of distinct partitions of $\mathcal{S}$: $(\mathcal{A}_1, \mathcal{B}_1), \ldots, (\mathcal{A}_m, \mathcal{B}_m)$ such that a_i is contained in a subset in the collection $\mathcal{A}_i$ and b_i is contained in a subset in the collection $\mathcal{B}_i$ for all $i = 1, \ldots, m$. An efficient algorithm is developed that constructs a maximum partition matching for a given collection $\mathcal{S}$. The algorithm can be used to construct optimal parallel routing between two nodes in interconnection networks.

Key words. maximum matching, greedy algorithm, star network, parallel routing

AMS subject classifications. 05A18, 05D15, 68M07, 68M10, 68Q25, 68R05

PII. S009753979630012X

1. Introduction.

Matching is one of the most extensively studied areas in computer science, since it is interesting from a combinatorial point of view and has wide applications as well. Examples of matching are the maximum graph matching problem [15], maximum graph adjacency matching problem [7], stable marriage problem [10], and three-dimensional matching problem [9].

In this paper, we introduce a new maximum matching problem, study its computational complexity, and demonstrate its applications in interconnection networks. Let $\mathcal{S} = \{C_1, C_2, \ldots, C_k\}$ be a collection of subsets of the universal set $U = \{1, 2, \ldots, n\}$ such that $\bigcup_{i=1}^{k} C_i = U$ and $C_i \cap C_j = \phi$ for all $i \neq j$. A *partition matching* (of order m) of $\mathcal{S}$ consists of two ordered subsets $L = \{a_1, a_2, \ldots, a_m\}$ and $R = \{b_1, b_2, \ldots, b_m\}$ of m elements of U (the subsets L and R may not be disjoint), together with a sequence of m distinct partitions of $\mathcal{S}$: $(\mathcal{A}_1, \mathcal{B}_1), (\mathcal{A}_2, \mathcal{B}_2), \ldots, (\mathcal{A}_m, \mathcal{B}_m)$ such that for all $i = 1, \ldots, m$, a_i is contained in a subset in the collection $\mathcal{A}_i$ and b_i is contained in a subset in the collection $\mathcal{B}_i$. The *maximum partition matching* problem is to construct a partition matching of order m for a given collection $\mathcal{S}$ with m maximized.

The maximum partition matching problem can be formulated in terms of the three-dimensional matching problem as follows: given an instance $\mathcal{S} = \{C_1, C_2, \ldots, C_k\}$ of the maximum partition matching problem, we construct an instance M for the three-dimensional matching problem such that a triple (a, b, P) is contained in M if and only if a and b are elements in U, and $P = (\mathcal{A}, \mathcal{B})$ is a partition of $\mathcal{S}$ such that a is contained in a set in $\mathcal{A}$ and b is contained in a set in $\mathcal{B}$. Unfortunately, the number of partitions of the collection $\mathcal{S}$ can be as large as 2^n; thus the above reduction is not polynomial-time bounded. Moreover, the three-dimensional matching problem is NP-hard [9].

* Received by the editors March 1, 1996; accepted for publication (in revised form) June 17, 1997; published electronically January 29, 1999.

http://www.siam.org/journals/sicomp/28-3/30012.html

†Information Engineering Department, I-Shou University, Kaohsiung County, Taiwan, Republic of China (ccchen@mail.isu.edu.tw). The research of this author was supported in part by an Engineering Excellence Award from Texas A&M University, College Station, TX.

‡Department of Computer Science, Texas A&M University, College Station, TX 77843-3112 (chen@cs.tamu.edu). The research of this author was supported in part by the National Science Foundation under grants CCR-9110824 and CCR-9613805.

We will present an algorithm of running time $O(n^2 \log n)$ that solves the maximum partition matching problem. We first show that when the number of subsets in the collection $\mathcal{S}$ is sufficiently large, a maximum partition matching can be constructed from a simpler "prematching" on the elements in U. For the case that the number of subsets in the collection $\mathcal{S}$ is small, we develop a greedy algorithm that uses a "chain justification" technique and finds a maximum partition matching. A sophisticated combinatorial analysis is given to prove the correctness of the algorithm.

The maximum partition matching problem arises in connection with the parallel routing problem in interconnection networks. We show how the above algorithm can be applied to construct an optimal parallel routing in star networks, which have received considerable attention recently and have been shown to be an attractive alternative to the widely used hypercube networks [1]. In particular, we present an efficient algorithm that constructs between two arbitrary nodes in the n-dimensional star network a maximum number of node-disjoint *shortest* paths, which can be further used to construct between the two nodes $n-1$ node-disjoint paths of minimum bulk length (the *bulk length* of $n-1$ node-disjoint paths between two nodes is defined to be the length of the longest path among the $n-1$ paths). Note that these two problems on general graphs are NP-hard [8, 9]. These results significantly improve the previous parallel routing results in star networks [6, 11, 13, 14].

We introduce necessary terminology that is used in the rest of our discussion.

Let $M = \langle L, R, (\mathcal{A}_1, \mathcal{B}_1), \ldots, (\mathcal{A}_m, \mathcal{B}_m)\rangle$ be a partition matching of the collection $\mathcal{S}$, where $L = \{a_1, \ldots, a_m\}$ and $R = \{b_1, \ldots, b_m\}$. We will say that the partition $(\mathcal{A}_i, \mathcal{B}_i)$ *left-pairs* the element a_i and *right-pairs* the element b_i. An element a is said to be *left-paired* if it is in the set L. Otherwise, the element a is *left-unpaired*. Similarly, we define *right-paired* and *right-unpaired* elements. The collections $\mathcal{A}_i$ and $\mathcal{B}_i$ are called the *left-collection* and *right-collection* of the partition $(\mathcal{A}_i, \mathcal{B}_i)$. The partition matching M may also be written as $M[(a_1, b_1), \ldots, (a_m, b_m)]$ if the partitions $(\mathcal{A}_1, \mathcal{B}_1), \ldots, (\mathcal{A}_m, \mathcal{B}_m)$ are implied. The partition matching $M[(a_1, b_1), \ldots, (a_m, b_m)]$ is *maximum* if m is the largest among all partition matchings of $\mathcal{S}$.

A permutation $u = a_1 a_2 \cdots a_n$ of the elements in the set $U = \{1, 2, \ldots, n\}$ can be given by a product of disjoint cycles [2], which will be called the *cycle structure* of the permutation. A $\pi[1, i]$ *transposition* on u is to exchange the positions of a_1 and a_i in u. More specifically, $\pi[1, i](u) = a_i a_2 a_3 \cdots a_{i-1} a_1 a_{i+1} \cdots a_n$. It is sometimes more convenient to write the transposition $\pi[1, i](u)$ as $\pi[a_i](u)$ to indicate that the transposition exchanges the position of the first symbol and the symbol a_i in u. Let us consider how a transposition changes the cycle structure of a permutation. Write u in its cycle structure

$$u = (a_1^{(1)} \cdots a_{n_1}^{(1)}\ 1)(a_1^{(2)} \cdots a_{n_2}^{(2)}) \cdots (a_1^{(k)} \cdots a_{n_k}^{(k)}).$$

Now suppose we apply the transposition $\pi[1, i]$ on u. There are two cases.

If a_i is not in the cycle containing the symbol 1, then $\pi[1, i]$ "merges" the cycle containing 1 with the cycle containing a_i. More precisely, suppose that $a_i = a_1^{(2)}$ (note that each cycle can be cyclically permuted and the order of the cycles is not important). Then the permutation $\pi[1, i](u)$ will have the cycle structure

$$\pi[1, i](u) = (a_1^{(2)} \cdots a_{n_2}^{(2)} a_1^{(1)} \cdots a_{n_1}^{(1)}\ 1)(a_1^{(3)} \cdots a_{n_3}^{(3)}) \cdots (a_1^{(k)} \cdots a_{n_k}^{(k)}).$$

If a_i is in the cycle containing the symbol 1, then $\pi[1, i]$ "splits" the cycle. More precisely, suppose that $a_i = a_j^{(1)}$, $j > 1$ (note that $a_1^{(1)} = a_1$ and we assume $i > 1$).

Algorithm. PREMATCHING

INPUT: The collection $\mathcal{S} = \{C_1, \ldots, C_k\}$ of subsets of U, with $2^k \geq 2n$.
OUTPUT: A partition matching M in $\mathcal{S}$.

1. $\mathcal{T} = \mathcal{S}$;
2. **while** $\mathcal{T}$ contains more than one set
   but does not consist of exactly three singular sets **do**
   Pick two sets C and C' of largest cardinality in $\mathcal{T}$;
   Pick an element a in C and an element a' in C';
   Pick two unused partitions of $\mathcal{S}$ of the forms $(\mathcal{A} \cup \{C\}, \mathcal{B} \cup \{C'\})$, and $(\mathcal{B} \cup \{C'\}, \mathcal{A} \cup \{C\})$, where $\mathcal{A} \cup \mathcal{B} = \mathcal{S} - \{C, C'\}$;
   Use the partition $(\mathcal{A} \cup \{C\}, \mathcal{B} \cup \{C'\})$ to left-pair a and right-pair a';
   Use the partition $(\mathcal{B} \cup \{C'\}, \mathcal{A} \cup \{C\})$ to left-pair a' and right-pair a;
   $C = C - \{a\}$; $C' = C' - \{a'\}$;
   If any of the sets C and C' is an empty set now, then delete it from the collection $\mathcal{T}$;
3. **if** $\mathcal{T}$ consists of exactly three singular sets $C_1 = \{a_1\}$, $C_2 = \{a_2\}$, and $C_3 = \{a_3\}$
   then Pick three unused partitions of $\mathcal{S}$ of the forms $(\mathcal{A}_1 \cup \{C_1\}, \mathcal{B}_1 \cup \{C_2\})$, $(\mathcal{A}_2 \cup \{C_2\}, \mathcal{B}_2 \cup \{C_3\})$, and $(\mathcal{A}_3 \cup \{C_3\}, \mathcal{B}_3 \cup \{C_1\})$ to left-pair a_1 and right-pair a_2, left-pair a_2 and right-pair a_3, and left-pair a_3 and right-pair a_1, respectively.

FIG. 2.1. *The algorithm* PREMATCHING.

Then $\pi[1, i](u)$ will have the cycle structure

$$\pi[1, i](u) = (a_1^{(1)} \cdots a_{j-1}^{(1)})(a_j^{(1)} \cdots a_{n_1}^{(1)} 1)(a_1^{(2)} \cdots a_{n_2}^{(2)}) \cdots (a_1^{(k)} \cdots a_{n_k}^{(k)}).$$

Note that if a symbol a_i is in a single symbol cycle in a cycle structure of a permutation $u = a_1 a_1 \cdots a_n$, then the symbol is in its "correct" position, i.e., $a_i = i$, and that if a symbol is in a cycle containing more than one symbol, then the symbol is not in its correct position. Denote by ε the identity permutation $\varepsilon = 12 \cdots n$.

2. Partition matching via element prematching. For the rest of this paper, we assume that $U = \{1, 2, \ldots, n\}$ and that $\mathcal{S} = \{C_1, C_2, \ldots, C_k\}$ is a collection of pairwise disjoint subsets of U such that $\bigcup_{i=1}^{k} C_i = U$.

A necessary condition for two subsets $\{a_1, a_2, \ldots, a_m\}$ and $\{b_1, b_2, \ldots, b_m\}$ of U to form a partition matching for the collection $\mathcal{S}$ is that a_i and b_i belong to different subsets in the collection $\mathcal{S}$ for all $i = 1, 2, \ldots, m$. This motivates the following definition.

DEFINITION 2.1. *Two subsets* $\{a_1, a_2, \ldots, a_m\}$ *and* $\{b_1, b_2, \ldots, b_m\}$ *of* U *form an* element prematching $P = \{(a_i, b_i) \mid 1 \leq i \leq m\}$ *if* a_i *and* b_i *do not belong to the same subset in the collection* $\mathcal{S}$ *for all* $i = 1, 2, \ldots, m$. *The element prematching* P *is* maximum *if* m *is the largest among all element prematchings of* $\mathcal{S}$.

The following lemma follows directly from Definition 2.1.

LEMMA 2.2. *Let* $\{a_1, a_2, \ldots, a_m\}$ *and* $\{b_1, b_2, \ldots, b_m\}$ *be a maximum element prematching of* $\mathcal{S}$. *If the subsets* $\{a_1, a_2, \ldots, a_m\}$ *and* $\{b_1, b_2, \ldots, b_m\}$ *also form a partition matching* M *of* $\mathcal{S}$, *then* M *is a maximum partition matching.*

We show in this section that when the number of subsets in the collection $\mathcal{S}$ is sufficiently large, a maximum partition matching in $\mathcal{S}$ can be constructed via a maximum element prematching. Consider the algorithm PREMATCHING in Figure 2.1. We say that a set is *singular* if it consists of a single element.

The rest of this section is for a proof of correctness and complexity analysis of the algorithm PREMATCHING.

LEMMA 2.3. *Let $P = \{(a_i, b_i) \mid 1 \le i \le m\}$ be the set of pairs constructed by the algorithm* PREMATCHING. *Then P forms a maximum element prematching for the collection $\mathcal{S}$.*

Proof. Since for each pair (a_i, b_i) constructed by the algorithm PREMATCHING, the elements a_i and b_i are from different sets in $\mathcal{S}$, the set P forms an element prematching. We show that this element prematching P is maximum.

Denote by $|C_i|$ the cardinality of the set C_i. Without loss of generality, we assume $|C_1| \ge |C_2| \ge \cdots \ge |C_k|$. There are two cases.

Case 1. $|C_1| > n/2$. Because in an element prematching each pair (a, b) must have either $a \in \cup_{i=2}^{k} C_i$ or $b \in \cup_{i=2}^{k} C_i$, a maximum element prematching can have at most $2\sum_{i=2}^{k} |C_i|$ pairs.

On the other hand, in case $|C_1| > n/2$ the algorithm PREMATCHING actually constructs $2\sum_{i=2}^{k} |C_i|$ pairs: Since $|C_1| > \sum_{i=2}^{k} |C_i|$, in each execution of the body of the **while** loop in step 2 the set C_1 always has the largest cardinality. Thus, the algorithm always picks an element a in C_1 and an element b in $\bigcup_{i=2}^{k} C_i$ and then makes the pairs (a, b) and (b, a). The loop will stop when all elements in $\bigcup_{i=2}^{k} C_i$ have been used, which results in exactly $2\sum_{i=2}^{k} |C_i|$ pairs. Thus, the lemma is true for this case.

Case 2. $|C_1| \le n/2$. In this case we show that the algorithm PREMATCHING always constructs an element prematching with n pairs (thus maximum).

If $\mathcal{S}$ consists of only two sets C_1 and C_2, then since $|C_1| \ge |C_2|$, $|C_1| \le n/2$, and $|C_1| + |C_2| = n$, we must have $|C_1| = |C_2|$. Now it becomes trivial to verify in this case that the algorithm PREMATCHING constructs a maximum element prematching with n pairs.

Thus we assume that the collection $\mathcal{S}$ contains at least three sets. We prove the lemma for this case by induction on the size n of the universal set U. When $n = 3$, the collection $\mathcal{S}$ consists of exactly three singular sets, and step 3 of the algorithm shows how an element prematching with three pairs can be constructed. Now assume that $n > 3$. Note that after the first execution of the body of the **while** loop in step 2, the collection $\mathcal{T}$ becomes

$$\mathcal{T}' = \{C_1', C_2', C_3, \ldots, C_k\},$$

where $C_1' = C_1 - \{a\}$, $C_2' = C_2 - \{a'\}$, and $|C_1'| + |C_2'| + |C_3| + \cdots + |C_k| = n - 2$. We show that the largest set in $\mathcal{T}'$ contains at most $(n-2)/2$ elements.

If C_1' is still the largest set, then from $|C_1| \le n/2$ we have $|C_1| \le \sum_{i=2}^{k} |C_i|$. Consequently, $|C_1'| \le |C_2'| + |C_3| + \cdots + |C_k|$, i.e., $|C_1'| \le (n-2)/2$. On the other hand, if C_1' is no longer the largest set, then C_3 must be the largest set in $\mathcal{T}'$, and before the first execution of the body of the **while** loop in step 2 we have $|C_1| = |C_2| = |C_3|$. Thus, $|C_3| \le n/3$. Note that $|C_3|$ is an integer; thus $|C_3| \le n/3$ implies $|C_3| \le (n-2)/2$ for all $n > 3$.

Thus, the collection $\mathcal{T}'$ consists of k subsets of the universal set $U - \{a, a'\}$ of $n - 2$ elements, and the largest set in $\mathcal{T}'$ contains no more than $(n-2)/2$ elements. Note that the algorithm PREMATCHING applies the same strategy on the collection $\mathcal{T}'$. By the inductive hypothesis, the algorithm constructs an element prematching P' with $n - 2$ pairs for the collection $\mathcal{T}'$. Combining this with the pairs (a, a') and (a', a) constructed in the first execution of the body of the **while** loop in step 2, we obtain an element prematching with n pairs for the collection $\mathcal{S}$.

Combining all these analyses, we conclude with the lemma. ∎

To complete the correctness proof for the algorithm PREMATCHING, we need only to show that for each pair (a, a') constructed by the algorithm, there is always a distinct partition of $\mathcal{S}$ that implements the pairing. By the assumption of the algorithm, we have $2^k \geq 2n$.

We first consider step 2 of the algorithm PREMATCHING. Suppose that at some moment the algorithm needs a partition to left-pair an element $a \in C$ and right-pair an element $a' \in C'$. Note that each execution of the body of the **while** loop uses two partitions of the forms $(\mathcal{A}, \mathcal{B})$ and $(\mathcal{B}, \mathcal{A})$. Thus each execution of the body of the **while** loop can use at most one partition whose left-collection contains C and whose right-collection contains C'. Consequently, less than $\lfloor n/2 \rfloor$ partitions whose left-collection contains C and whose right-collection contains C' have been used. Since $2^k \geq 2n$ and there are $2^{k-2} \geq n/2$ partitions of $\mathcal{S}$ whose left-collection contains C and whose right-collection contains C', we conclude that there is always an unused partition P that can be used to left-pair the element a and right-pair the element a'.

The proof proceeds similarly for step 3. For example, suppose we want to left-pair the element $a_3 \in C_3$ and right-pair the element $a_1 \in C_1$. There are $2^{k-2} \geq n/2$ total partitions of $\mathcal{S}$ whose left-collection contains C_3 and whose right-collection contains C_1, of which at most $(n-3)/2$ have been used (by step 2). Also note that no partition that is used to left-pair a_1 and right-pair a_2 or left-pair a_2 and right-pair a_3 in step 3 has C_3 in its left-collection and C_1 in its right-collection. Therefore, there is always an unused partition that can be used to left-pair a_3 and right-pair a_1.

This shows that the algorithm PREMATCHING constructs a partition matching in the collection $\mathcal{S}$. Combining this with Lemmas 2.2 and 2.3, we have the following theorem.

THEOREM 2.4. *Let $\mathcal{S} = \{C_1, C_2, \ldots, C_k\}$ be a collection of nonempty subsets of the universal set $U = \{1, 2, \ldots, n\}$ such that $\bigcup_{i=1}^{k} C_i = U$ and $C_i \cap C_j = \phi$ for $i \neq j$. If $2^k \geq 2n$, then the algorithm* PREMATCHING *constructs a maximum partition matching in $\mathcal{S}$.*

The algorithm PREMATCHING can be implemented to have running time $O(n^2)$. For this, we represent each partition of $\mathcal{S}$ by a binary number of k bits and assume that simple arithmetic operations on k-bit binary numbers take constant time. With this representation, testing whether a set C_i is in the left-collection or in the right-collection of a partition P takes constant time. We keep a list $\mathcal{L}$ for the used partitions, sorted by their k-bit binary representations. When a pair (C_i, C_j) of sets is given and we need to find a partition whose left-collection contains C_i and whose right-collection contains C_j (this kind of partitions will be called "partitions pairing (C_i, C_j)" in the following description), we go through the list $\mathcal{L}$ to identify all used partitions pairing (C_i, C_j) in $\mathcal{L}$. Let $\mathcal{L}'$ be the (sorted) sublist containing all used partitions pairing (C_i, C_j) in $\mathcal{L}$. Note that given a partition P pairing (C_i, C_j), we can construct the "next" partition pairing (C_i, C_j) by a special "adding 1" operation on P; this adding 1 operation first deletes the ith bit b_i and the jth bit b_j from P, then adds 1 to the resulting number, and finally reinserts the bits b_i and b_j back into the ith and jth positions, respectively. The adding 1 operation can be implemented by a constant number of arithmetic operations; thus it takes constant time. Therefore, by going through the sublist $\mathcal{L}'$ we can find a "gap" between two consecutive partitions P_1 and P_2 in $\mathcal{L}'$; i.e., P_2 is not the partition pairing (C_i, C_j) next to the partition P_1, so the partition pairing (C_i, C_j) next to the partition P_1 is unused. However, if there is no gap between any two consecutive partitions in $\mathcal{L}'$, the partition pairing (C_i, C_j) next to the last partition in $\mathcal{L}'$ is unused. In any case, the constructed unused partition is

used to pair (C_i, C_j) and is added to the list $\mathcal{L}$ (by scanning the list $\mathcal{L}$ and inserting the partition into a proper position in $\mathcal{L}$). In conclusion, in time $O(n)$ we can find an unused partition to pair two given elements and update the list $\mathcal{L}$. Consequently, the algorithm PREMATCHING runs in time $O(n^2)$.

3. Maximum partition matching: General case. According to Theorem 2.4, we only have to investigate maximum partition matchings for collections of k subsets in U such that $2^k < 2n$. We shall show in this section that a maximum partition matching for such collections can be constructed by a greedy strategy.

Suppose we have constructed a partition matching $M = M[(a_1, b_1), \ldots, (a_h, b_h)]$ and want to expand this matching. The partitions of the collection $\mathcal{S}$ then can be classified into two classes: h of the partitions are used to pair the h pairs (a_i, b_i), $i = 1, \ldots, h$, and the remaining $2^k - h$ partitions are unused. Now if there is an unused partition $P = (\mathcal{A}, \mathcal{B})$ such that there is a left-unpaired element a in $\mathcal{A}$ and a right-unpaired element b in $\mathcal{B}$, then we simply pair the element a with the element b using the partition P, thus expanding the partition matching M.

Now suppose that there is no such unused partition; i.e., for all unused partitions $(\mathcal{A}, \mathcal{B})$, either $\mathcal{A}$ contains no left-unpaired elements or $\mathcal{B}$ contains no right-unpaired elements. This case may not necessarily imply that the current partition matching is the maximum. For example, suppose that $(\mathcal{A}, \mathcal{B})$ is an unused partition such that there is a left-unpaired element a in $\mathcal{A}$ but no right-unpaired elements in $\mathcal{B}$. Assume further that there is a *used* partition $(\mathcal{A}', \mathcal{B}')$ that pairs elements (a', b') such that the element b' is in $\mathcal{B}$ and there is a right-unpaired element b in $\mathcal{B}'$. Then we can let the partition $(\mathcal{A}', \mathcal{B}')$ pair the elements (a', b) and let the partition $(\mathcal{A}, \mathcal{B})$ pair the elements (a, b'), thus expanding the partition matching M. An explanation of this process is that the used partitions have been incorrectly used to pair elements; thus, to construct a maximum partition matching, we must re-pair some of the elements. To further investigate this relation, we need to introduce a few notations.

For a used partition P of $\mathcal{S}$, we underline a set in the left-collection (respectively, the right-collection) of P to indicate that an element in the set is left-paired (respectively, right-paired) by the partition P. The sets will be called the *left-paired set* and the *right-paired set* of the partition P, respectively.

DEFINITION 3.1. *A used partition P is* directly left-reachable *from a partition $P_1 = (\mathcal{A}_1, \mathcal{B}_1)$ if the left-paired set of P is contained in $\mathcal{A}_1$ (the partition P_1 can be either used or unused). The partition P is* directly right-reachable *from a partition $P_2 = (\mathcal{A}_2, \mathcal{B}_2)$ if the right-paired set of P is contained in $\mathcal{B}_2$. A partition P_s is* left-reachable *(respectively,* right-reachable*) from a partition P_1 if there are partitions $P_2, \ldots, P_{s-1}$ such that P_i is directly left-reachable (respectively, directly right-reachable) from P_{i-1} for all $i = 2, \ldots, s$.*

The left-reachability and the right-reachability are transitive relations.

Let $P_1 = (\mathcal{A}_1, \mathcal{B}_1)$ be an unused partition such that there are no left-unpaired elements in $\mathcal{A}_1$, and let $P_s = (\mathcal{A}_s, \mathcal{B}_s)$ be a partition left-reachable from P_1 and there is a left-unpaired element a_s in $\mathcal{A}_s$. We show how we can use a *chain justification* to make a left-unpaired element for the collection $\mathcal{A}_1$.

By Definition 3.1, there are used partitions $P_2, \ldots, P_{s-1}$ such that P_i is directly left-reachable from P_{i-1} for $i = 2, \ldots, s$. We can further assume that P_i is *not* directly left-reachable from P_{i-2} for $i = 3, \ldots, s$—otherwise we simply delete the partition P_{i-1} from the sequence. Thus, these partitions can be written as

$$P_1 = (\{C_1\} \cup \mathcal{A}'_1, \mathcal{B}_1), \qquad P_2 = (\{\underline{C_1}, C_2\} \cup \mathcal{A}'_2, \mathcal{B}_2),$$
$$P_3 = (\{\underline{C_2}, C_3\} \cup \mathcal{A}'_3, \mathcal{B}_3), \qquad \ldots,$$
$$P_{s-1} = (\{\underline{C_{s-2}}, C_{s-1}\} \cup \mathcal{A}'_{s-1}, \mathcal{B}_{s-1}),$$
$$P_s = (\{\underline{C_{s-1}}, C_s\} \cup \mathcal{A}'_s, \mathcal{B}_s),$$

where $\mathcal{A}'_1, \ldots, \mathcal{A}'_s$ are subcollections of $\mathcal{S}$ without an underlined set.

We can assume that the left-unpaired element a_s in $\mathcal{A}_s = \{\underline{C_{s-1}}, C_s\} \cup \mathcal{A}'_s$ is in a nonunderlined set C_s in $\mathcal{A}_s$—otherwise, we consider the sequence $P_1, P_2, \ldots, P_{s-1}$ instead. We modify the partition sequence into

$$P_1 = (\{C_1\} \cup \mathcal{A}'_1, \mathcal{B}_1), \qquad P_2 = (\{C_1, \underline{C_2}\} \cup \mathcal{A}'_2, \mathcal{B}_2),$$
$$P_3 = (\{C_2, \underline{C_3}\} \cup \mathcal{A}'_3, \mathcal{B}_3), \qquad \ldots,$$
$$P_{s-1} = (\{C_{s-2}, \underline{C_{s-1}}\} \cup \mathcal{A}'_{s-1}, \mathcal{B}_{s-1}),$$
$$P_s = (\{C_{s-1}, \underline{C_s}\} \cup \mathcal{A}'_s, \mathcal{B}_s).$$

The interpretation is as follows: We use the partition P_s to left-pair the left-unpaired element a_s (the right-paired element in the right-collection $\mathcal{B}_s$ is unchanged). Thus, the element a_{s-1} in the set C_{s-1} the partition P_s used to left-pair becomes left-unpaired. We then use the partition P_{s-1} to left-pair the element a_{s-1} and leave an element a_{s-2} in the set C_{s-2} left-unpaired. Then we use the partition P_{s-2} to left-pair a_{s-2}, ... At the end, we use the partition P_2 to left-pair an element a_2 in the set C_2 and leave an element a_1 in the set C_1 left-unpaired. Therefore, this process makes an element in the left-collection $\mathcal{A}_1 = \{C_1\} \cup \mathcal{A}'_1$ of the partition P_1 left-unpaired.

The above process will be called a *left-chain justification*. Thus, given an unused partition $P_1 = (\mathcal{A}_1, \mathcal{B}_1)$ in which the left-collection $\mathcal{A}_1$ has no left-unpaired elements and given a used partition $P_s = (\mathcal{A}_s, \mathcal{B}_s)$ left-reachable from P_1 such that the left-collection $\mathcal{A}_s$ of P_s has a left-unpaired element, we can apply the left-chain justification that keeps all used partitions in the partition matching M and makes a left-unpaired element for the partition P_1. A process called *right-chain justification* for right-collections of the partitions can be described similarly. This motivates the algorithm Greedy Expanding in Figure 3.1.

Theorem 3.2. *The algorithm* Greedy Expanding *runs in time* $O(n^2 \log n)$.

Proof. For each set C_i, we keep a counter for the number of left-unpaired elements in C_i and a counter for the number of right-unpaired elements in C_i, so that checking whether a partition has an unpaired element can be done in time $O(k)$.

To test left-reachabilities in each execution of Cases 1–4, we let v_i be the collection of all used partitions whose left-paired set is C_i, for $i = 1, \ldots, k$. Note that if a partition in v_i is directly left-reachable from a partition P, then the left-collection of P contains C_i; thus *all* partitions in v_i are left-reachable from P. If v_i contains a used partition whose left-collection contains a left-unpaired element, then we "label" v_i by such a partition (if there is more than one such partition, pick any one of them). We construct a directed graph G whose vertices are v_1, ..., v_k. There is an edge from vertex v_i to vertex v_j in G if and only if all partitions in v_j are directly left-reachable from a partition in v_i. The graph G contains at most $O(k^2)$ edges and can be constructed in time $O(nk)$ by scanning the partitions in each vertex in G.

For each set C_i in $\mathcal{S}$, in time $O(k^2)$ we can find all vertices v_j in the graph G such that there is a path in G from v_i to v_j. If any v_j of these vertices is labeled by a partition, then we associate the set C_i with the vertex v_j (if there is more than one such vertex, pick any one of them).

Algorithm. GREEDY EXPANDING

INPUT: A collection $\mathcal{S} = \{C_1, \ldots, C_k\}$ of pairwise disjoint subsets of U, with $2^k < 2n$.
OUTPUT: A partition matching M_{exp} in $\mathcal{S}$.

Start with $M_{exp} = \emptyset$ and repeat the following steps until no more changes.

1. **Case 1.** There is an unused partition $P = (\mathcal{A}, \mathcal{B})$ that has a left-unpaired element a in $\mathcal{A}$ and a right-unpaired element b in $\mathcal{B}$
Then pair the two elements (a, b) by the partition P and add P to the matching M_{exp};
2. **Case 2.** There is an unused partition $P = (\mathcal{A}, \mathcal{B})$ that has a right-unpaired element b in $\mathcal{B}$ but no left-unpaired elements in $\mathcal{A}$, and there is a used partition $P' = (\mathcal{A}', \mathcal{B}')$ left-reachable from P and the collection $\mathcal{A}'$ has a left-unpaired element
Then use the left-chain justification to make a left-unpaired element a for the collection $\mathcal{A}$, pair the two elements (a, b) by the partition P, and add P to the matching M_{exp};
3. **Case 3.** There is an unused partition $P = (\mathcal{A}, \mathcal{B})$ that has a left-unpaired element a in $\mathcal{A}$ but no right-unpaired elements in $\mathcal{B}$, and there is a used partition $P' = (\mathcal{A}', \mathcal{B}')$ right-reachable from P and the collection $\mathcal{B}'$ has a right-unpaired element
Then use the right-chain justification to make a right-unpaired element b for the collection $\mathcal{B}$, pair the two elements (a, b) by the partition P, and add P to the matching M_{exp};
4. **Case 4.** There is an unused partition $P = (\mathcal{A}, \mathcal{B})$ that has no left-unpaired elements in $\mathcal{A}$ and no right-unpaired elements in $\mathcal{B}$, and there are two used partitions $P' = (\mathcal{A}', \mathcal{B}')$ left-reachable from P and $P'' = (\mathcal{A}'', \mathcal{B}'')$ right-reachable from P such that $\mathcal{A}'$ has a left-unpaired element and $\mathcal{B}''$ has a right-unpaired element
Then use the left-chain justification to make a left-unpaired element a for $\mathcal{A}$, and use the right-chain justification to make a right-unpaired element b for $\mathcal{B}$, then pair the two elements (a, b) by the partition P, and add P to the matching M_{exp}.

FIG. 3.1. *The algorithm* GREEDY EXPANDING.

The above preprocessing takes time $O(nk + k^3)$.

Let $P = \{\mathcal{A}, \mathcal{B}\}$ be an unused partition. It is easy to verify that for any collection v_j, *all* partitions in v_j are left-reachable from P if and only if there is a set C_i in $\mathcal{A}$ such that there is a path from v_i to v_j in the graph G. In particular, a used partition whose left-collection contains a left-unpaired element is left-reachable from P if and only if a set C_i in $\mathcal{A}$ is associated with a vertex v_j in G labeled by a used partition $P' = \{\mathcal{A}', \mathcal{B}'\}$, where $\mathcal{A}'$ contains a left-unpaired element. Therefore, the conditions in Cases 1–4 in the algorithm GREEDY EXPANDING for each unused partition P can be tested in time $O(k)$. When a set C_i in $\mathcal{A}$ is associated with a vertex v_j labeled by a used partition $P' = \{\mathcal{A}', \mathcal{B}'\}$, the actual left-reachability path from P to P' can be constructed by first finding a path p in the graph G from vertex v_i to vertex v_j and then searching the partitions in each of the vertices on the path p. Thus such a left-reachability path from the unused partition P to the used partition P' can be constructed in time $O(nk + k^2)$.

The right-reachability can be handled similarly.

Since there are at most $2^k < 2n$ partitions, we conclude that each execution of the loop (Cases 1–4) takes time bounded by $O(nk + k^3) = O(n \log n)$ since $k = O(\log n)$. Since each execution of the loop adds at least one partition to the partition matching M_{exp}, the total running time of the algorithm GREEDY EXPANDING is bounded by $O(n^2 \log n)$. □

After execution of the algorithm GREEDY EXPANDING, we obtain a partition matching M_{exp}. For each partition $P = (\mathcal{A}, \mathcal{B})$ not included in M_{exp}, either $\mathcal{A}$ has no

left-unpaired elements and no used partition left-reachable from P has a left-unpaired element in its left-collection or $\mathcal{B}$ has no right-unpaired elements and no used partition right-reachable from P has a right-unpaired element in its right-collection.

DEFINITION 3.3. *Define L_{free} to be the set of partitions P not used by M_{exp} such that the left-collection of P has no left-unpaired elements and no used partition left-reachable from P has a left-unpaired element in its left-collection, and define R_{free} to be the set of partitions P' not used by M_{exp} such that the right-collection of P' has no right-unpaired elements and no used partition right-reachable from P' has a right-unpaired element in its right-collection.*

According to the algorithm GREEDY EXPANDING, each partition not used by M_{exp} is either in the set L_{free} or in the set R_{free}. The sets L_{free} and R_{free} may not be disjoint.

DEFINITION 3.4. *Define L_{reac} to be the set of partitions in M_{exp} that are left-reachable from a partition in L_{free}, and define R_{reac} to be the set of partitions in M_{exp} that are right-reachable from a partition in R_{free}.*

According to Definition 3.3 and Definition 3.4, if a used partition P is in the set L_{reac}, then all elements in its left-collection are left-paired, and if a used partition P is in the set R_{reac}, then all elements in its right-collection are right-paired.

We first show that if L_{reac} and R_{reac} are not disjoint, then we can construct a maximum partition matching from the partition matching M_{exp} constructed by the algorithm GREEDY EXPANDING.

LEMMA 3.5. *If the sets L_{reac} and R_{reac} contain a common partition and the partition matching M_{exp} has less than n pairs, then there is a set C_0 in $\mathcal{S}$, $|C_0| \leq n/2$ such that either all elements in each set $C \neq C_0$ are left-paired and every used partition whose left-paired set is not C_0 is contained in L_{reac} or all elements in each set $C \neq C_0$ are right-paired and every used partition whose right-paired set is not C_0 is contained in R_{reac}.*

Proof. Let the partition $P = (\mathcal{A}, \mathcal{B})$ be in the intersection of L_{reac} and R_{reac}. Therefore, all elements in $\mathcal{A}$ are left-paired and all elements in $\mathcal{B}$ are right-paired. Since there are totally n elements in $\mathcal{A} \cup \mathcal{B}$, one of $\mathcal{A}$ and $\mathcal{B}$ has at least $n/2$ elements.

Suppose that $\mathcal{A}$ has at least $n/2$ elements, which are all left-paired. Since M_{exp} has less than n pairs, there is a set C_0 that contains left-unpaired elements. In particular, C_0 is not contained in the collection $\mathcal{A}$. Thus, $|C_0| \leq n/2$.

Let $P_1, \ldots, P_t$ be the partitions in M_{exp} that are used to left-pair the elements in $\mathcal{A}$. Thus, $t \geq n/2$. Since the left-paired set of each P_i is also contained in $\mathcal{A}$, by Definition 3.1 P_i is directly left-reachable from $P = (\mathcal{A}, \mathcal{B})$. Now $P \in L_{reac}$. Thus, we also have $P_i \in L_{reac}$ for $i = 1, \ldots, t$. In particular, all elements in the left-collection of each P_i have been left-paired. Consequently, the set C_0 is not contained in the left-collection of any of these partitions $P_1, \ldots, P_t$.

Suppose that there is another set $C \neq C_0$ in $\mathcal{S}$ that also is not contained in the left-collection of any of the partitions $P_1, \ldots, P_t$. Then, since the total number of partitions whose left-collections do not contain the sets C_0 and C is 2^{k-2}, we get $2^{k-2} \geq t$. However, this would contradict the facts that $t \geq n/2$ and $2^k < 2n$.

Therefore, the set C_0 is the only set in $\mathcal{S}$ that is not contained in the left-collection of any of the partitions $P_1, \ldots, P_t$. In particular, the set C_0 is the only set that contains left-unpaired elements. All elements in each set $C \neq C_0$ are left-paired.

Now let P' be a used partition whose left-paired set $C \neq C_0$. Since the set C must be contained in the left-collection of some partition P_i among $\{P_1, \ldots, P_t\}$, by the definition of the left-reachability, the partition P' is left-reachable from the

partition P_i. Since the partition P_i is in L_{reac}, we conclude that the partition P' is also in L_{reac}.

Thus, we have proved that if the left-collection $\mathcal{A}$ of the partition $P = (\mathcal{A}, \mathcal{B})$ in the intersection of L_{reac} and R_{reac} has at least $n/2$ elements, then there is a set C_0 in $\mathcal{S}$, $|C_0| \leq n/2$, such that all elements in each set $C \neq C_0$ are left-paired and every used partition whose left-paired set is not C_0 is in the set L_{reac}.

In case the right-collection $\mathcal{B}$ of the partition P has at least $n/2$ elements, we can prove in a completely similar way that there is a set C_0 in $\mathcal{S}$, $|C_0| \leq n/2$, such that all elements in each set $C \neq C_0$ are right-paired and every used partition whose right-paired set is not C_0 is in R_{reac}. □

Now we are ready for the following theorem.

THEOREM 3.6. *If L_{reac} and R_{reac} have a common partition, then the collection $\mathcal{S}$ has a maximum partition matching of n pairs, which can be constructed in linear time from the partition matching M_{exp} constructed by the algorithm* GREEDY EXPANDING.

Proof. If M_{exp} has n pairs, then M_{exp} is already a maximum partition matching. Thus we assume that M_{exp} has less than n pairs. According to Lemma 3.5, we can assume, without loss of generality, that all elements in each set C_i, $i = 2, \ldots, k$, are left-paired and that every used partition whose left-paired set is not C_1 is in L_{reac}. Moreover, $|C_1| \leq \sum_{i=2}^{k} |C_i|$.

Let $t = \sum_{i=2}^{k} |C_i|$ and $d = |C_1|$. Then we can assume that M_{exp} consists of the partitions

$$P_1, \ldots, P_t, P_{t+1}, \ldots, P_{t+h},$$

where $P_1, \ldots, P_t$ are used by M_{exp} to left-pair the elements in $\cup_{i=2}^{k} C_i$, and $P_{t+1}, \ldots, P_{t+h}$ are used by M_{exp} to left-pair the elements in C_1, $h < d$. Moreover, all partitions $P_1, \ldots, P_t$ are in the set L_{reac}. Thus, the set C_1 must be contained in the right-collection in each of the partitions $P_1, \ldots, P_t$.

We ignore the partitions $P_{t+1}, \ldots, P_{t+h}$ and use the partitions $P_1, \ldots, P_t$ to construct a maximum partition matching of n pairs. Note that $\{P_1, \ldots, P_t\}$ also forms a partition matching in the collection $\mathcal{S}$.

Consider the algorithm PARTITION FLIPPING given in Figure 3.2, where *flipping* a partition $(\mathcal{A}, \mathcal{B})$ gives the partition $(\mathcal{B}, \mathcal{A})$.

We must prove that the algorithm PARTITION FLIPPING correctly constructs a partition matching with n pairs.

Step 1 of the algorithm is always possible: Since C_1 is contained in the right-collection of each partition P_i, $i = 1, \ldots, t$, and $t \geq d$ for each right-unpaired element b in C_1, we can always pick a partition P_i that right-pairs an element in $\cup_{i=2}^{k} C_i$ and let P_i right-pair the element b. We keep doing this replacement until all d elements in C_1 get right-paired. At this point, the number of partitions in $\{P_1, \ldots, P_t\}$ that right-pair elements in $\cup_{i=2}^{k} C_i$ is exactly $t - d$.

Step 3 is always possible since the partitions $P_1, \ldots, P_t$ left-pair all elements in $\cup_{i=2}^{k} C_i$.

Now we verify that the sequence $\{P_1, \ldots, P_t, P'_1, \ldots, P'_d\}$ is a partition matching in $\mathcal{S}$.

No two partitions P_i and P_j can be identical since $\{P_1, \ldots, P_t\}$ is supposed to be a partition matching in $\mathcal{S}$. No two partitions P'_i and P'_j can be identical since they are obtained by flipping two different partitions in $\{P_1, \ldots, P_t\}$. No partition P_i is identical to a partition P'_j because P_i has C_1 in its right-collection whereas P'_j has C_1 in its left-collection. Therefore, the partitions $P_1, \ldots, P_t$, $P'_1, \ldots, P'_d$ are all distinct.

Algorithm. PARTITION FLIPPING

INPUT: A partition matching $\{P_1, \ldots, P_t\}$ that left-pairs all elements in $\cup_{i=2}^{k} C_i$, $t = \sum_{i=2}^{k} |C_i|$, and the set C_1 is contained in the right-collection of each partition P_i, $i = 1, \ldots, t$, $d = |C_1| \leq t$.

OUTPUT: A maximum partition matching in $\mathcal{S}$ with n pairs.

1. If not all elements in the set C_1 are right-paired by $P_1, \ldots, P_t$, replace a proper number of right-paired elements in $\cup_{i=2}^{k} C_i$ by the right-unpaired elements in C_1 so that all elements in C_1 are right-paired by $P_1, \ldots, P_t$;
2. Suppose that the partitions $P_1, \ldots, P_{t-d}$ right-pair $t - d$ elements $b_1, \ldots, b_{t-d}$ in $\cup_{i=2}^{k} C_i$, and that $P_{t-d+1}, \ldots, P_t$ right-pair the d elements in C_1;
3. Suppose that $\overline{P}_1, \ldots, \overline{P}_{t-d}$ are the $t - d$ partitions in $\{P_1, \ldots, P_t\}$ that left-pair the elements $b_1, \ldots, b_{t-d}$;
4. Flip each of the d partitions in $\{P_1, \ldots, P_t\} - \{\overline{P}_1, \ldots, \overline{P}_{t-d}\}$ to get d partitions $P'_1, \ldots, P'_d$ to left-pair the d elements in C_1. The right-paired element of each P'_i is the left-paired element before the flipping;
5. $\{P_1, \ldots, P_t, P'_1, \ldots, P'_d\}$ is a partition matching in $\mathcal{S}$ with n pairs.

FIG. 3.2. *The algorithm* PARTITION FLIPPING.

Each of the partitions $P_1, \ldots, P_t$ left-pairs an element in $\cup_{i=2}^{k} C_i$, and each of the partitions $P'_1, \ldots, P'_d$ left-pairs an element in C_1. Thus, all elements in the universal set U get left-paired in $\{P_1, \ldots, P_t, P'_1, \ldots, P'_d\}$.

Finally, the partitions $P_1, \ldots, P_t$ right-pair all elements in C_1 and the elements $b_1, \ldots, b_{t-d}$ in $\cup_{i=2}^{k} C_i$. Now, by our selection of the partitions, the partitions $P'_1, \ldots, P'_d$ precisely right-pair all the elements in $\cup_{i=2}^{k} C_i - \{b_1, \ldots, b_{t-d}\}$. Thus, all elements in U also get right-paired in $\{P_1, \ldots, P_t, P'_1, \ldots, P'_d\}$.

This leads to the conclusion that the sequence $\{P_1, \ldots, P_t, P'_1, \ldots, P'_d\}$ is a maximum partition matching in the collection $\mathcal{S}$.

The running time of the algorithm PARTITION FLIPPING is obviously linear. □

Now we consider the case when L_{reac} and R_{reac} have no common partitions.

THEOREM 3.7. *If L_{reac} and R_{reac} have no common partitions, then the partition matching M_{exp} constructed by the algorithm* GREEDY EXPANDING *is a maximum partition matching.*

Proof. Let W_{other} be the set of used partitions in M_{exp} that belong to neither L_{reac} nor R_{reac}. Then

$$L_{free} \cup R_{free} \cup L_{reac} \cup R_{reac} \cup W_{other}$$

is the set of all partitions of the collection $\mathcal{S}$, and

$$L_{reac} \cup R_{reac} \cup W_{other}$$

is the set of partitions contained in the partition matching M_{exp}. Since all sets L_{reac}, R_{reac}, and W_{other} are pairwise disjoint, the number of partitions in M_{exp} is precisely

$$|L_{reac}| + |R_{reac}| + |W_{other}|.$$

Now consider the set $W_L = L_{free} \cup L_{reac}$. Let U_L be the set of elements that appear in the left-collection of a partition in W_L. We have the following situations:

1. Every $P \in L_{reac}$ left-pairs an element in U_L.
2. Every element in U_L is left-paired.

Algorithm. GENERAL MAXIMUM PARTITION MATCHING

INPUT: A collection $\mathcal{S} = \{C_1, \ldots, C_k\}$ of pairwise disjoint subsets of $U = \{1, 2, \ldots, n\}$.
OUTPUT: A maximum partition matching M in $\mathcal{S}$.

1. **Case 1.** $2^k \geq 2n$
   call the algorithm PREMATCHING to construct a maximum partition matching;
2. **Case 2.** $2^k < 2n$
   call the algorithm GREEDY EXPANDING to construct a partition matching M_{exp}; compute L_{reac} and R_{reac};
   if L_{reac} and R_{reac} have a common partition
   then call the algorithm PARTITION FLIPPING to construct a maximum partition matching
   else M_{exp} is a maximum partition matching.

FIG. 3.3. *The algorithm* GENERAL MAXIMUM PARTITION MATCHING.

3. If an element a in U_L is left-paired by a partition P, then $P \in L_{reac}$. (*Proof.* Let the element a be in the set C. Then the set C is the left-paired set of the partition P. Since the element a is in U_L, the set C is also contained in the left-collection of a partition P' that is in either L_{free} or L_{reac}. The partition P is left-reachable from P'; thus, it must be in L_{reac}.)

Therefore, the partitions in L_{reac} precisely left-pair the elements in U_L. This gives $|L_{reac}| = |U_L|$. Since there are only $|U_L|$ elements that appear in the left-collections in partitions in $L_{free} \cup L_{reac}$, we conclude that the partitions in $W_L = L_{free} \cup L_{reac}$ can be used to left-pair at most $|U_L| = |L_{reac}|$ elements in *any partition matching* in the collection $\mathcal{S}$.

Similarly, the partitions in the set $W_R = R_{free} \cup R_{reac}$ can be used to right-pair at most $|R_{reac}|$ elements in *any partition matching* in the collection $\mathcal{S}$.

Therefore, any partition matching in the collection $\mathcal{S}$ can include at most $|L_{reac}|$ partitions in the set W_L, at most $|R_{reac}|$ partitions in the set W_R, and at most all partitions in the set W_{other}. Consequently, a maximum partition matching in $\mathcal{S}$ consists of at most

$$|L_{reac}| + |R_{reac}| + |W_{other}|$$

partitions. Since the partition matching M_{exp} constructed by the algorithm GREEDY EXPANDING contains just this many partitions, M_{exp} is a maximum partition matching in the collection $\mathcal{S}$. □

We summarize all the discussions above into the following theorem.

THEOREM 3.8. *The maximum partition matching problem can be solved in time* $O(n^2 \log n)$.

Proof. The problem is solved by the algorithm GENERAL MAXIMUM PARTITION MATCHING given in Figure 3.3. The correctness of the algorithm has been proved by Theorems 2.4, 3.6, and 3.7. To construct the sets L_{reac} and R_{reac}, we use an algorithm similar to the one described for the algorithm GREEDY EXPANDING. That is, we first compute all the used partitions reachable from each set C_i and then use this information to examine each unused partition. This can be done in time $O(n^2 \log n)$, and we leave the detailed verification to interested readers. □

4. Parallel routing in star networks. In this section, we show an application of maximum partition matching in parallel routing in star networks.

The star network [1] has received considerable attention from researchers recently as a graph model for interconnection networks. It has been shown to be an attractive

alternative to the widely used hypercube model. Like the hypercube, the star network has rich structural and symmetric properties as well as fault tolerant characteristics. Moreover, it has a smaller diameter and degree while being comparable to a hypercube with the same number of vertices.

Formally, the *n-dimensional star network* (or simply *the n-star network*) is an undirected graph consisting of $n!$ nodes labeled with the $n!$ permutations on symbols $1, 2, \ldots, n$. There is an edge between a node u to a node v if and only if there is a transposition $\pi[1, i]$, $2 \leq i \leq n$, such that $\pi[1, i](u) = v$. The n-star network is an $(n-1)$-connected and vertex symmetric Cayley graph [1].

Parallel routing, i.e., finding parallel node-disjoint paths between two nodes in a star network, has been investigated recently [6, 11, 13, 14]. Since the n-star network is vertex symmetric, a parallel routing between two arbitrary nodes can be mapped to a parallel routing between a node and the identity node ε. Let $\text{dist}(u)$ define the distance from the node u to ε. Day and Tripathi [6] and Jwo, Lakshmivarahan, and Dhall [11] have shown that for any node u there are $n-1$ node-disjoint paths connecting u and ε such that no path has length larger than $\text{dist}(u) + 4$. An algorithm was described in [11] to construct the maximum number of node-disjoint paths of length $\text{dist}(u)$ between the nodes u and ε. Unfortunately, the algorithm runs in exponential time in the worst case. Moreover, the algorithm seems to contain a serious bug. For example, the algorithm always constructs an even number of shortest paths from a node u to ε, whereas a star network may have an odd number of node-disjoint shortest paths between a node u and ε (see our discussion in the next section).

Let $u = a_1 a_2 \cdots a_n$ be a node in the n-star network (i.e., u is a permutation on $1, \ldots, n$). Suppose that the cycle structure of u is $u = c_1 \cdots c_k e_1 \cdots e_m$, where c_i are cycles of length at least 2 and e_j are cycles of length 1. If we further let $l = \sum_{i=1}^{k} |c_i|$, then the shortest distance from u to the identity node ε is given by the formula [1]

$$\text{dist}(u) = \begin{cases} l + k, & \text{if } a_1 = 1, \\ l + k - 2, & \text{if } a_1 \neq 1. \end{cases}$$

From this formula, we can derive an upper bound on the number of node-disjoint shortest paths from u to ε. We distinguish two cases. Recall that $\pi[1, i]$ is the transposition on permutations that exchanges the positions of the first symbol and the ith symbol and that $\pi[a]$ is the transposition on permutations that exchanges the positions of the first symbol and the symbol a.

Suppose $a_1 = 1$ in the node $u = a_1 a_2 \cdots a_n$ with cycle structure $c_1 \cdots c_k e_1 \cdots e_m$. If $a \neq 1$ is in a single symbol cycle e_i, then by the discussion in section 1 and the above formula, it is not hard to show that $\text{dist}(\pi[a](u)) = \text{dist}(u) + 1$; i.e., the edge from u to $\pi[a](u)$ does not lead to a shortest path from u to ε. Thus, in this case the total number of node-disjoint shortest paths from u to ε is bounded by $l = \sum_{i=1}^{k} |c_i|$. It is also easy to see that if a is in a cycle c_i of length of at least 2, then $\text{dist}(\pi[a](u)) = \text{dist}(u) - 1$.

Suppose $a_1 \neq 1$. We further assume that in the cycle structure $c_1 \cdots c_k e_1 \cdots e_m$ of u, we have $c_1 = (a_1^{(1)} a_2^{(1)} \cdots a_d^{(1)})$, where $a_1 = a_1^{(1)}$ and $a_d^{(1)} = 1$. Let a be an element in the cycle c_1 such that $a \neq a_1$ and $a \neq a_2^{(1)}$. Then by the discussion in section 1 and the above formula, $\text{dist}(\pi[a](u)) = \text{dist}(u) + 1$; i.e., the edge from u to $\pi[a](u)$ does not lead to a shortest path from u to ε. Similarly, if a is in a single symbol cycle e_j, then the edge from u to $\pi[a](u)$ does not lead to a shortest path from u to ε. Thus, in this case the total number of node-disjoint shortest paths from u to ε is bounded

by $1 + \sum_{i=2}^{k} |c_i|$. It is also easy to see that if $a = a_2^{(1)}$ or if a is in a cycle c_i of length at least 2, $i > 1$, then $\text{dist}(\pi[a](u)) = \text{dist}(u) - 1$.

We summarize the above discussion in the following lemma.

LEMMA 4.1. *Let $u = a_1 a_2 \cdots a_n$ be a node in the n-star network with a cycle structure $u = c_1 \cdots c_k e_1 \cdots e_m$, where c_i are cycles of length at least 2 and e_j are cycles of length 1.*

If $a_1 = 1$, then the number of node-disjoint paths of length dist(u) from u to ε is bounded by $\sum_{i=1}^{k} |c_i|$.

If $a_1 \neq 1$, then the number of node-disjoint paths of length dist(u) from u to ε is bounded by $1 + \sum_{i=2}^{k} |c_i|$ (we assume that the symbol 1 is contained in cycle c_1).

The above discussion also tells us that on a shortest path from a node v to ε, from each node $u = a_1 a_2 \cdots a_n$ on the path, with a cycle structure $u = c_1 \cdots c_k e_1 \cdots e_m$, to the next node on the path, we must perform one of the following two operations:

Rule 1. If $a_1 = 1$, then merge the single symbol cycle $\{1\}$ into a cycle c_i of length at least 2. This corresponds to applying a transposition $\pi[a]$ with $a \in c_i$.

Rule 2. In the case $a_1 \neq 1$, assume that $c_1 = (a_1^{(1)} a_2^{(1)} \cdots a_d^{(1)})$, where $a_1 = a_1^{(1)}$ and $a_d^{(1)} = 1$. Then either merge the cycle c_1 with a cycle c_i, $i > 1$ (this corresponds to applying the transposition $\pi[a]$, where $a \in c_i$), or delete the symbol a_1 from cycle c_1 (this corresponds to applying the transposition $\pi[a_2^{(1)}]$ and putting a_1 in a single symbol cycle).

Remark. Thus, a transposition $\pi[a]$ is never applied along the shortest path if $a \neq 1$ is in a single symbol cycle. Consequently, once a symbol $a \neq 1$ is in a single symbol cycle, it will stay in the single symbol cycle forever along the shortest path.

Now we are ready to discuss parallel routing on the n-star network. Again suppose that $u = a_1 a_2 \cdots a_n$ is a node of the n-star network and we want to construct a maximum number of node-disjoint shortest paths from the node u to the identity node ε in the n-star network.

If $a_1 = 1$, then by the above analysis there are at most $l = \sum_{i=1}^{k} |c_i|$ node-disjoint shortest paths. In fact, it is not very hard to construct l node-disjoint shortest paths from such a node u to ε [6, 11]. We will not discuss this case here. Interested readers are referred to [6, 11].

We will concentrate on the other case, which is more difficult.

Problem A (parallel routing (hard case)). Given a node $u = a_1 a_2 \cdots a_n$ in the n-star network, where $a_1 \neq 1$, with a cycle structure $c_1 \cdots c_k e_1 \cdots e_m$, where $c_1 = (a_1^{(1)} \cdots a_d^{(1)})$, $a_1^{(1)} = a_1$, and $a_d^{(1)} = 1$, construct a maximum number of node-disjoint shortest paths (of length dist(u)) in the n-star network from the node u to the identity node ε.

We first derive another upper bound for the number of node-disjoint shortest paths from the node u to ε in terms of the maximum partition matching of the cycles $c_2, \ldots, c_k$, regarding $c_2, \ldots, c_k$ as sets of symbols.

LEMMA 4.2. *Let u be the node described in Problem* A. *Then the number of node-disjoint shortest paths from u to ε cannot be larger than 2 plus the number of partitions of a maximum partition matching in $\mathcal{S} = \{c_2, \ldots, c_k\}$.*

Proof. Let $P_1, \ldots, P_s$ be s node-disjoint shortest paths from u to ε. For each path P_i, let u_i be the first node on P_i such that the symbol 1 is in the first position in the permutation u_i. The node u_i is obtained by repeatedly applying Rule 2, starting from the node u. It is easy to prove by induction that for any node v from u to u_i on the path P_i, the only possible cycle of length larger than 1 in v that is not in $\{c_2, \ldots, c_k\}$

is the one that contains the symbol 1. In particular, the node u_i must have a cycle structure of the form

$$u_i = c'_1 \cdots c'_{k_i} e'_1 \cdots e'_{m_i},$$

where c'_i are cycles of length at least 2, e'_j are cycles of length 1, and $\mathcal{B}_i = \{c'_1, \ldots, c'_{k_i}\}$ is a subcollection of $\mathcal{S} = \{c_2, \ldots, c_k\}$.

Assume that the first edge on P_i is from u to $\pi[b_i](u)$. By the discussion of Lemma 4.1, b_i is either $a_2^{(1)}$ or one of the symbols in $\cup_{i=2}^{k} c_i$. Moreover, by Rule 2, once b_i is contained in the cycle containing the symbol 1, it will stay in the cycle containing the symbol 1 until it is put into a single symbol cycle. In particular, the symbol b_i is not in the set $\cup_{j=1}^{k_i} c'_j$.

Now consider the last edge on the path P_i, which must be from a node w_i with cycle structure $(d_i 1)$ to the identity node ε. Since the symbol d_i is in a cycle of length larger than 1 in w_i, by the remark, d_i is also in a cycle of length larger than 1 in the node u_i; that is, $d_i \in \cup_{j=1}^{k_i} c'_j$. The only exception is $d_i = a_{d-1}^{(1)}$ (in this case $u_i = \varepsilon$).

Now we let $\mathcal{A}_i = \mathcal{S} - \mathcal{B}_i$. Then we can conclude that except at most two paths P_1 and P_2, each P_i, $i \geq 3$, of the other paths $P_3, \ldots, P_s$ must start with an edge $\{u, \pi[b_i](u)\}$, where b_i is in $\mathcal{A}_i$, and end with an edge $\{w_i, \varepsilon\}$, where w_i has a cycle structure $(d_i 1)$ and $d_i \in \mathcal{B}_i$ (the two exceptional paths P_1 and P_2 may start with the edge $\{u, \pi[a_2^{(1)}](u)\}$ or end with the edge $\{w, \varepsilon\}$, where w has a cycle structure $(a_{d-1}^{(1)} 1)$).

Now since the s paths $P_1, \ldots, P_s$ are node-disjoint, the symbols $b_3, \ldots, b_s$ are all pairwise distinct, and the symbols $d_3, \ldots, d_s$ are also pairwise distinct. Moreover, since all nodes $u_3, \ldots, u_s$ are pairwise distinct, the collections $\mathcal{B}_3, \ldots, \mathcal{B}_s$ of cycles are also pairwise distinct. Consequently, the partitions $(\mathcal{A}_3, \mathcal{B}_3)$, …, $(\mathcal{A}_s, \mathcal{B}_s)$ form a partition matching of the collection $\mathcal{S} = \{c_2, \ldots, c_k\}$.

This leads us to conclude that s cannot be larger than 2 plus the number of partitions in a maximum partition matching of the collection $\mathcal{S} = \{c_2, \ldots, c_k\}$, thus proving the lemma. □

Now we show how we construct a maximum number of node-disjoint shortest paths from the node u to the identity node ε in the n-star network. We first show how to route a single shortest path from u to ε, given a partition $(\mathcal{A}, \mathcal{B})$ of the collection $\mathcal{S} = \{c_2, \ldots, c_k\}$ and a pair of symbols b and d, where b is in $\mathcal{A}$ and d is in $\mathcal{B}$. We also allow b to be $a_2^{(1)}$—in this case, d must be in $\cup_{i=2}^{k} c_i$, and $\mathcal{A} = \phi$ and $\mathcal{B} = \mathcal{S}$. Similarly, we allow d to be $a_{d-1}^{(1)}$—in this case, b must be in $\cup_{i=2}^{k} c_i$, and $\mathcal{B} = \phi$ and $\mathcal{A} = \mathcal{S}$. Consider the algorithm Single Routing given in Figure 4.1. Since the algorithm Single Routing starts with the node u and applies only transpositions described in Rules 1 and 2, we conclude that the algorithm Single Routing constructs a shortest path from the node u to the node ε.

Now we are ready to describe the final algorithm. Consider the algorithm Optimal Parallel Routing given in Figure 4.2.

Theorem 4.3. *Algorithm* Optimal Parallel Routing *constructs a maximum number of node-disjoint shortest paths from the node u to the identity node ε in time $O(n^2 \log n)$.*

Proof. From Lemmas 4.1 and 4.2, we know that the number of shortest paths constructed by the algorithm Optimal Parallel Routing matches the maximum number of node-disjoint shortest paths from u to ε. What remains is to show that all these paths are node-disjoint.

Algorithm. SINGLE ROUTING

INPUT: A partition $(\mathcal{A}, \mathcal{B})$ of $\mathcal{S} = \{c_2, \ldots, c_k\}$, and two symbols b in $\mathcal{A}$ and d in $\mathcal{B}$. b can be $a_2^{(1)}$ with $\mathcal{A} = \phi$, and d can be $a_{d-1}^{(1)}$ with $\mathcal{B} = \phi$.

OUTPUT: A shortest path from u to ε starting with the edge $\{u, \pi[b](u)\}$ and ending with the edge $\{w, \varepsilon\}$, where w has a cycle structure $(d1)$.

1. **if** $b \neq a_2^{(1)}$
   apply $\pi[b]$ to u to merge the cycle in $\mathcal{A}$ that contains b into the cycle c_1; then merge in an arbitrary order the rest of the cycles in $\mathcal{A}$ into the cycle containing the symbol 1;
2. repeatedly delete symbols in the cycle containing the symbol 1 until the symbol 1 is put into a single symbol cycle;
3. **if** $d \neq a_{d-1}^{(1)}$
   suppose that the cycle C containing d in $\mathcal{B}$ is $(d' \cdots d)$. Apply $\pi[d']$ to merge $(d' \cdots d)$ with the single symbol cycle $\{1\}$; then merge in an arbitrary order the rest of the cycles in $\mathcal{B}$ into the cycle containing the symbol 1;
4. repeatedly delete symbols in the cycle containing the symbol 1 until you reach the node ε.

FIG. 4.1. *The algorithm* SINGLE ROUTING.

Algorithm. OPTIMAL PARALLEL ROUTING

INPUT: The node u in the n-star network, as described in Problem A.

OUTPUT: A maximum number of node-disjoint shortest paths from u to ε.

1. Construct a maximum partition matching $M[(b_1, d_1), \ldots, (b_s, d_s)]$ in the collection $\mathcal{S} = \{c_2, \ldots, c_k\}$ with the partitions $(\mathcal{A}_1, \mathcal{B}_1)$, …, $(\mathcal{A}_s, \mathcal{B}_s)$ of $\mathcal{S}$;
2. **if** $s = \sum_{i=2}^{k} |c_i|$
   then construct $s + 1$ node-disjoint shortest paths as follows.
   2.1. Call the algorithm SINGLE ROUTING with the partition $(\phi, \mathcal{S})$ of $\mathcal{S}$ and the symbol pair $(a_2^{(1)}, d_1)$;
   2.2. Call the algorithm SINGLE ROUTING with the partition $(\mathcal{S}, \phi)$ of $\mathcal{S}$ and the symbol pair $(b_1, a_{d-1}^{(1)})$;
   2.3. For $i = 2$ to s, call the algorithm SINGLE ROUTING with the partition $(\mathcal{A}_i, \mathcal{B}_i)$ of $\mathcal{S}$ and the symbol pair (b_i, d_i);
3. **if** $s < \sum_{i=2}^{k} |c_i|$
   then construct $s + 2$ node-disjoint shortest paths as follows.
   3.1. Let b_0 be a left-unpaired symbol and d_0 be a right-unpaired symbol in the partition matching constructed in step 1.
   3.2. Call the algorithm SINGLE ROUTING with the partition $(\phi, \mathcal{S})$ of $\mathcal{S}$ and the symbol pair $(a_2^{(1)}, d_0)$;
   3.3. Call the algorithm SINGLE ROUTING with the partition $(\mathcal{S}, \phi)$ of $\mathcal{S}$ and the symbol pair $(b_0, a_{d-1}^{(1)})$;
   3.4. For $i = 1$ to s, call the algorithm SINGLE ROUTING with the partition $(\mathcal{A}_i, \mathcal{B}_i)$ of $\mathcal{S}$ and the symbol pair (b_i, d_i).

FIG. 4.2. *The algorithm* OPTIMAL PARALLEL ROUTING.

Suppose that the algorithm OPTIMAL PARALLEL ROUTING constructs h shortest paths P_1, P_2, …, P_h from node u to node ε, where $h = s + 1$ or $h = s + 2$ depending on whether $s = \sum_{i=2}^{k} |c_i|$, and suppose that the path P_i is constructed by calling the algorithm SINGLE ROUTING on partition $(\mathcal{A}_i, \mathcal{B}_i)$ of $\mathcal{S}$ and symbol pair (b_i, d_i) for all $i = 1, \ldots, h$. Note that if $\mathcal{A}_i = \phi$, then we have $b_i = a_2^{(1)}$, and if $\mathcal{B}_i = \phi$, then we have $d_i = a_{d-1}^{(1)}$. Now fix an i and consider the path P_i, which is constructed from the partition $(\mathcal{A}_i, \mathcal{B}_i)$ and the symbol pair (b_i, d_i). Let $\mathcal{A}_i = \{c_2^{(i)}, \ldots, c_l^{(i)}\}$ and $\mathcal{B}_i = \{c_{l+1}^{(i)}, \ldots, c_k^{(i)}\}$, where if $\mathcal{A}_i \neq \phi$, then the cycle $c_2^{(i)}$ is of form $c_2^{(i)} = (b_i * * b_i')$,

and if $\mathcal{B}_i \neq \phi$, then the cycle $c_{l+1}^{(i)}$ is of form $c_{l+1}^{(i)} = (* *\ d_i)$, where we use "$* *$" to represent the irrelevant part of a cycle structure. Finally, recall that the cycle c_1 has the form $c_1 = (a_1^{(1)} * * a_{d-1}^{(1)} 1)$.

The interior nodes of the path P_i can be split into three segments: $I_1^{(i)}$, $I_2^{(i)}$, and $I_3^{(i)}$. The first segment $I_1^{(i)}$ corresponds to nodes constructed in step 1 of the algorithm SINGLE ROUTING that first merges cycle $c_2^{(i)}$ into cycle c_1, obtaining a cycle of the form $(b_i * * b_i' a_1^{(1)} * * a_{d-1}^{(1)} 1)$, and then merges cycles $c_3^{(i)}, \ldots, c_l^{(i)}$ into the cycle containing symbol 1. Therefore, all nodes in this segment are of the form

$$(* *\ b_i * * b_i' a_1^{(1)} * * a_{d-1}^{(1)} 1) * *.$$

The second segment $I_2^{(i)}$ corresponds to the nodes constructed by step 2 of the algorithm SINGLE ROUTING that deletes symbols in the cycle containing the symbol 1. All nodes in this segment are of the form

$$(* *\ 1) c_{l+1}^{(i)} \cdots c_k^{(i)} * *.$$

The third segment $I_3^{(i)}$ corresponds to the nodes constructed by steps 3 and 4 of the algorithm SINGLE ROUTING, which first merges the cycle $c_{l+1}^{(i)}$ into the single symbol cycle $\{1\}$, obtaining a cycle of form $(* *\ d_i 1)$, then merges the cycles $c_{l+2}^{(i)}, \ldots, c_k^{(i)}$, and then deletes symbols in the cycle containing the symbol 1. Therefore, all nodes in this segment should have the form

$$(* *\ d_i 1) * *.$$

In case $\mathcal{A}_i = \phi$, we have $b_i = a_2^{(1)}$ and the segment $I_1^{(i)}$ is empty, and in case $\mathcal{B}_i = \phi$, we have $d_i = a_{d-1}^{(1)}$ and the segment $I_3^{(i)}$ is empty.

We now show that any two shortest paths P_i and P_j, $i \neq j$, constructed by the algorithm OPTIMAL PARALLEL ROUTING are node-disjoint. Let v be a node on the path P_i.

Suppose that $v = (* *\ b_i * * b_i' a_1^{(1)} * * a_{d-1}^{(1)} 1) * *$ is a node on the first segment $I_1^{(i)}$ of the path P_i. The node cannot be on the first segment $I_1^{(j)}$ of the path P_j since all nodes on $I_1^{(j)}$ are of form $(* *\ b_j * * b_j' a_1^{(1)} * * a_{d-1}^{(1)} 1) * *$ and $b_i \neq b_j$ (thus $b_i' \neq b_j'$). Moreover, the node v cannot be on the second or the third segment of P_j since the cycle structure of a node on the second or the third segment of P_j has more single symbol cycles (note that each execution of step 2 of the algorithm SINGLE ROUTING creates a new single symbol cycle in the cycle structure).

If $v = (* *\ 1) c_{l+1}^{(i)} \cdots c_k^{(i)} * *$ is on the second segment $I_2^{(i)}$ of the path P_i, then v cannot be on the second segment $I_2^{(j)}$ of P_j since each node on $I_2^{(j)}$ is of form $(* *\ 1) c_{f+1}^{(j)} \cdots c_k^{(j)} * *$ and

$$\mathcal{B}_i = \{c_{l+1}^{(i)}, \ldots, c_k^{(i)}\} \neq \mathcal{B}_j = \{c_{f+1}^{(j)}, \ldots, c_k^{(j)}\}.$$

The node v cannot be on the third segment $I_3^{(j)}$ of P_j either since each node on the segment $I_3^{(j)}$ is of form $(* *\ d_j 1) * *$, where $d_j \in \cup_{i=2}^{k} c_i$, while the cycle containing the symbol 1 in the node v is either a single symbol cycle or of form $(* *\ a_{d-1}^{(1)} 1)$, where $a_{d-1}^{(1)}$ is in c_1.

Finally, if $v = (** \ d_i 1) **$ is on the third segment of the path P_i, then v cannot be on the third segment of P_j because $d_i \neq d_j$.

By symmetry, the above analysis shows that the two shortest paths P_i and P_j constructed by the algorithm Optimal Parallel Routing must be node-disjoint.

The running time of the algorithm Optimal Parallel Routing is dominated by step 1 of the algorithm, which takes time $O(n^2 \log n)$ according to Theorem 3.8. Thus, the algorithm Optimal Parallel Routing runs in time $O(n^2 \log n)$. □

5. Conclusion and remarks. We have presented an efficient algorithm for the maximum partition matching problem. By a nontrivial reduction, we have shown that finding the maximum number of node-disjoint shortest paths between two given nodes in the star networks can be reduced to the maximum partition matching problem. This gives the first correct and efficient algorithm for constructing the maximum number of node-disjoint shortest paths between two given nodes in the star networks.

The problem of constructing the maximum number of node-disjoint shortest paths between two given nodes in the star networks was previously investigated in [11], which presents an algorithm (Algorithm 3.2 in [11]) that claims to find the maximum number of node-disjoint shortest paths between two given nodes in the star networks. For each node $u = c_1 \cdots c_k e_1 \cdots e_m$, the algorithm in [11] runs in time exponential in k, thus in time exponential in n in the worst case (when $k = \Theta(n)$). More seriously, the algorithm seems based on an incorrect observation that claims that when $k > 1$, the maximum number of node-disjoint shortest paths from u to ε is always even (see the paragraph following Lemma 3.11 in [11]). Therefore, in case $k > 1$, the algorithm in [11] always produces an even number of node-disjoint shortest paths from the node u to ε. The incorrectness of this can be shown as follows. Consider a node $u = c_1 \cdots c_k e_1 \cdots e_m$ in the n-star network, where $k > 1$. It is easy to make the node u satisfy the following conditions: (1) $|c_2| \geq |c_i|$ for $i = 3, \ldots, k$ and $|c_2| \leq \sum_{i=3}^{k} |c_i|$; (2) the number $s = \sum_{i=2}^{k} |c_i|$ is even; and (3) $k - 1$ is at least as large as $\log(2s)$. Now, according to the discussion in section 2, we can construct a partition matching of order s for the collection $\{c_2, \ldots, c_k\}$. Moreover, by step 2 of the algorithm Optimal Parallel Routing, the maximum number of node-disjoint shortest paths from u to ε is $s + 1$, which is an odd number. A concrete example of this construction in the n-star network can be found in [3].

Finally, we describe how the algorithm Optimal Parallel Routing can be used to construct, between two nodes in the n-star network, $n - 1$ node-disjoint paths of minimum bulk length. Let G be an h-connected graph. By Menger's theorem [12], for any pair of nodes u and v in G there are h node-disjoint paths connecting u and v. The *bulk length* of h node-disjoint paths connecting u and v in G is defined to be the length of the longest path among the h paths. The *bulk distance* between the two nodes u and v is defined to be the minimum bulk length over all groups of h node-disjoint paths connecting u and v. Clearly, the bulk distance between two nodes u and v is at least as large as the distance between u and v, which is defined to be the length of the shortest path connecting u and v. In general, the problem of computing the bulk distance between two given nodes in a graph is NP-hard [8, 9].

The bulk distance problem on the star networks has been studied recently [4, 5, 6, 11, 13, 14]. Since the n-star network is $(n - 1)$-connected and vertex symmetric [1], the bulk distance problem on two arbitrary nodes in the n-star network can be converted to the problem of finding $n - 1$ node-disjoint paths of minimum bulk length from the node u to the identity node ε. Let $\mathrm{dist}(u)$ and $B\mathrm{dist}(u)$ be the distance and bulk distance, respectively, between the node u and the identity node ε in the

n-star network. It has been shown that $B\text{dist}(u)$ is equal to $\text{dist}(u)$ plus an even number [6]. Day and Tripathi [6] have developed an algorithm that constructs $n-1$ node-disjoint paths between u and ε with bulk length $\text{dist}(u)+4$. Thus, we always have $B\text{dist}(u) \leq \text{dist}(u)+4$. The authors of the present paper [4, 5] have established a sufficient and necessary condition for the node u to have bulk distance $\text{dist}(u)+4$ and have developed an $O(n^2 \log n)$ time algorithm to construct $n-1$ node-disjoint paths between u and ε of bulk length $\text{dist}(u)+2$ when the bulk distance of u is less than $\text{dist}(u)+4$.

Combining these results and the results in the present paper, we obtain an $O(n^2 \log n)$ time algorithm that constructs $n-1$ node-disjoint paths of bulk length $B\text{dist}(u)$ between any node u and the identity node ε in the n-star network, as follows. We first check whether the bulk distance $B\text{dist}(u)$ is $\text{dist}(u)+4$, using the formula given in [4, 5]. If $B\text{dist}(u) = \text{dist}(u)+4$, then we apply the algorithm given in [6] to construct $n-1$ node-disjoint paths of bulk length $\text{dist}(u)+4 = B\text{dist}(u)$ from u to ε. If the bulk distance of u is less than $\text{dist}(u)+4$, then we apply the algorithm Optimal Parallel Routing in the present paper to find the maximum number of node-disjoint shortest paths. If the algorithm returns $n-1$ such shortest paths, then these paths are the $n-1$ node-disjoint paths of bulk distance $\text{dist}(u)$ between u and ε. If the algorithm returns less than $n-1$ such shortest paths, then we know that the bulk distance of u is $\text{dist}(u)+2$ so the algorithm developed in [4] can be applied to construct $n-1$ node-disjoint paths of bulk length $\text{dist}(u)+2$. This completes the description of the algorithm that always constructs $n-1$ node-disjoint paths of bulk length $B\text{dist}(u)$ between a node u and the identity node ε in the n-star network. The running time of the algorithm is clearly $O(n^2 \log n)$.

We would like to make a few remarks on the complexity of the above algorithm. The bulk distance problem on general graphs is NP-hard [9]. Thus, it is very unlikely that the bulk distance problem can be solved in time polynomial in the size of the input graph. On the other hand, our algorithm solves the bulk distance problem in time $O(n^2 \log n)$ on the n-star network. Note that the n-star network has $n!$ nodes. Therefore, the running time of our algorithm is actually a polynomial of the logarithm of the size of the input star network. Moreover, our algorithm is almost optimal (differs at most by a $\log n$ factor) since the following lower bound can be easily observed—the distance $\text{dist}(u)$ from u to ε can be as large as $\Theta(n)$. Thus, constructing $n-1$ node-disjoint paths from u to ε takes time at least $\Theta(n^2)$ in the worst case.

Acknowledgments. The authors would like to thank Professor Eva Tardos for her thorough and valuable comments and useful discussion on an earlier version of this paper. The authors are grateful to Professor Jonathan Gross for informing them of the recent discovery by Galil and Yu on graph bulk distance [8]. The authors also thank Professor Laxmi Bhuyan, Professor Don Friesen, Professor Mi Lu, and Dr. Xiangdong Yu for their comments and discussion. Finally, the authors express their sincere thanks to the referees, whose critical comments and suggestions were important for removing possible bugs and confusion in a previous version and have greatly improved the presentation of the paper.

REFERENCES

[1] S. B. Akers and B. Krishnamurthy, *A group-theoretic model for symmetric interconnection networks*, IEEE Trans. Comput., 38 (1989), pp. 555–565.

[2] G. Birkhoff and S. MacLane, *A Survey of Modern Algebra*, Macmillan, New York, 1965.

[3] C. C. Chen, *Combinatorial and Algebraic Methods in Star and de Bruijn Networks*, Ph.D. thesis, Dept. of Computer Science, Texas A&M University, College Station, TX, 1995.
[4] C. C. Chen and J. Chen, *Vertex-disjoint routings in star graphs*, in Proc. IEEE 1st International Conference on Algorithms and Architectures for Parallel Processing, Brisbane, Australia, 1995, pp. 460–464.
[5] C. C. Chen and J. Chen, *Optimal parallel routing in star networks*, IEEE Trans. Comput., 46 (1997), pp. 1293–1303.
[6] K. Day and A. Tripathi, *A comparative study of topological properties of hypercubes and star graphs*, IEEE Trans. Parallel Distributed Syst., 5 (1994), pp. 31–38.
[7] M. L. Furst, J. L. Gross, and L. A. McGeoch, *Finding a maximum-genus graph imbedding*, J. Assoc. Comput. Mach., 35 (1988), pp. 523–534.
[8] Z. Galil and X. Yu, *Short length versions of Menger's theorem*, in Proc. 27th Annual ACM Symposium on Theory of Computing, ACM, Las Vegas, NV, 1995, pp. 499–508.
[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.
[10] D. Gusfield and R. W. Irving, *The Stable Marriage Problem: Structure and Algorithms*, MIT Press Ser. Found. Comput., MIT Press, Cambridge, MA, 1989.
[11] J. Jwo, S. Lakshmivarahan, and S. K. Dhall, *Characterization of node disjoint (parallel) path in star graphs*, in Proc. 5th International Parallel Processing Symposium, Anaheim, CA, 1991, pp. 404–409.
[12] K. Menger, *Zur allgemeinen kurventheorie*, Fund. Math., 10 (1927), pp. 96–115.
[13] J. Misic and Z. Jovanovic, *Routing function and deadlock avoidance in a star graph interconnection network*, J. Parallel Distributed Comput., 22 (1994), pp. 216–228.
[14] S. Sur and P. K. Srimani, *Topological properties of star graphs*, Comput. Math. Appl., 25 (1993), pp. 87–98.
[15] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA, 1983.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 955–969

OPTIMAL BIDDING ALGORITHMS AGAINST CHEATING IN MULTIPLE-OBJECT AUCTIONS*

MING-YANG KAO†, JUNFENG QI‡, AND LEI TAN§

Abstract. This paper studies some basic problems in a multiple-object auction model using methodologies from theoretical computer science. We are especially concerned with situations where an adversary bidder knows the bidding algorithms of all the other bidders. In the two-bidder case, we derive an optimal randomized bidding algorithm, by which the disadvantaged bidder can procure at least half of the auction objects despite the adversary's a priori knowledge of his algorithm. In the general k-bidder case, if the number of objects is a multiple of k, an optimal randomized bidding algorithm is found. If the $k-1$ disadvantaged bidders employ that same algorithm, each of them can obtain at least $1/k$ of the objects regardless of the bidding algorithm the adversary uses. These two algorithms are based on closed-form solutions to certain multivariate probability distributions. In situations where a closed-form solution cannot be obtained, we study a restricted class of bidding algorithms as an approximation to desired optimal algorithms.

Key words. auction theory, bidding algorithms, electronic commerce, automated negotiation mechanisms, software agents, market-based control

AMS subject classifications. 05A99, 60C05, 68R05, 90A09, 90A12, 90D10, 90D13

PII. S0097539796305377

1. Introduction.

This paper investigates some basic problems in auction theory. Broadly speaking, an auction is a market mechanism with explicit or implicit rules for allocating resources and determining prices on the basis of bids from market participants [4, 11, 13, 19]. Auctions are frequently used to price various types of assets. For instance, the U.S. Treasury raises funds by auctioning T-bonds and T-notes, while the Department of the Interior sells mineral rights on federally owned properties via auction. Economists are interested in auctions as an efficient way to price and allocate goods which have no standard market value. Auctions are believed to be the simplest and most familiar means of price determination for multilateral trading without intermediary market makers [11, 13, 19].

In typical auctions, there are one seller and a group of competing buyers who bid to possess the auction objects. Procurements describe situations in which a single buyer wishes to purchase objects from a set of potential suppliers. There are four basic forms of auctions in use [11, 13, 15]. In an *English auction* or *ascending bid auction*, the price of an object is successively raised until only one bidder remains and wins the object. In a *Dutch auction*, which is the converse of an English auction, an initial high price is subsequently lowered until a bidder accepts the current price. In a *first-price sealed-bid auction*, potential buyers submit sealed bids for an object. The highest bidder is awarded the object and pays the amount of his bid. In a *second-price sealed-bid auction*, the highest bidder wins the object but pays a price equal to the

*Received by the editors June 14, 1996; accepted for publication (in revised form) December 11, 1997; published electronically January 29, 1999. An extended abstract appeared in *Proc. of the 3rd Annual Internat. Computing and Combinatorics Conference*, Lecture Notes in Comput. Sci. 1276, Springer-Verlag, Berlin, 1997, pp. 192–201.

http://www.siam.org/journals/sicomp/28-3/30537.html

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

‡Department of Economics, Duke University, Durham, NC 27708 (qijf@econ.duke.edu).

§Department of Computer Science, Duke University, Durham, NC 27708 (lei@cs.duke.edu).

second-highest bid. While there are many other forms of auctions, these four are of the greatest interest.

Previous literature on auction theory mainly studied bidding behavior under the assumption that the objective of bidders is to maximize expected profits in the absence of any budget constraints. Such work concentrates on the allocation of a single object to one of many bidders. Each bidder has a *valuation*, which is his estimate of the value of the object. In the *independent private valuation* (IPV) model, each bidder knows his valuation for the object ex ante. Each bidder's valuation is assumed to be drawn independently from the same probability distribution. In the *common value* (CV) model, it is assumed that bidders obtain imperfect estimates of the value of the object. The bidders all assign the same value to the object ex post. Both models are well studied in auction theory [14, 15, 16, 17].

Very little work in computer science has been conducted on problems related to auctions. Neither auction mechanisms nor bidding algorithms have been formally studied. Nevertheless, computer scientists have realized the importance of auctions as an efficient method of resource allocation [4]. Gagliano, Fraser, and Schaefer [10] applied auction techniques to the allocation of decentralized network resources. Yang, Barash, and Upton [20] proposed an auction-based scheme in which task and resource allocations are determined through negotiations among system entities.

Our work investigates some basic issues of automated negotiation mechanisms which are emerging in electronic commerce and other applications of software agents for resource allocation. For the purpose of maximizing transaction volume and speed [6], we focus on the simultaneous auction of several objects and propose a *multiple-object auction* model. This model further differs from the IPV and CV models in two significant aspects. In this model, each bidder faces a binding budget constraint which is identical to all the bidders. Such constraints can be used to enforce fairness of some form when their compliance is verifiable. Our model also addresses security concerns in electronic transaction environments. We explicitly recognize the possibility that electronically transmitted information about bids may be legitimately or illegitimately revealed against the wishes of their bidders. The IPV and CV models assume that no bidder has such informational advantage on bids or bidding algorithms over other bidders [9]. The assumptions of our model are specified as follows:

- There are a total of k bidders, $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$, each of whom has the same total resource to devote toward winning objects. We normalize this amount to be 1. Assume that $k \geq 2$.
- A total of n objects are auctioned. Assume that $n \geq k$. Each bidder's goal is to maximize the number of objects he wins. The objects are therefore of equal value to a bidder.
- Each bidder submits a sequence of n bids simultaneously for the n objects. Each object is won by the highest bidder at the price of his bid. If m bidders submit the same highest bid for an object, each wins the object with probability $1/m$. (The results of our bidding algorithms in sections 2 and 3 are not affected by the specific tie-breaking rules that are used.) For technical reasons, no zero bid is allowed. (This restriction is used only in section 4.)
- Some bidders may know the bidding algorithms of others. The information structure can be characterized by a directed graph in which an arc from a bidder $\mathcal{B}_i$ to another bidder $\mathcal{B}_j$ means that $\mathcal{B}_i$ knows $\mathcal{B}_j$'s algorithm. For instance, in Figure 1.1, $\mathcal{B}_4$ knows the algorithms of $\mathcal{B}_1$ and $\mathcal{B}_2$; $\mathcal{B}_3$ knows

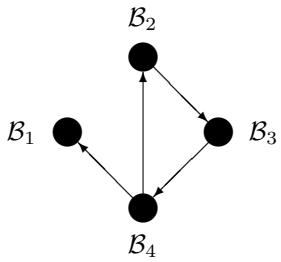


FIG. 1.1. *A graph of information structure.*

$\mathcal{B}_4$'s; $\mathcal{B}_2$ knows $\mathcal{B}_3$'s; $\mathcal{B}_1$ knows only his own. The bidders all compete noncooperatively. We assume that each bidder knows the number of bidders and that of objects.

We analyze the performance of a number of bidding algorithms with which bidders can assign their bids. Almost all the bidding algorithms in this paper are randomized ones. We first study the case of two bidders, i.e., $k = 2$, and then extend the results to the case of multiple bidders. In the two-bidder case, let $\mathcal{B}$ and $\mathcal{A}$ denote the bidders. We assume that $\mathcal{A}$ knows $\mathcal{B}$'s bidding algorithm, while $\mathcal{B}$ does not know $\mathcal{A}$'s; i.e., $\mathcal{B}$ is a disadvantaged bidder and $\mathcal{A}$ an adversary. Here, $\mathcal{A}$ is an oblivious adversary, because although $\mathcal{A}$ knows $\mathcal{B}$'s bidding algorithm, he does not know the outcome of the random choices that $\mathcal{B}$ makes. We give an optimal randomized bidding algorithm for $\mathcal{B}$ by which he can procure at least one half of the objects despite $\mathcal{A}$'s a priori knowledge of his bidding algorithm. The main difficulty with obtaining this optimal bidding algorithm is finding a closed-form solution to a desired multivariate probability distribution [1, 5, 7, 12, 18].

We next study the case where there are more than two bidders, and an adversary bidder knows the bidding algorithms of all the others. If the number of objects is a multiple of the number of bidders, an optimal randomized bidding algorithm is found. If all the disadvantaged bidders employ that same bidding algorithm, each of them can obtain at least $1/k$ of the objects regardless of the bidding algorithm the adversary uses. This bidding algorithm is also based on a closed-form solution to a desired multivariate probability distribution.

When the number of objects is not a multiple of the number of bidders, a closed-form solution of a desired probability distribution cannot be obtained. Motivated by this, we study a class of bidding algorithms to approximate desired optimal algorithms. A bidding algorithm in this class computes an initial sequence of bids, and the actual bid sequence is a random permutation of the initial sequence.

Section 2 describes the optimal bidding algorithm for the disadvantaged bidder in the two-bidder case. In section 3, the optimal randomized bidding algorithm from section 2 is generalized for the multiple-bidder case. In section 4, a class of bidding algorithms are introduced to approximate desired optimal algorithms when a closed-form solution cannot be determined. Section 5 concludes the paper.

For brevity, let $W(\mathcal{B}_i)$ denote the expected number of objects that $\mathcal{B}_i$ wins with a bidding algorithm that is explicitly or implicitly specified.

2. The two-bidder case. This section studies the two-bidder case. We assume that $\mathcal{A}$ knows $\mathcal{B}$'s bidding algorithm, while $\mathcal{B}$ does not know $\mathcal{A}$'s. We give an optimal

randomized bidding algorithm for $\mathcal{B}$ such that $W(\mathcal{B}) = n/2$ despite $\mathcal{A}$'s informational advantage. Since this problem is a zero-sum game, this bound of $n/2$ would be straightforward if von Neumann's min-max theorem were applicable. However, our problem has an infinite pure strategy space, and it is not immediately clear that the min-max theorem is applicable [2, 3, 8, 9, 19].

2.1. $\mathcal{B}$'s optimal bidding algorithm. The following lemma gives an upper bound for the expected number of objects $\mathcal{B}$ can win.

LEMMA 2.1. $W(\mathcal{B}) \leq \frac{n}{2}$.

Proof. Since $\mathcal{A}$ knows $\mathcal{B}$'s bidding algorithm, $\mathcal{A}$ can perform at least as well as $\mathcal{B}$ by employing the same algorithm. Then this lemma follows from the fact that our auction is a zero-sum game. □

Lemma 2.2 describes the marginals of a desired multivariate probability distribution with which $\mathcal{B}$ can form an optimal bidding algorithm.

LEMMA 2.2. *Assume that $\mathcal{B}$ draws his bid sequence $b_1, b_2, \ldots, b_n$ from an n-dimensional probability distribution such that each b_i has the same marginal probability distribution $F_2(b_i)$, where*

$$F_2(b_i) = \begin{cases} \frac{n}{2} \cdot b_i, & b_i \in [0, \frac{2}{n}], \\ 1, & b_i \in (\frac{2}{n}, 1], \end{cases} \tag{2.1}$$

subject to $\sum b_i = 1$. Then, $\mathcal{A}$'s optimal bidding algorithm wins exactly $n/2$ objects on average.

Proof. Let $a_1, a_2, \ldots, a_n$ be $\mathcal{A}$'s optimal bids for the n objects, respectively. $\mathcal{A}$'s probability of winning the ith object is $F_2(a_i)$. Since $\mathcal{B}$'s bids are within $[0, 2/n]$, it is not to $\mathcal{A}$'s advantage to bid over $2/n$. Hence $a_i \leq 2/n$ and $F_2(a_i) = \frac{n}{2} a_i$. $\mathcal{A}$'s optimal bids maximize $W(\mathcal{A})$ as follows:

$$\begin{aligned} \max_{\substack{\sum a_i = 1, \\ 0 \leq a_i \leq \frac{2}{n}}} W(\mathcal{A}) &= \max_{\substack{\sum a_i = 1, \\ 0 \leq a_i \leq \frac{2}{n}}} F_2(a_1) + F_2(a_2) + \cdots + F_2(a_n) \\ &= \max_{\substack{\sum a_i = 1, \\ 0 \leq a_i \leq \frac{2}{n}}} \frac{n}{2} \cdot (a_1 + \cdots + a_n) = \frac{n}{2}. \end{aligned}$$ □

Lemma 2.4 systematically constructs a bid sequence for $\mathcal{B}$ which satisfies the conditions given in Lemma 2.2. We define two additional functions for Lemma 2.4. Let

$$s(v) = \frac{81}{2} \cdot \frac{v}{2 - 3v}. \tag{2.2}$$

Let $h(x, y, z)$ be the function defined on $\{(x, y, z) | 0 \leq x, y, z \leq \frac{1}{3}\}$ such that

$$h(x, y, z) = s(|x - y| + |y - z| + |z - x|). \tag{2.3}$$

LEMMA 2.3. *The function $h(x, y, z)$ is a joint probability density function of x, y and z.*

Proof. Note that $h(x, y, z) \geq 0$. To show that *h(x,y,z)* is a joint probability density function, we need only verify that the integral of *h(x,y,z)* over $\{(x, y, z) | 0 \leq x, y, z \leq \frac{1}{3}\}$ is 1. Let

$$r(x, y) = \int_0^{1/3} h(x, y, z)\, dz. \tag{2.4}$$

Consider the case $x \geq y$. Then

$$\begin{aligned} &\text{if } x \geq y \geq z, \quad h(x,y,z) = s(2(x-z));\\ &\text{if } x \geq z \geq y, \quad h(x,y,z) = s(2(x-y));\\ &\text{if } z \geq x \geq y, \quad h(x,y,z) = s(2(z-y)). \end{aligned}$$

Hence if $x \geq y$,

$$r(x,y) = \int_0^y s(2(x-z))\,dz + \int_y^x s(2(x-y))\,dz + \int_x^{1/3} s(2(z-y))\,dz, \tag{2.5}$$

which equals

$$\frac{9}{2}\left(2\ln(1-3(x-y)) - \ln(3y(1-3x)) - \frac{1-6(x-y)}{1-3(x-y)}\right). \tag{2.6}$$

By symmetry, if $y \geq x$,

$$r(x,y) = \frac{9}{2}\left(2\ln(1-3(y-x)) - \ln(3x(1-3y)) - \frac{1-6(y-x)}{1-3(y-x)}\right).$$

It can be verified that

$$\int_0^{1/3}\int_0^{1/3} r(x,y)\,dxdy = 1.$$

Thus,

$$\int_0^{1/3}\int_0^{1/3}\int_0^{1/3} h(x,y,z)\,dxdydz = 1. \qquad \square$$

LEMMA 2.4. *$\mathcal{B}$ can use the following procedure to draw his bids $b_1, b_2, \ldots, b_n$ such that $\sum b_i = 1$ and the marginal probability distribution of each b_i is as described by (2.1).*

Case 1: *$n = 2m$ is even. $\mathcal{B}$ draws b_1 from the probability distribution F_2 and sets $b_i = b_1$ and $b_{m+i} = \frac{2}{n} - b_1$ for $i = 1, \ldots, m$.*

Case 2: *$n = 2m+1$ is odd. $\mathcal{B}$ draws b_1 from F_2 and then sets $b_i = b_1$ and $b_{m-1+i} = \frac{2}{n} - b_1$ for $i = 1, \ldots, m-1$. For the remaining three bids $b_{2m-1}, b_{2m}, b_{2m+1}$, $\mathcal{B}$ draws (x,y,z) according to h in (2.3) and sets*

$$b_{2m-1} = \frac{3}{n}\left(x - y + \frac{1}{3}\right), \quad b_{2m} = \frac{3}{n}\left(y - z + \frac{1}{3}\right), \quad b_{2m+1} = \frac{3}{n}\left(z - x + \frac{1}{3}\right). \tag{2.7}$$

Proof. Note that $\sum_{i=1}^n b_i = 1$, whether n is even or odd.

Case 1. This lemma is correct since if a random variable X is drawn from the uniform probability distribution on $[0, \frac{2}{n}]$, then $\frac{2}{n} - X$ has the same probability distribution.

Case 2. The proof of Case 1 shows that the marginal probability distribution of each b_i is F_2 for $i = 1, \ldots, 2m-2$. It remains to show that $b_{2m-1}, b_{2m}, b_{2m+1}$ are also distributed the same way. Because these three random variables are symmetric to each other in (2.7), we discuss only b_{2m-1} in detail. Let $t = x - y + \frac{1}{3}$. Since x and y are defined on $[0, \frac{1}{3}]$, t is defined on $[0, \frac{2}{3}]$. We have two cases: $t \in [0, \frac{1}{3}]$ and

$t \in [\frac{1}{3}, \frac{2}{3}]$. The two cases are symmetric, and we discuss only the latter. Let *G(t)* denote the probability distribution of *t*. Then

$$G(t) = 1 - \int\int_{u-v+\frac{1}{3}\geq t, 0\leq u,v\leq \frac{1}{3}} r(u,v)\,dvdu = 1 - \int_{t-1/3}^{1/3}\int_{0}^{u-t+1/3} r(u,v)\,dvdu.$$

Since $u \geq u-t+1/3$, $r(u,v)$ can take the form of (2.6), and we can obtain $G(t) = \frac{3}{2}t$. Since $b_{2m-1} = \frac{3}{n}t$, F_2 is the probability distribution of b_{2m-1}. □

THEOREM 2.5. *The bidding algorithm given in Lemma* 2.4 *is optimal for* $\mathcal{B}$ *and ensures* $\mathcal{B}$ *at least* $n/2$ *objects in expected terms.*

Proof. Lemma 2.1 gives an upper bound for $W(\mathcal{B})$. Lemmas 2.2 and 2.4 give an upper bound for $W(\mathcal{A})$, which in turn gives a matching lower bound for $W(\mathcal{B})$ because $W(\mathcal{B}) + W(\mathcal{A}) = n$. □

2.2. Deriving the joint probability density function $h(x, y, z)$. The most difficult step of obtaining the function h is guessing that *x,y,* and *z* appear together as $|x-y|+|y-z|+|z-x|$. It is worthwhile to show the derivation of the function s in (2.2) that gives the joint probability density function *h(x,y,z).* As in (2.4), let *r(x,y)* be the probability distribution of *(x,y).* Also let $t = x - y + \frac{1}{3}$. Since $t = \frac{n}{3}b_{2m-1}$ needs to be uniformly distributed over $[0, \frac{2}{3}]$, we need to have

$$1 - \int_{t-1/3}^{1/3}\int_{0}^{u-t+1/3} r(u,v)\,dvdu = \frac{3}{2}t \quad \text{for all } t \in \left[\frac{1}{3}, \frac{2}{3}\right] \tag{2.8}$$

and

$$\int_{0}^{t}\int_{u-t+1/3}^{1/3} r(u,v)\,dvdu = \frac{3}{2}t \quad \text{for all } t \in \left[0, \frac{1}{3}\right].$$

These two cases are symmetric, and we discuss only the case given by (2.8) in detail. For notational simplicity, let

$$s(2v) = q(v), \quad \int^{u} q(v)\,dv = p(u), \quad r_2(x,y) = \frac{\partial r(x,y)}{\partial y}.$$

Differentiating (2.8) with respect to t twice, we obtain

$$\int_{t-1/3}^{1/3} r_2\left(u, u-t+\frac{1}{3}\right)\,du + r\left(t-\frac{1}{3}, 0\right) = 0. \tag{2.9}$$

Since $x \geq y$ in (2.8), the following is derived from (2.5):

$$r_2(u,v) = -(u-v)q'(u-v) - q\left(\frac{1}{3} - v\right) + q(u-v).$$

Then

$$\begin{aligned}\int_{t-1/3}^{1/3} & r_2\left(u, u-t+\frac{1}{3}\right)\,du \\ &= -\left(t-\frac{1}{3}\right)\left(\frac{2}{3}-t\right)q'\left(t-\frac{1}{3}\right) + p\left(t-\frac{1}{3}\right) - p\left(\frac{1}{3}\right) + q\left(t-\frac{1}{3}\right)\left(\frac{2}{3}-t\right).\end{aligned} \tag{2.10}$$

We obtain from (2.5)

$$r\left(t-\frac{1}{3},0\right)=\left(t-\frac{1}{3}\right)q\left(t-\frac{1}{3}\right)+p\left(\frac{1}{3}\right)-p\left(t-\frac{1}{3}\right). \tag{2.11}$$

Setting $w=t-\frac{1}{3}$, we can derive the following differential equation from (2.9), (2.10), and (2.11):

$$w(1-3w)q'(w)=q(w).$$

The solution to the differential equation is

$$q(w)=c\frac{3w}{1-3w},$$

where c is a constant. Therefore

$$s(v)=c\frac{3v}{2-3v}.$$

Since *h(x,y,z)* is a probability density function for $(x,\ y,\ z)$, c is set to $\frac{27}{2}$ to satisfy

$$\int_0^{1/3}\int_0^{1/3}\int_0^{1/3}h(x,y,z)\,dxdydz=1.$$

3. The multiple-bidder case. This section generalizes the results in section 2 to give an optimal randomized bidding algorithm for the case of multiple bidders. We assume that the bidding algorithms of $k-1$ bidders are known to a single adversary bidder $\mathcal{A}$. If all the $k-1$ disadvantaged bidders employ our bidding algorithm, each of them wins at least a fraction $1/k$ of the objects regardless of the bidding algorithm the adversary uses.

LEMMA 3.1. *Assume that each of the $k-1$ disadvantaged bidders independently draws his bid sequence $b_1,b_2,\ldots,b_n$ from an n-dimensional probability distribution such that each b_i has the same marginal probability distribution $F_k(b_i)$, where*

$$F_k(b_i)=\begin{cases}\left(\frac{n}{k}\cdot b_i\right)^{\frac{1}{k-1}} & \text{if } b_i\in[0,\frac{k}{n}],\\ 1 & \text{if } b_i\in(\frac{k}{n},1],\end{cases} \tag{3.1}$$

subject to $\sum b_i=1$. Then, $W(\mathcal{A})$ is at most n/k.

Proof. Let $b_{i,j}$ denote the bid on the ith object of the jth disadvantaged bidder. Let a_i be $\mathcal{A}$'s bid on the ith object. Because the bids of the $k-1$ disadvantaged bidders are within $[0,k/n]$, $\mathcal{A}$ has no incentive to bid over k/n. Thus $a_i\le k/n$ and $F(a_i)=\left(\frac{n}{k}\cdot a_i\right)^{\frac{1}{k-1}}$. Since bids from different disadvantaged bidders are independent,

$$\begin{aligned}&\text{Prob}\{a_i\text{wins the } i\text{th object}\}\\ &=\text{Prob}\{b_{i,1}\le a_i\}\cdot\text{Prob}\{b_{i,2}\le a_i\}\cdots\text{Prob}\{b_{i,k-1}\le a_i\}\\ &=(F_k(a_i))^{k-1}\\ &=\frac{n}{k}\cdot a_i.\end{aligned}$$

From the fact that $\sum a_i\le 1$, $\mathcal{A}$ wins exactly n/k objects on average. ∎

It appears quite difficult to find a closed-form solution to a joint probability distribution whose marginals are as described by (3.1).

CONJECTURE 3.2. *There exists an n-dimensional joint probability distribution such that its marginal probability distribution of every component is as described by* (3.1)*, while the components from all dimensions sum to* 1.

For $k = 2$, this conjecture has been proved in section 2. If n is a multiple of k, we prove this conjecture as follows. Let

$$e(b_1, b_2, \ldots, b_k) = \begin{cases} (b_1 b_2 \cdots b_k)^{\frac{1}{k-1}-1}, & b_1 + b_2 + \cdots + b_k = 1,\ b_i > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Let

$$\alpha = \int_{b_1+\cdots+b_k=1} e(b_1, b_2, \ldots, b_k) db_1 db_2 \cdots db_{k-1}.$$

Normalizing e by using α, we have

$$g(b_1, b_2, \ldots, b_k) = \begin{cases} \frac{(b_1 b_2 \cdots b_k)^{\frac{1}{k-1}-1}}{\alpha}, & b_1 + b_2 + \cdots + b_k = 1,\ b_i > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

With this normalization, g is a probability density function of $(b_1, b_2, \ldots, b_k)$. For example, if $n = k = 3$, the probability density function shown in (3.2) is

$$g(b_1, b_2, b_3) = \begin{cases} \frac{1}{2\pi\sqrt{b_1 b_2 b_3}}, & b_1 + b_2 + b_3 = 1,\ b_i > 0, \\ 0 & \text{otherwise.} \end{cases}$$

The following lemma proves Conjecture 3.2 for the case $n = k$.

LEMMA 3.3. *If $n = k$ and the bid sequence $b_1, b_2, \ldots, b_n$ is drawn from the n-dimensional joint probability distribution in* (3.2)*, then the marginal probability distribution for each b_i is as described by* (3.1).

Proof. Because $b_1, b_2, \ldots, b_k$ are symmetric for g, we need only show that the probability distribution of b_k is as described in (3.1). Let

$$b_i = (1 - b_k) u_{k-i},\ i = 2, \ldots, k-1.$$

Then

$$db_i = (1 - b_k) du_{k-i}.$$

Let

$$\alpha' = \int_{u_1+\cdots+u_{k-1}=1} (u_1 u_2 \cdots u_{k-1})^{\frac{1}{k-1}-1} du_{k-2} du_{k-3} \cdots du_1.$$

Note that $\alpha = (k-1) \cdot \alpha'$. The probability distribution of b_k equals

$$\int_{\substack{0 \le w \le b_k \\ b_1+b_2+\cdots+b_{k-1}+w=1}} g(b_1, \ldots, b_{k-1}, w) db_2 \cdots db_{k-1} dw$$

$$= \frac{1}{\alpha} \cdot \int_0^{b_k} \int_0^{1-w} \int_0^{1-w-b_{k-1}} \cdots \int_0^{1-w-\cdots-b_3}$$

$$
\begin{aligned}
&((1-b_2-\cdots-w)b_2b_3\cdots w)^{\frac{1}{k-1}-1}db_2\ db_3\cdots db_{k-1}dw\\
&=\frac{1}{\alpha}\cdot\int_0^{b_k} w^{\frac{1}{k-1}-1}\int_0^1\int_0^{1-u_1}\cdots\int_0^{1-u_1-\cdots-u_{k-3}}\\
&\quad(u_1u_2\cdots u_{k-2}(1-u_1-\cdots-u_{k-2}))^{\frac{1}{k-1}-1}\,du_{k-2}\cdots du_2\,du_1\,dw\\
&=\frac{1}{\alpha}\cdot\int_0^{b_k}\alpha' w^{\frac{1}{k-1}-1}\,dw\\
&=\frac{\alpha'}{\alpha}(k-1)a^{\frac{1}{k-1}}\\
&=F_k(b_k).\qquad\square
\end{aligned}
$$

The following lemma extends Lemma 3.3 to the case $n = k{\cdot}m$ for some integer.

LEMMA 3.4. *If $n = k{\cdot}m$ for some integer m, there exists a procedure to generate a bid sequence $b_1, b_2, \ldots, b_n$ such that the probability distribution for each b_i can be described by* (3.1)*, and the bids b_i sum to* 1.

Proof. If $m = 1$, the lemma is the same as Lemma 3.3. If $m > 1$, we divide the objects into m groups of k objects each and employ Lemma 3.3 to obtain bids for the first group. We then set the bids for the other $m-1$ groups to the corresponding bids for the first group. We scale every bid by a factor of $\frac{1}{m}$ so that the bids sum to 1. This gives the desired probability distribution. □

THEOREM 3.5. *If $n = k{\cdot}m$ for some integer m and the disadvantaged bidders all employ the bidding algorithm characterized by Lemma* 3.4*, then each can obtain at least n/k objects in expected terms, which is optimal.*

Proof. From Lemmas 3.1 and 3.4 and the fact that our game is a zero-sum game, the $k-1$ disadvantaged bidders win $\frac{k-1}{k}{\cdot}n$ objects in total. Since they all use the same bidding algorithm, by symmetry each of them wins n/k objects. This upper bound of n/k is also a lower bound since the adversary can always win at least n/k objects by employing the same bidding algorithm as the disadvantaged bidders. □

4. Position-randomized bidding algorithms. In section 3, an optimal randomized bidding algorithm for the bidders with informational disadvantage is derived for the case where the number of objects is a multiple of that of bidders. This algorithm is based on a closed-form solution to a desired multivariate probability distribution. If n is not a multiple of k, a closed-form solution cannot be obtained with our current techniques. Motivated by this, we consider situations where all the bidders are restricted to a class of bidding algorithms called *position-randomized bidding algorithms*. A position-randomized bidding algorithm consists of two steps. Step 1 deterministically selects an initial sequence of n bids. Step 2 permutes the sequence. The ith element of the final sequence is the actual bid for the ith object. As in section 3, we assume that all the disadvantaged bidders adopt an identical bid sequence at step 1 and the same probability distribution at step 2. A position-randomized bidding algorithm can be considered as an approximation to optimal bidding algorithms desired for resolving Conjecture 3.2 in section 3.

The next lemma examines how probability distributions chosen at step 2 affect the expected numbers of objects bidders win.

LEMMA 4.1. *For a given initial bid sequence $a_1, a_2, \ldots, a_n$ of $\mathcal{A}$ and a given initial bid sequence $b_1, b_2, \ldots, b_n$ of the disadvantaged bidders,*

- *W_1 denotes the expected number of objects $\mathcal{A}$ wins by using the uniform probability distribution while the disadvantaged bidders may use any arbitrary probability distribution;*

- *W_2 denotes the expected number of objects $\mathcal{A}$ wins without permuting his initial bid sequence while the disadvantaged bidders employ the uniform probability distribution;*
- *W_3 denotes the expected number of objects $\mathcal{A}$ wins using any given probability distribution while the disadvantaged bidders employ the uniform probability distribution.*

If $a_1, a_2, \ldots, a_n$ are all different from $b_1, b_2, \ldots, b_n$, then $W_1 \geq W_2 = W_3$.

Proof. For each a_i,

- $W_{1,i}$ denotes the expected number of objects a_i wins if $\mathcal{A}$ uses the uniform probability distribution while the disadvantaged bidders may use any arbitrary probability distribution;
- $W_{2,i}$ denotes the expected number of objects a_i wins if $\mathcal{A}$ does not permute his initial bid sequence and the disadvantaged bidders employ the uniform probability distribution;
- $W_{3,i}$ denotes the expected number of objects a_i wins if $\mathcal{A}$ uses a given probability distribution and the disadvantaged bidders employ the uniform probability distribution.

Since $W_j = W_{j,1} + \cdots + W_{j,n}$ for $j \in \{1, 2, 3\}$, it suffices to prove that $W_{1,i} \geq W_{2,i} = W_{3,i}$. Without loss of generality, assume that $b_1 \leq b_2 \leq \cdots \leq b_n$. Let p be the largest index such that $b_p < a_i$; if no such b_p exists, let $p = 0$. Since $a_i < b_j$ for $j = p+1, \ldots, n$,

$$W_{2,i} = \left(\frac{p}{n}\right)^{k-1}.$$

To calculate $W_{3,i}$, let $Q_{q,r}$ be the probability that $\mathcal{A}$ places a_q on the rth object. Then

$$\begin{aligned} W_{3,i} &= \sum_{r=1}^{n} \text{Prob}\{a_i \text{ wins the } r\text{th object}\} \\ &= \sum_{r=1}^{n} Q_{i,r} \left(\frac{p}{n}\right)^{k-1}. \end{aligned}$$

Since $\sum_{r=1}^{n} Q_{i,r} = 1$,

$$W_{2,i} = W_{3,i}.$$

To calculate $W_{1,i}$, let $P_{q,r}$ be the probability that a disadvantaged bidder places b_q on the rth object. Then

$$\begin{aligned} W_{1,i} &= \sum_{r=1}^{n} \frac{1}{n} \cdot \text{Prob}\{a_i \text{ wins the } r\text{th object}\} \\ &= \sum_{r=1}^{n} \frac{1}{n} (P_{1,r} + P_{2,r} + \cdots + P_{p,r})^{k-1}. \end{aligned}$$

Since $\sum_{r=1}^{n} P_{q,r} = 1$ for each q, by Hödel's inequality

$$W_{1,i} \geq W_{2,i}. \qquad \square$$

Since $W_1 \geq W_3$ in Lemma 4.1, the disadvantaged bidders should always use the uniform probability distribution at step 2. Since $W_2 = W_3$, we may assume that $\mathcal{A}$

does not permute his initial bid sequence whenever the disadvantaged bidders use the uniform probability distribution. We next use Lemma 4.1 to derive a lower bound for the expected number of objects $\mathcal{A}$ can win. Let

$$\begin{aligned}
\epsilon &= \text{a positive infinitesimal amount;}\\
\beta &= \sum_{i=1}^{n} i^{k-1};\\
c_i &= \frac{i^{k-1}}{\beta};\\
E &= \{c_0, c_1, c_2, \ldots, c_n\};\\
D &= \{\epsilon, c_2+\epsilon, c_3+\epsilon, \ldots, c_n+\epsilon\}.
\end{aligned}$$

LEMMA 4.2. *$\mathcal{A}$ can win at least $\frac{\beta-1}{n^{k-1}}$ objects on average for any given initial bid sequence and probability distribution employed by the disadvantaged bidders.*

Proof. Given an initial bid sequence $b_1 \le b_2 \le \cdots \le b_n$ of the $k-1$ disadvantaged bidders, $\mathcal{A}$ chooses his initial bid sequence to be $b_1-(n-1)\epsilon, b_2+\epsilon, \ldots, b_n+\epsilon$. Since $\mathcal{A}$'s bids are different from $b_1, b_2, \ldots, b_n$, in light of Lemma 4.1, we may assume that the disadvantaged bidders permute their bids with the uniform probability distribution. Consequently, the expected number of objects won by $\mathcal{A}$ is as desired. □

We next prove a matching upper bound for the expected number of objects $\mathcal{A}$ can win.

LEMMA 4.3. *If the disadvantaged bidders employ $c_1, c_2, \ldots, c_n$ as their initial bid sequence and permute it with the uniform probability distribution, then $\mathcal{A}$ has an optimal initial bid sequence $a'_1, a'_2, \ldots, a'_n$ such that $a'_i \in D$ for all i.*

Proof. Given an optimal initial bid sequence $a_1, a_2, \ldots, a_n$ of $\mathcal{A}$, we show that this sequence can be transformed into a desired sequence $a'_1, a'_2, \ldots, a'_n$ without decreasing $W(\mathcal{A})$. Let m be the number of $\mathcal{A}$'s bids that are in E. There are three cases.

Case 1: $m = 0$. For each a_i, let $a'_i = c_j + \epsilon$ where j is the biggest index such that $c_j < a_i$. Then the expected number of objects won by $a'_1, a'_2, \ldots, a'_n$ is the same as that of $a_1, a_2, \ldots, a_n$, and the new sequence is as desired.

Case 2: $m = 1$. This case is impossible since $\mathcal{A}$ can increase $W(\mathcal{A})$ by decreasing one of his bids outside E by ϵ and increasing the one that is in E by ϵ.

Case 3: $m \ge 2$. Without loss of generality, let $a_1, a_2, \ldots, a_m$ be $\mathcal{A}$'s m bids in E in the increasing order. We first decrease a_1 by $(m-1)\epsilon$ and increase a_j by ϵ for $j = 2, \ldots, m$. As shown below, this adjustment never decreases $W(\mathcal{A})$. Then, since $\mathcal{A}$'s adjusted bids are not in E, his new initial bid sequence can be further transformed into a desired sequence as in Case 1. Let w_1 be the decreased amount of $W(\mathcal{A})$ resulted from decreasing a_1. Let w_j be the increased amount of $W(\mathcal{A})$ resulted from increasing a_j for $j = 2, \ldots, m$. We need to show that $-w_1 + w_2 + \cdots + w_m \ge 0$. It suffices to prove that $w_2 - w_1 \ge 0$. Let $\#_p$ denote the expected number of objects a_j wins if $a_j = c_p$. Then

$$\begin{aligned}
\#_p &= \sum_{i=0}^{k-1} \frac{1}{i+1} \cdot \text{Prob}\{a_j \text{ ties with } i \text{ disadvantaged bidders and beats the others}\}\\
&= \sum_{i=0}^{k-1} \frac{1}{i+1} \binom{k-1}{i} \left(\frac{1}{n}\right)^{i} \left(\frac{p-1}{n}\right)^{k-1-i}
\end{aligned}$$

$$= \sum_{i=0}^{k-1} \frac{1}{i+1} \binom{k-1}{i} \frac{(p-1)^{k-1-i}}{n^{k-1}}$$
$$= \left(\frac{p-1}{n}\right)^{k-1} \cdot \frac{1}{k} \cdot \left(\left(\frac{1}{p-1}\right)^{k} - 1\right).$$

Assume that $a_1 = c_q$ and $a_2 = c_r$. Then $w_1 = \#_q - (\frac{q-1}{n})^{k-1}$ and $w_2 = (\frac{r}{n})^{k-1} - \#_r$. Note that w_2 increases with r. Since $q \le r$, w_2 is minimized when $a_1 = a_2$ and thus $q = r$. Consequently,

$$\begin{aligned}
w_2 - w_1 &\ge \left(\frac{q}{n}\right)^{k-1} - \#_q - \#_q + \left(\frac{q-1}{n}\right)^{k-1} \\
&= \left(\frac{q}{n}\right)^{k-1} + \left(\frac{q-1}{n}\right)^{k-1} - 2 \cdot \sum_{i=0}^{k-1} \frac{1}{i+1} \binom{k-1}{i} \frac{(q-1)^{k-1-i}}{n^{k-1}} \\
&= \left(\frac{q}{n}\right)^{k-1} - \left(\frac{q-1}{n}\right)^{k-1} - 2 \cdot \sum_{i=1}^{k-1} \frac{1}{i+1} \binom{k-1}{i} \frac{(q-1)^{k-1-i}}{n^{k-1}} \\
&= \sum_{i=1}^{k-1} \binom{k-1}{i} \left(\frac{q-1}{n}\right)^{i} \left(\frac{1}{n}\right)^{k-1-i} - 2 \cdot \sum_{i=1}^{k-1} \frac{1}{i+1} \binom{k-1}{i} \frac{(q-1)^{k-1-i}}{n^{k-1}} \\
&= \sum_{i=1}^{k-1} \left(1 - \frac{2}{i+1}\right) \binom{k-1}{i} \frac{(q-1)^{k-1-i}}{n^{k-1}} \\
&\ge 0. \qquad \square
\end{aligned}$$

LEMMA 4.4. *If the $k-1$ disadvantaged bidders all employ $c_1, c_2, \ldots, c_n$ as their initial bid sequence and permute it with the uniform probability distribution, then $\mathcal{A}$ can win at most $\frac{\beta-1}{n^{k-1}}$ objects on average.*

Proof. From Lemma 4.3, $\mathcal{A}$ has an optimal initial bid sequence $a'_1, a'_2, \ldots, a'_n$, such that for all j, $a'_j \in D$. If $a'_j = \epsilon$, then it cannot win any object. If $a'_j = c_i + \epsilon$, then it can win $(\frac{i}{n})^{k-1}$ objects on average. The unit price $\mathcal{A}$ pays for these objects is strictly greater than

$$\frac{\frac{i^{k-1}}{\beta}}{(\frac{i}{n})^{k-1}} = \frac{n^{k-1}}{\beta}.$$

Since the expected number of objects won by such a'_j is an integral multiple of $\frac{1}{n^{k-1}}$, $W(\mathcal{A}) = m \cdot \frac{1}{n^{k-1}}$ for some integer m, and

$$m \cdot \frac{1}{n^{k-1}} \cdot \frac{n^{k-1}}{\beta} < 1.$$

Since m is an integer, $m \le \beta - 1$ and thus $W(\mathcal{A}) \le \frac{\beta-1}{n^{k-1}}$. $\square$

THEOREM 4.5. *If the disadvantaged bidders all employ $c_1, c_2, \ldots, c_n$ as their initial bid sequence and permute it with the uniform probability distribution, then each of them can win at least $1/k$ of $n - \frac{\beta-1}{n^{k-1}}$ objects on average, which is optimal.*

Proof. By Lemma 4.4, $\mathcal{A}$ wins at most $\frac{\beta-1}{n^{k-1}}$ objects on average. By Lemma 4.2, this upper bound is also the lower bound of the expected number of objects $\mathcal{A}$ can win. This theorem follows from the fact that our auction is a zero-sum game. $\square$

5. Extensions and open problems. This paper leaves several problems unsolved. Section 3 still lacks an optimal randomized bidding algorithm for the disadvantaged bidders when n is not a multiple of k. In section 4, if zero bids are allowed, the initial bid sequence $c_1, \ldots, c_n$ is no longer optimal for the disadvantaged bidders. In general, if disadvantaged bidders do not use identical bidding algorithms, it is not even clear what an optimal bidding algorithm should mean, especially for a more complicated information structure than discussed in this paper.

Our model can be extended to study sequential bidding. The bidders submit sealed bids for an object. Once that object is sold, the next object is auctioned the same way until all the objects are sold. For the case where n is a multiple of k, an optimal sequential bidding algorithm is described in the following lemma.

LEMMA 5.1. *If n is a multiple of k and the objects are auctioned sequentially, then a bidder can obtain n/k objects by bidding k/n on every object until his budget is exhausted.*

Proof. Assume that $\mathcal{B}_i$ employs this bidding algorithm. From his budget constraint, he wins at most n/k objects. This upper bound is also a lower bound. To prove this claim by contradiction, assume that $\mathcal{B}_i$ wins fewer than n/k objects and thus does not exhaust all his budget. Then, the total number of objects won by the other bidders exceeds $\frac{k-1}{k}\cdot n$. Because n is a multiple of k and $\mathcal{B}_i$ has not exhausted his budget, every object's winning bid must be at least k/n. Therefore, the total of the winning bids of the other bidders exceeds $k-1$. Since this contradicts the budget constraint, $\mathcal{B}_i$ can win at least n/k objects. □

Our model can also be extended to the case where the objects may have distinct values. In a general setting, the objects are divided into m groups. Let n_i denote the number of objects in the ith group, which may be any positive real number. The bidders are asked to submit bids for the m groups simultaneously. Whoever bids the highest for a group obtains all the objects in that group subject to the same tie-breaking rule. An m-group auction is equivalent to an auction of m objects with distinct values where n_i is the value of the ith group. As before, assume that an adversary bidder $\mathcal{A}$ knows the bidding algorithms of the other $k-1$ bidders and that all those disadvantaged bidders employ the same bidding algorithm.

LEMMA 5.2. *Assume that each disadvantaged bidder bids $n_i \cdot b_i$ for the ith group, where $b_1, b_2, \ldots, b_m$ are drawn from an m-dimensional probability distribution such that the marginal probability distribution of each b_i is F_k subject to $n_1 \cdot b_1 + \cdots + n_m \cdot b_m = 1$. Then the optimal expected number of objects won by $\mathcal{A}$ is n/k.*

Proof. Let $n_i \cdot b_{i,j}$ denote the bid on the ith object by the jth disadvantaged bidder. Let $n_i \cdot a_i$ be $\mathcal{A}$'s bid on the ith object. Because $b_1, b_2, \ldots, b_m \in [0, k/n]$, $\mathcal{A}$ has no incentive to set a_i greater than k/n. Thus, $a_i \le k/n$ and $F_k(a_i) = \left(\frac{n}{k}\cdot a_i\right)^{\frac{1}{k-1}}$. Since bids from different disadvantaged bidders are independent,

$$\begin{aligned} &\text{Prob}\{n_i \cdot a_i \text{ wins the } i\text{th object}\} \\ &= \text{Prob}\{b_{i,1} \le a_i\} \cdot \text{Prob}\{b_{i,2} \le a_i\} \cdots \text{Prob}\{b_{i,k-1} \le a_i\} \\ &= (F_k(a_i))^{k-1} \\ &= \frac{n}{k} \cdot a_i. \end{aligned}$$

$\mathcal{A}$ maximizes $W(\mathcal{A})$ as follows:

$$\max_{\substack{\sum n_i \cdot a_i = 1 \\ 1 \le a_i \le \frac{k}{n}}} W(\mathcal{A})$$

$$= \max_{\substack{\sum n_i \cdot a_i = 1 \\ 1 \le a_i \le \frac{k}{n}}} n_1 \cdot \left(\frac{n}{k} \cdot a_1\right) + n_2 \cdot \left(\frac{n}{k} \cdot a_2\right) + \cdots + n_m \cdot \left(\frac{n}{k} \cdot a_m\right)$$

$$= \frac{n}{k}. \qquad \square$$

CONJECTURE 5.3. *There exists an m-dimensional probability distribution for $(b_1, b_2, \ldots, b_m)$ subject to the constraint $n_1 \cdot b_1 + n_2 \cdot b_2 + \cdots + n_m \cdot b_m = 1$ such that the marginal probability distribution of each b_i is as described by* (3.1).

Remark. This conjecture can be reduced to the case $m = 2$ or $m = 3$.

We conclude the paper with two research directions. One is to consider general information structures as specified by arbitrary directed graphs; the other is to investigate more general budget constraints beyond the homogeneous one of this paper. It would be of significance to design bidding algorithms that can optimally or approximately achieve game-theoretic equilibria in meaningful combinations of these two directions.

Acknowledgments. We are indebted to Phil Long, Kasturi Varadarajan, and Professor Dennis Yang at the Economics Department of Duke University for very helpful comments. We also wish to thank the anonymous referees for contagious enthusiasm toward this work and unusually thoughtful comments and detailed suggestions.

REFERENCES

[1] V. BENES AND J. STEPAN, *Extremal solutions in the marginal problem*, in Advances in Probability Distributions with Given Marginals: Beyond the Copulas, Kluwer Academic Publishers, Norwell, MA, 1994, pp. 189–206.

[2] D. BLACKWELL, *An analog of the minimax theorem for vector payoffs*, Pacific J. Math., 6 (1956), pp. 1–8.

[3] D. BLACKWELL AND M. A. GIRSHICK, *Theory of Games and Statistical Decisions*, John Wiley, New York, 1954.

[4] S. H. CLEARWATER, ED., *Market-Based Control, a Paradigm for Distributed Resource Allocation*, World Scientific, River Ridge, NJ, 1996.

[5] G. DALL'AGLIO, *Fréchet classes: The beginnings*, in Advances in Probability Distributions with Given Marginals: Beyond the Copulas, Kluwer Academic Publishers, Norwell, MA, 1994, pp. 1–12.

[6] FEDERAL COMMUNICATIONS COMMISSION, *Fifth Report and Order, FCC* 94-178*,* Washington, DC, 1994.

[7] M. FRÉCHET, *Sur les tableaux de corrélation dont les marges sont données*, Ann. Univ. Lyon. Sect. A (3), 14 (1951), pp. 53–77.

[8] Y. FREUND AND R. SCHAPIRE, *Game theory, on-line prediction and boosting*, in Proc. 9th Annual Conference on Computational Learning Theory, ACM, New York, 1996, pp. 325–332.

[9] D. FUDENBERG AND J. TIROLE, *Game Theory*, MIT Press, Cambridge, MA, 1991.

[10] R. A. GAGLIANO, M. D. FRASER, AND M. E. SCHAEFER, *Auction allocation of computing resources*, Comm. ACM, 38 (1995), pp. 88–99.

[11] K. HENDRICKS AND H. J. PAARSH, *A survey of recent empirical work concerning auctions*, Canad. J. Econom., 28 (1995), pp. 403–426.

[12] S. KOTZ AND J. P. SEEGER, *A new approach to dependence in multivariate distributions*, in Advances in Probability Distributions with Given Marginals: Beyond the Copulas, Kluwer Academic Publishers, Norwell, MA, 1994, pp. 113–128.

[13] J. MCMILLAN AND R. P. MCAFEE, *Auctions and bidding*, J. Econom. Literature, 25 (1987), pp. 699–738.

[14] P. R. MILGROM, *Good news and bad news: Representation theorems and applications*, Bell J. Econom., 12 (1981), pp. 380–391.

[15] P. R. Milgrom and R. J. Weber, *A theory of auctions and competitive bidding*, Econometrica, 50 (1982), pp. 1089–1122.
[16] R. B. Myerson, *Optimal auction design*, Math. Oper. Res., 6 (1981), pp. 58–73.
[17] C. Pitchik and A. Schotter, *Perfect equilibria in budget-constrained sequential auctions: An experimental study*, RAND J. Econom., 19 (1988), pp. 363–388.
[18] B. Schweizer, *Thirty years of copulas*, in Advances in Probability Distributions with Given Marginals: Beyond the Copulas, Kluwer Academic Publishers, Norwell, MA, 1994, pp. 13–50.
[19] R. Wilson, *Strategic analysis of auctions*, in Handbook of Game Theory with Economic Applications, Vol. 1, R. J. Aumann and S. Hart, eds., Elsevier Science, New York, 1992, pp. 227–279.
[20] E. H. Yang, M. M. Barash, and D. M. Upton, *Accommodation of priority parts in a distributed computer-controlled manufacturing system with aggregate bidding schemes*, in Proc. 2nd Industrial Engineering Research Conference, 1993, pp. 827–831.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 970–983

THREE-PROCESSOR TASKS ARE UNDECIDABLE*

ELI GAFNI† AND ELIAS KOUTSOUPIAS†

Abstract. We show that no algorithm exists for deciding whether a finite task for three or more processors is wait-free solvable in the asynchronous read-write shared-memory model. This impossibility result implies that there is no constructive (recursive) characterization of wait-free solvable tasks. It also applies to other shared-memory models of distributed computing, such as the comparison-based model.

Key words. asynchronous distributed computation, task-solvability, wait-free computation, contractibility problem

AMS subject classifications. 68Q05, 68Q22

PII. S0097539796305766

1. Introduction.

A fundamental area in the theory of distributed computation is the study of asynchronous wait-free shared-memory distributed algorithms. Characterizing the class of distributed tasks that can be solved, no matter how "inefficiently," is an important step toward a complexity theory for distributed computation. A breakthrough was the demonstration by Fischer, Lynch, and Paterson [6] that certain simple tasks, such as *consensus*, are not solvable. Subsequently, Biran, Moran, and Zaks [1] gave a complete characterization of the tasks solvable by two processors and of tasks that can be solved when only one processor can fail. Recently, three teams [3, 9, 17] independently extended this result by providing powerful necessary conditions for task solvability, which enabled them to show that the *k-set agreement task* is not solvable for more than k processors. Finally, Herlihy and Shavit [10, 11] gave a simple condition that is necessary and sufficient for a given task to admit a wait-free protocol. This condition was extended by Borowsky and Gafni [2] to the more general model of asynchronous distributed computation of resiliency and set-consensus.

Here, we put the quest for complete characterization of solvable tasks to an abrupt end by showing that *there is no recursive characterization of wait-free tasks.* More precisely, we show that the problem of deciding whether a given finite task for three or more processors admits a wait-free protocol is undecidable. We also show that this holds for the comparison-based model (when processors can only compare their IDs). An immediate consequence of our result is that for any recursive function *f(s)* there are finite *solvable* tasks of size (number of input-output tuples) s that cannot be solved by any protocol in less than *f(s)* steps. Unfortunately, this may hamper the development of a "complexity theory" of asynchronous distributed computation.

Our proof exploits a surprising connection between distributed computation and topology. In particular, we give a reduction from the *contractibility problem* to the *task-solvability problem.* The contractibility problem asks whether a given loop of a

* Received by the editors June 28, 1996; accepted for publication (in revised form) August 29, 1997; published electronically January 29, 1999. A preliminary abstract of this paper appeared as *3-processor tasks are undecidable,* in Proc. 14th Annual ACM Symposium on Principles of Distributed Computing, Ottawa, Ont., Canada, ACM, New York, 1995, p. 271.

http://www.siam.org/journals/sicomp/28-3/30576.html

†Computer Science Department, UCLA, 3731 Boelter Hall, Los Angeles, CA 90095 (eli@cs.ucla.edu, elias@cs.ucla.edu).

simplicial complex is contractible, that is, whether it can be continuously transformed into a point.

The history of the contractibility problem goes back to Poincaré and Dehn at the beginning of the twentieth century (see [19]). Dehn [4] noticed that the contractibility problem is equivalent to the word problem of groups—given a word of a group as a product of its generators, decide whether it is equal to the identity. The relation between the contractibility of a loop and the word problem comes from the fact that a loop of a complex is contractible iff the corresponding word of the *fundamental group* of the complex is the identity. Dehn gave an algorithm (Dehn's algorithm) for the contractibility problem when the complex is a surface; for some recent interesting results for this special case see [5]. Attempts to extend Dehn's algorithm to higher dimensional manifolds made no substantial progress, however, for the very good reason that, as Novikov [15] showed in 1955, the word problem is undecidable.

The equivalence between the contractibility problem and the word problem of groups is based on the fact that every group with a finite representation (with generators and relations) is the fundamental group of a finite simplicial complex. Since the fundamental group depends only on the 2-skeleton of a complex (the collection of all simplices of dimension 2 or less), it follows that the contractibility problem is undecidable even for two-dimensional complexes. It is also known that every group with finite representation is the fundamental group of some four-dimensional manifold [13]. Thus, the contractibility problem is undecidable even for four-dimensional manifolds.

Our main result is a reduction from the contractibility problem to the task-solvability problem. We outline the ideas behind this reduction here. The Herlihy–Shavit condition [11] states that a task is solvable iff there is a chromatic subdivision of the input complex together with a simplicial map which is consistent with the input-output relation (carrier-preserving) and preserves colors. Here we consider a class of simple 3-processor tasks that is restricted to those whose input complex consists of a single triangle (2-simplex). In addition, these tasks have the property that whenever less than three processors participate, they must output a simplex of a fixed loop L of the output complex. The Herlihy–Shavit condition implies that if the task is solvable, then L is contractible. In fact, if we drop from the Herlihy–Shavit condition the restriction that the map must be color-preserving, the opposite would be true: L is contractible if the task is solvable. The difficult part of our reduction, then, is to extend this relation to the case of chromatic complexes and color-preserving simplicial maps. To do this, we proceed in stages. We first show that the contractibility problem remains undecidable for loops of length 3 of chromatic complexes. The final reduction is to take a chromatic complex with a loop of length 3 and transform it into a 3-processor task that is solvable iff the loop is contractible.

In section 2, we discuss the solvability problem, present the Herlihy–Shavit condition, and define the special class of tasks that we consider in this paper. In section 3, we discuss the contractibility problem and strengthen the result that the contractibility problem is undecidable for the special case of chromatic complexes and loops of length 3. We give a reduction from this stronger version of the contractibility problem to the task-solvability problem in section 4. The results from section 3 and the Herlihy–Shavit condition are then used to prove that the reduction works. We conclude by discussing some of the implications of our results.

2. The task-solvability problem. We will use standard terminology from algebraic topology (see [14]). All complexes considered here are *finite* and *pure*; that is,

all maximal simplices have the same dimension (usually two-dimensional).

In topology, a simplex is defined by a set of $n+1$ points, but in the theory of distributed computation a simplex represents a consistent set of views of $n+1$ processors. The natural ordering of processors (according to their IDs) imposes structure on the complexes in that their simplices are ordered. This order defines a natural *coloring* of the vertices of the complex, where colors represent the rank of the ID of a processor. More precisely, a coloring of an n-dimensional simplicial complex is an assignment of colors $\{0, 1, \ldots, n\}$ to its vertices such that each vertex receives exactly one color and vertices of each simplex receive distinct colors. A *chromatic* simplicial complex is a simplicial complex together with a coloring.

A distributed task is a natural generalization of the notion of a (computable) function for the model of distributed computation. The computation of functions by a distributed system imposes such tight coordination of processors that only trivial functions can be computed wait-free by asynchronous distributed systems. Mainly for this reason, the study of distributed computation is focused on computing relations, a natural generalization of functions which requires less tight coordination of processors. In general, a distributed task is an input-output relation. Because in a distributed system some processors may take no steps at all, the task input-output relation must be defined on partial inputs and outputs. This requirement is captured nicely by assuming that the inputs form a chromatic simplicial complex. The vertices of a simplex of this complex denote the inputs to a subset of processors, the *participating* processors [11]. Similarly, the possible outputs of a distributed task form a chromatic simplicial complex.

DEFINITION 1. *A distributed task for $n+1$ processors is a nonempty relation T between the simplices of two n-dimensional chromatic complexes I, O, $T \subset I \times O$, which preserves colors; that is, when $(A, B) \in T$, then A and B have the same colors (and therefore the same dimension).*

A distributed task is solvable when there is a distributed protocol such that the input to processor with ID k is a vertex of I with color k, its output is a vertex of O of color k, the set of the input vertices form a simplex $A \in I$, and the set of output vertices form a simplex $B \in O$ with $(A, B) \in T$. In other words, the participating processors get vertices of an input simplex and output vertices of a simplex of the output complex such that the input simplex and the output simplex form a pair of the task input-output relation. Each processor knows only its vertex, not the whole input simplex. Finding out the input simplex is usually impossible because that task is equivalent to the consensus problem, which is not solvable. Of course, the notion of solvability depends on the computational model. Here, we consider the standard computational model of wait-free protocols for shared read-write memory. In a wait-free protocol, a processor must produce a valid output even when all other processors fail.

A typical distributed task is shown in Figure 1. The input complex I contains only one triangle $\{a, b, c\}$, and the output complex O is a subdivided triangle. The numbers on the vertices are colors. The input-output relation $T \subset I \times O$ contains the tuples $(\{a, b, c\}\{x, y, z\})$ for all triangles $\{x, y, z\}$ of O (there are seven such triangles); it also contains all possible color-preserving tuples of simplices of the boundaries of I and O.

A problem central to the theory of distributed computation is the characterization of the set of solvable tasks. This problem has a trivial negative answer: whether a 1-processor task is solvable is equivalent to whether the task, that is, the input-

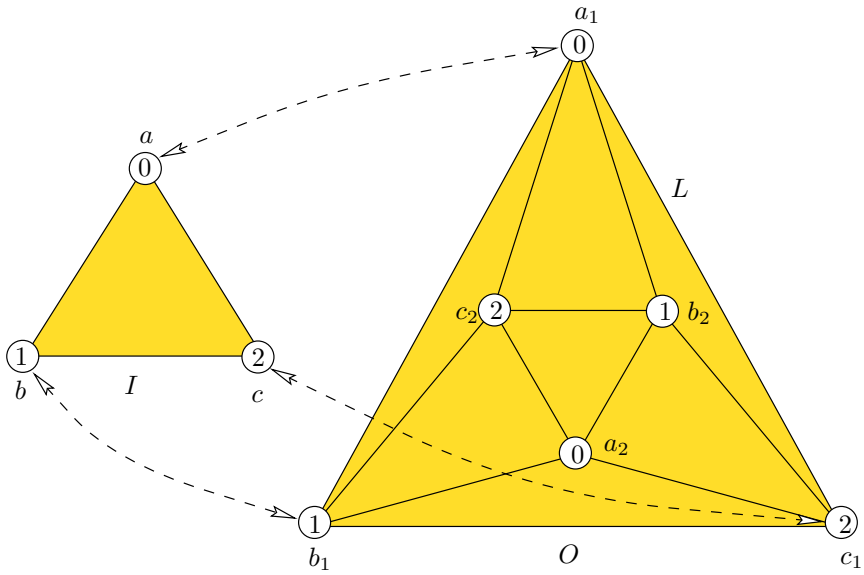


FIG. 1. *A standard inputless task.*

output relation, is recursive (computable); a similar observation was made in [12]. However, this is an unsatisfactory answer because it sheds no light on the difficulties inherent in distributed computation. Furthermore, many interesting distributed tasks are straightforward input-output relations. The interesting question, then, is whether a characterization of "simple" tasks exists. Here, we show that the answer for three or more processors remains negative, even for the simplest kind of tasks—finite tasks with a trivial input complex. For less than three processors, it is known that there exists a simple characterization for finite tasks of two processors that reduces the task-solvability problem to the connectivity properties of the output complex [1].

Our proof uses the Herlihy–Shavit condition for task solvability. Roughly speaking, this condition entails that a subcomplex of the output complex is "similar to" the input complex. To state the condition precisely, we need a few definitions: Consider a chromatic complex C and a subdivision C' of C (a subdivision of a complex is a refinement of it; see, for example, [14, p. 84]). For a simplex $A \in C'$, its carrier, $\text{carrier}(A)$, is the smallest simplex of C that contains A. The complex C' is a *chromatic subdivision* of C if it is chromatic and its coloring has the property that each vertex $u \in C'$ has the color of some vertex of $\text{carrier}(u)$.

PROPOSITION 1 (Herlihy–Shavit). *A task $T \subset I \times O$ is solvable wait-free iff there exists a subdivision I' of I and a color-preserving simplicial map $\mu : I' \mapsto O$ such that for each simplex $A \in I'$ there exists a simplex $B \in O$ with $\mu(A) \subset B$ and* $(\text{carrier}(A), B) \in T$.

A map μ that satisfies the above condition will be called *carrier-preserving* and *color-preserving*.

Proposition 1 provides a powerful tool for checking whether a particular task is solvable. For example, by applying the Herlihy–Shavit condition we can conclude immediately that the task of Figure 1 is wait-free solvable. To see this, notice that in this case we can take the subdivision I' to be the output complex and the map μ

to be the identity map.[1] If, however, we create a "hole" in the output complex by removing the triangle $\{a_2, b_2, c_2\}$, the resulting task is not solvable; intuitively, the map μ cannot create a "torn" image of I.

The main objective of our paper is to show that the condition of Proposition 1 is not constructive, namely, that there is no effective way to find I' from T; computing μ is easy, since one can try all possible simplicial maps from I' to O. We will restrict our attention to the simple case of tasks of three processors, $n = 2$. In this case, the simplices are triangles and the simplicial complexes are of dimension 2. For this dimension, our intuition about topological facts is usually correct; exactly the opposite is true for higher dimensions. We introduce one further simplification: we will deal only with tasks where the input complex consists of only one triangle. Furthermore, for each proper face of the input triangle there is exactly one possible output. In particular, there is a loop L of the output complex that has length 3 such that when less than three processors participate in the execution, the processors must output a simplex of L, and this simplex is unique because of the coloring requirements. When all three processors participate, the output can be any simplex of the output. We will call such a task a *standard inputless task* (O, L). The task of Figure 1 is an example of such a task. Since the input to each processor is fixed, we interpret a standard inputless task as follows: processors do not really get any input; rather, they simply execute a protocol in order to "agree on" some triangle of the output complex O. This could be trivially achieved (by agreeing on a triangle in advance), except for the difficulty that when some processors do not participate, the output simplex must belong to the loop L.

For a standard inputless task, the Herlihy–Shavit condition can be restated as "the task is solvable iff there is a chromatic subdivision I' of a triangle I and a color-preserving simplicial map μ that maps the boundary of I' to the loop L and that can be extended over I'." The coloring restrictions imply that the simplicial map μ maps the boundary of I' only once around L. Putting it differently, the requirement that μ is color-preserving guarantees that it is also carrier-preserving.

If we disregard colors for the moment, a standard inputless task is solvable iff there is a carrier-preserving simplicial map μ from the boundary of a subdivided triangle I' to the loop L which can be extended over the whole triangle. This condition shows the close connection between task solvability and the contractibility problem, because such I' and μ exist iff the loop L of the output complex O is contractible (we will elaborate on this connection in section 4). It is not, however, immediate that this observation holds for the special case of chromatic complexes and color-preserving simplicial maps. Here, we extend this connection to the chromatic case by a series of reductions.

3. The contractibility problem. Let X be a topological space. A *loop* L of X is a continuous map from the 1-sphere $S = \{x \in \mathbf{R}^2 : |x| = 1\}$ to X. Two loops L and L' are *homotopic* when L can be continuously deformed to L'. More precisely, L and L' are said to be homotopic if there exists a continuous map $F : S \times [0, 1] \mapsto X$, such that $F(x, 0) = L(x)$ and $F(x, 1) = L'(x)$ [14, p. 103]. A *nonsingular* loop is one without self-intersections (when the map is an injection). A loop L is *null-homotopic*, or *contractible*, when it is homotopic to a constant loop; the image of a constant loop is a point. Equivalently, loop L is null-homotopic when it can be continuously deformed to a point [18, p. 158]. For example, in Figure 2 the loop L_1 is null-homotopic while

[1] Strictly speaking, I' is combinatorially homeomorphic to O, and μ is this homeomorphism.

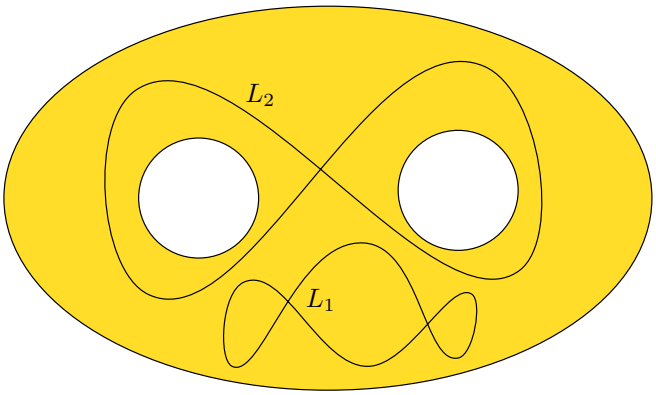


FIG. 2. *Contractible (L_1) and noncontractible (L_2) loops.*

the loop L_2 is not.

Let C be a simplicial complex, that is, a collection of simplices in the Euclidean space R^m. The polytope $|C|$ of C is the underlying Euclidean space consisting of the union of the simplices of C. A loop of a complex C is a simplicial loop of its polytope $|C|$. Thus, the image of a loop is a sequence of edges (v_1, v_2), $(v_2, v_3), \ldots, (v_{k-1}, v_k)$, (v_k, v_1) (the image of a null-homotopic loop is simply a vertex v). We usually do not distinguish between the loop of a complex and its image (as we have done many times so far); so, for example, we can refer to a simplex of the loop when we really mean a simplex of the image of it.

To show that task solvability is undecidable, we will use the standard technique of reducing a known undecidable problem to it. In our case, this problem is the contractibility problem [19].

DEFINITION 2. *The contractibility problem is defined as follows: given a simplicial complex C and a loop L of C, is L null-homotopic?*

For this definition to be complete, we need to fix the representation of C and L. Since we are interested only in whether the problem is decidable, the details of the representation are not important. For our purposes here, we assume that C and L are given explicitly by their simplices.

There is an important connection between the homotopic properties of loops and group theory, through the fundamental group of a complex. In particular, a loop is null-homotopic iff the corresponding word of the fundamental group is equal to the identity. This connection between contractibility and group theory results in the following proposition.

PROPOSITION 2. *The contractibility problem is undecidable for two-dimensional complexes.*

This folklore result is based on the fact that for every group G with a finite representation with generators and relations, there exists a finite simplicial complex with fundamental group G. This complex can be easily constructed from G (see, for example, [19, p. 129]). In fact, something stronger holds: each group G is the fundamental group of a four-dimensional simplicial manifold [13, pp. 143–144]. This means that the contractibility problem is undecidable for four-dimensional manifolds. In contrast, for two-dimensional manifolds (e.g., sphere, torus, projective plane) it is decidable [4]. Some recent work on this special case has led to a linear-time algorithm for almost all two-dimensional manifolds [5]. The contractibility problem for three-

dimensional manifolds is, to our knowledge, still unresolved; however, it is known that not every group with finite representation can be the fundamental group of a three-dimensional manifold.

Notice also that Proposition 2 refers to two-dimensional complexes. This is based on the fact that the fundamental group of a complex of any dimension is identical to the fundamental group of its 2-skeleton.

Since every group can be the fundamental group of a complex, the contractibility problem is equivalent to the word problem of groups. The word problem asks whether a word of a group (as a product of its generators) is equal to the identity [19, p. 46]. Novikov [15] showed that the word problem is undecidable: there exists a group G such that no algorithm can decide whether a word of this group is equal to the identity (for a textbook proof see [16, Chapter 12]). Notice that the group G need not be part of the input, although for our purposes the weaker version of the result when the group is part of the input will suffice.

We will make use of a stronger version of Proposition 2. We first observe that the contractibility problem is undecidable for *link-connected* two-dimensional complexes. A simplicial complex is link-connected when the link of every vertex is connected (the link of a vertex is the subcomplex induced by its adjacent vertices). To see that the contractibility problem remains undecidable for link-connected complexes, notice that it is undecidable for the 2-skeleton of 4-manifolds, and clearly these complexes are link-connected. Therefore we have the stronger proposition.

PROPOSITION 3. *The contractibility problem is undecidable for link-connected two-dimensional complexes.*

Link-connectivity must be preserved by all our reductions, but we will not use it until the last part (Lemma 3) of the proof of the main result.

The plan for reducing this undecidable problem to the task-solvability problem is as follows: First, we strengthen Proposition 3 to chromatic complexes and loops of length 3. A chromatic complex together with a loop of length 3 defines a standard inputless task. Using the Herlihy–Shavit condition, we then show that this task is solvable iff the loop is contractible.

We begin by showing that Proposition 3 holds for nonsingular loops (i.e., loops without self-intersections).

LEMMA 1. *The contractibility problem is undecidable for nonsingular loops of link-connected two-dimensional complexes.*

Proof. Given a link-connected two-dimensional simplicial complex C and a loop L of C, we create a new complex C' and a singular loop L' of C' such that L is null-homotopic iff L' is null-homotopic. The idea is that C' can be produced by attaching an annulus (ring) A to C: one boundary of A is identified with the loop L, and the other boundary is a nonsingular loop L' (see Figure 3). The annulus A is free of self-intersections except for points of L.

We claim that L is contractible in C iff L' is contractible in C'. But first we need a definition. A topological space Y is a *deformation retract* of a topological space X, $Y \subset X$, iff there is a continuous map $f : X \times [0,1] \mapsto X$ such that for all $x \in X$, $f(x,0) = x$ and $f(x,1) \in Y$, and for all $y \in Y$ and all t, $f(y,t) = y$.

If Y is a deformation retract of X, then Y and X have the same homotopy type [14, p. 108]. It is clear that $|C|$ is a deformation retract of $|C'|$: f gradually collapses the annulus $|A|$ to the loop $|L|$ keeping $|C|$ fixed. It follows that $|C|$ and $|C'|$ have the same homotopy and therefore that L is contractible in C iff it is also contractible in C'. The claim follows from the fact that L and L' are homotopic in C'.

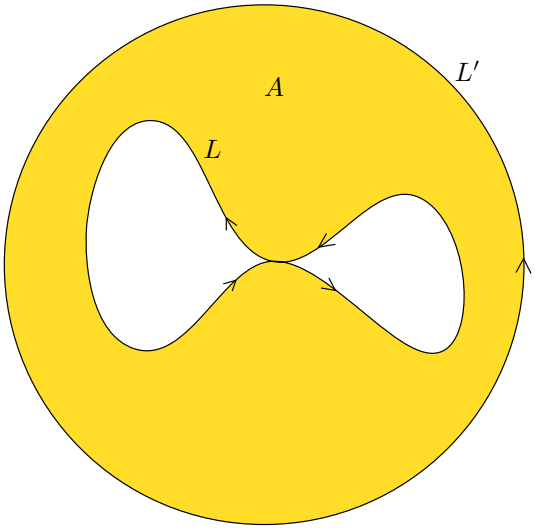


FIG. 3. *Reduction to nonsingular loop.*

A minor issue is that the annulus A must be constructed explicitly. We give here one such construction. Let $(x_0, x_1), (x_1, x_2), \ldots, (x_{k-1}, x_k), (x_k, x_0)$ be the edges of L (some of them may be identical when part of the loop retraces itself). The boundary of annulus A identified with L contains vertices $y_0, y_1, \ldots, y_k$ such that y_i will be identified with x_i. The other boundary, L', of annulus A contains distinct vertices $z_0, z_1, \ldots, z_k$. The triangles of annulus A are $\{y_i, y_{i+1}, z_i\}$ and $\{y_{i+1}, z_i, z_{i+1}\}$, for $i = 0, 1, \ldots, k$. We have to verify that these are indeed triangles (i.e., all vertices are distinct) and that annulus A is free of self-intersections except for points in L. Some of the vertices x_i of L may be identical, because the loop L may cross or even retrace itself. However, since (x_i, x_{i+1}) is an edge of L, it follows that y_i and y_{i+1} are distinct and therefore that the given triangulation of annulus A is valid. It is also easy to verify that annulus A has no self-intersections outside L.

Finally, we have to verify that the new complex C' is link-connected. It is clear that the links of vertices not in L are connected. Consider now the link $\mathrm{lk}(x_i)$ of a vertex $x_i \in L$. Since C is link-connected, every vertex of $C \cap \mathrm{lk}(x_i)$ is connected through $\mathrm{lk}(x_i)$ to x_{i-1} and to x_{i+1}. In particular, x_{i-1} and x_{i+1} are connected through $\mathrm{lk}(x_i)$ (or they are identical). Similarly, every vertex in $A \cap \mathrm{lk}(x_i)$ is connected to x_{i-1} or to x_{i+1}. Therefore, $\mathrm{lk}(x_i)$ is connected. □

This lemma allow us to consider only nonsingular loops. We may sometimes treat a nonsingular loop L of a complex C as the one-dimensional subcomplex of C consisting of the edges of L. We are now ready to strengthen Proposition 3 to chromatic complexes and loops of length 3.

THEOREM 1. *The contractibility problem is undecidable for loops of length* 3 *of link-connected two-dimensional chromatic complexes.*

Proof. Consider a link-connected two-dimensional simplicial complex C and a nonsingular loop L of it. We will show how to produce a chromatic complex C' and a loop $L' \subset C'$ of length 3.

Producing a chromatic complex is easy. Let bsd C denote the *barycentric subdivision* of the simplicial complex C [14, p. 85]. We can color the simplicial complex

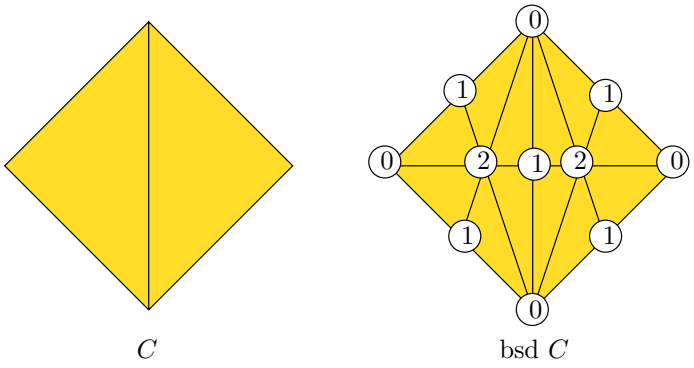


FIG. 4. *The chromatic barycentric subdivision.*

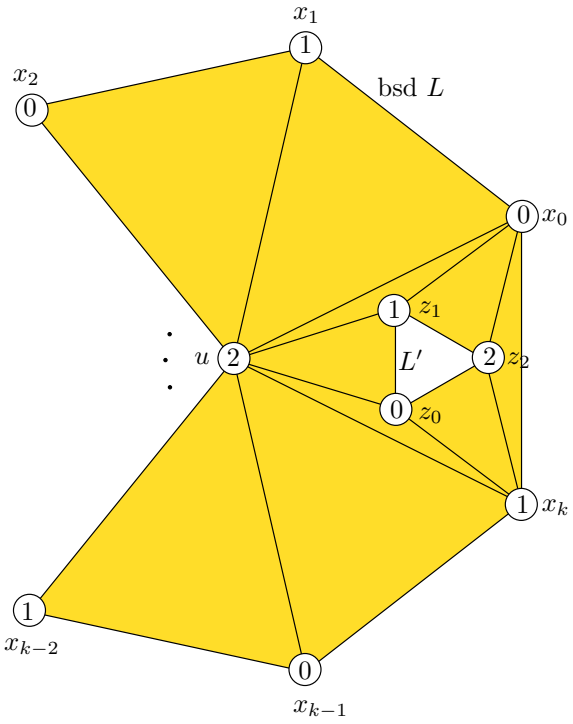


FIG. 5. *Reduction to loops of length* 3.

bsd C with three colors as shown in Figure 4. Original vertices of C are colored with 0, vertices on the edges—with carrier an edge—with 1, and the remaining vertices—with carrier a triangle—with color 2. With this coloring, bsd C becomes a chromatic complex. The nonsingular loop L of C corresponds to a nonsingular loop bsd L of the chromatic complex bsd C. Clearly, L is null-homotopic in C iff bsd L is null-homotopic in bsd C'. Notice also that the vertices of bsd L have colors 0 or 1.

Finally, to produce a complex C' and a loop L' of length 3, we employ the reduction of Lemma 1: C' is the result of attaching a chromatic annulus A to the nonsingular loop bsd L. Let (x_0, x_1), $(x_1, x_2), \ldots, (x_{k-1}, x_k)$, (x_k, x_0) be the edges

of bsd L. One boundary of the chromatic annulus A is identified with bsd L, while the other boundary L' contains three vertices, z_0, z_1, z_2, with colors 0, 1, and 2, respectively. There is also an internal vertex u of A with color 2. The chromatic annulus A is shown in Figure 5 (again, numbers on vertices indicate colors). We omit its precise description here since the reader can easily derive it from the figure. Remaining to be verified is that A is an annulus without self-intersections, and this follows directly from the fact that bsd L is nonsingular. Note that this is the only place where we need Lemma 1. We could actually use a simpler construction by letting L' be the loop (x_0, u, x_k), but the construction of Figure 5 is consistent with the proof of Lemma 1.

An argument identical with that of the proof of Lemma 1 establishes that the complex C' is link-connected and that L' is null-homotopic iff L is null-homotopic. The theorem follows from Lemma 1. □

The requirement that loop L has length 3 is a "technical" detail. We could prove our main result by simply considering loops L where, instead of an edge (z_i, z_{i+1}), there is a chromatic path between z_i and z_{i+1}. The restriction to loops of length 3 results in simpler constructions and proofs, however.

4. Reduction to task-solvability. To show that task-solvability for three processors is undecidable, we will reduce the stronger version of the contractibility problem of Theorem 1 to the task-solvability problem. The reduction is straightforward. Given a link-connected two-dimensional chromatic complex C and a loop L of length 3, the output is the standard inputless task $T = (C, L)$. We will show that the loop L is contractible in C iff the standard inputless task (C, L) is solvable.

The proof is based on the two following lemmas.

LEMMA 2. *Let $T = (C, L)$ be a standard inputless task. Loop L is contractible in C iff there is a subdivision I' of the input triangle I and a simplicial map $\psi : I' \mapsto C$ that is carrier-preserving.*

Proof. Notice first that we require neither that I' be a chromatic subdivision nor that μ be color-preserving. It follows directly from the definition of null-homotopic loops that loop L is contractible in C iff there is a *continuous map* ϕ from a disk B to $|C|$ that maps *homeomorphically* the boundary of the disk to $|L|$. Since the triangle I is homeomorphic to a disk, L is contractible in C iff there is a continuous map from I to $|C|$ that maps its boundary to the loop $|L|$ homeomorphically (and, therefore, simplicially). The problem with this definition is that ϕ is a continuous map, not a simplicial one. However, a fundamental result from algebraic topology, the simplicial approximation theorem [14, p. 89], allows us to replace the continuous map ϕ with a simplicial one. By the simplicial approximation theorem, there is a subdivision I' of the triangle I and a simplicial map $\psi : I' \mapsto C$ that *approximates* ϕ. It suffices, therefore, to verify that ψ is also carrier-preserving. By the definition of simplicial approximations, for each point x of I, $\psi(x)$ is a vertex of the smallest simplex of C that contains $\phi(x)$. Since ϕ maps simplicially the boundary of I to $|L|$, all points of an edge E of I are mapped to the same edge $\phi(E)$ of L. Thus, the vertices of I' with carrier E are mapped by ψ to vertices of $\phi(E)$, which shows that ψ is carrier-preserving. □

Lemma 2 shows the close connection between the contractibility of loops and the solvability of tasks. However, it requires only that the map ψ be carrier-preserving, while the Herlihy–Shavit condition requires the map to be chromatic too. The following lemma shows that this is not a problem.

LEMMA 3. *Let $T = (C, L)$ be a standard inputless task, where C is link-connected. If there exists a subdivision I' of the input triangle I and a carrier-preserving simplicial*

map $\psi : I' \mapsto C$, then there exists a chromatic subdivision A of I and a simplicial map $\mu : A \mapsto C$ that is both carrier-preserving and color-preserving.

Proof. The proof here is an adaptation of the proof of a similar result in [11, Lemma 5.21]. The basic idea is that the colors of C induce a coloring of I'. A vertex $u \in I'$ is assigned the color of its image $\psi(u) \in C$. We call such a coloring of I' ψ-induced. This coloring makes ψ a color-preserving map. However, such a coloring may not make I' a chromatic complex, because two adjacent vertices u_1 and u_2 of I' may receive the same color. Because ψ is a simplicial map, this happens only when these vertices are mapped to the same node, in which case we say that the edge $\{u_1, u_2\}$ is monochromatic. Similarly, we say that a triangle is monochromatic when all its vertices are mapped to the same vertex.

Let A be a subdivision of I such that there is a carrier-preserving simplicial map $\mu : A \mapsto C$ such that the number of monochromatic simplices of A with the μ-induced coloring is minimum. We claim that A has no monochromatic edges or triangles. Suppose that this is not the case. We will reach a contradiction by exhibiting a subdivision A' of A—and therefore of I—with one monochromatic simplex less than A.

Consider first the case when A with the μ-induced coloring has a monochromatic triangle $\{u_1, u_2, u_3\}$, and therefore $\mu(u_1) = \mu(u_2) = \mu(u_3)$. Let p be a vertex in the link of $\mu(u_1)$; such a vertex always exists because the complex C is pure. Consider now the subdivision A' of A where the triangle $\{u_1, u_2, u_3\}$ is subdivided into three triangles $\{u_1, u_2, c\}$, $\{u_1, c, u_3\}$, and $\{c, u_2, u_3\}$, where c is the barycenter of $\{u_1, u_2, u_3\}$. Consider also the map $\mu' : A' \mapsto C$ that agrees with μ on the vertices of A and $\mu'(c) = p$. But then A' with the μ'-induced coloring has one monochromatic simplex (the triangle $\{u_1, u_2, u_3\}$) less than A with the μ-induced coloring.

We now assume that no triangle of A is monochromatic but that there is a monochromatic edge $\{u_1, u_2\}$ on the boundary of A. Then, u_1 and u_2 belong to exactly one triangle of A. Let b be the third vertex of this triangle. We can construct a subdivision A' of A by subdividing the triangle $\{u_1, u_2, b\}$ into two triangles $\{u_1, c, b\}$ and $\{c, u_2, b\}$, where c is the barycenter of $\{u_1, u_2\}$. Consider the extension $\mu' : A' \mapsto C$ such that μ' agrees with μ on all vertices of A and $\{\mu'(c), \mu'(u_1)\}$ is an edge of the loop L. Because μ' is carrier-preserving, we have again reached a contradiction since A with the μ-induced coloring has one monochromatic simplex more than A' with the μ'-induced coloring.

The last, and more complicated, case to consider is when A has a monochromatic edge $\{u_1, u_2\}$ that is not in its boundary. This is the only place where we must require that complex C be link-connected. The edge $\{u_1, u_2\}$ belongs to exactly two triangles. Let a and b be the remaining vertices of these two triangles. Since μ is a simplicial map, $\mu(a)$ belongs to the link of $\mu(u_1)$ in C. Let p be a vertex in the link of $\mu(u_1)$ (not necessarily distinct for $\mu(a)$). But then the fact that C is link-connected implies that there is a path with edges (p_1, p_2), $(p_2, p_3), \ldots, (p_{k-1}, p_k)$ in the link of $\mu(u_1)$ that connects $\mu(a) = p_1$ and $p = p_k$. We can always choose a nonempty path, even when $p = a$, because C is pure. Similarly, there is a path with edges (q_1, q_2), $(q_2, q_3), \ldots, (q_{l-1}, q_l)$ which connects $\mu(b)$ and $p = q_l$. This suggests the following subdivision A' of A: The triangle $\{u_1, u_2, a\}$ is subdivided into triangles $\{\bar{p}_i, \bar{p}_{i+1}, u_1\}$ and $\{\bar{p}_i, \bar{p}_{i+1}, u_2\}$, $i = 1, 2, \ldots, k-1$. The vertices $\bar{p}_i$, $i = 2, 3, \ldots, k$, are new and distinct, and $\bar{p}_k$ is the barycenter of $\{u_1, u_2\}$. Similarly, the triangle $\{u_1, u_2, b\}$ is subdivided into triangles $\{\bar{q}_i, \bar{q}_{i+1}, u_1\}$ and $\{\bar{q}_i, \bar{q}_{i+1}, u_2\}$, $i = 1, 2, \ldots, l-1$, where $\bar{q}_l = \bar{p}_k$. Consider also the extension $\mu' : A' \mapsto C$ such that μ' agrees with μ on

A and $\mu'(\bar{p}_i) = p_i$, $i = 1, 2, \ldots, k$ and $\mu'(\bar{q}_i) = q_i$, $i = 1, 2, \ldots, l$. Using the fact that C is chromatic, it is easy to verify that B with the μ'-induced coloring has one monochromatic simplex less than A with the μ-induced coloring. □

An alternative proof of Lemma 3 can be obtained by employing the convergence algorithm of Borowsky and Gafni [2]. We outline this proof here. By the Herlihy–Shavit condition, it suffices to show that the task T is solvable. The protocol consists of two phases. In the first phase, processors "converge" on a simplex of I'. Let x_i be the vertex of I' where processor i converges. If the color of $\mu(x_i)$ is i, then the processor i stops and outputs $\mu(x_i)$. Obviously, at least one processor stops in this phase. Although the remaining processors do not know the output of the stopped processors, they know a simplex of C that contains the outputs of stopped processors. In the second phase, the remaining processors converge in C in the link of the output of all stopped processors; each of the remaining processors starts at a vertex of its color and, if possible, a vertex of the loop L. Since C is link-connected and chromatic, the remaining processors can indeed converge. Thus T is solvable.

We can now prove the main theorem of this paper.

THEOREM 2. *The task-solvability problem for three or more processors in the read-write wait-free model is undecidable.*

Proof. By Theorem 1, there is no algorithm to decide, given a standard inputless task $T = (C, L)$, whether the loop L is contractible in C when C is link-connected. However, by Lemmas 2 and 3 the loop L is contractible iff there is a chromatic subdivision I' of the input triangle I and a color-preserving and carrier-preserving simplicial map $\mu : I' \mapsto C$. This is precisely the Herlihy–Shavit condition for T to be solvable, and therefore L is contractible in C iff T is solvable. Hence, task solvability is undecidable for three processors.

This immediately implies that the solvability problem for more than three processors is also undecidable: Consider, for example, tasks where there is only one possible output of all but the first three processors; such a task is solvable iff the subtask for the first three processors is solvable. □

Biran, Moran, and Zaks [1] define a slightly different model of distributed computation in which the processors must produce a valid output only if all of them complete their protocol. For this model, the input-output relation contains only n-dimensional simplices. For each task $T \subset I \times O$, it is easy to construct an equivalent task T' in the model of [1]: The input for task T' to processor i may be a special value p_i that indicates that the processor does not "participate" in T. In that case, the processor must output a special value q_i. Otherwise, the input is a vertex of I and the output a vertex of O in such a way that the input-output relation of processors whose inputs are not special values is identical to T. It is easy to see that T is solvable iff T' is solvable in the model of [1]. This immediately implies the following.

COROLLARY 1. *The task-solvability problem for the model of Biran, Moran, and Zaks* [1] *is undecidable for three or more processors.*

Another interesting variant of the shared read-write memory model is the comparison-based model where processors cannot access directly their IDs but can only compare them [11]. A typical task for this model is the *renaming* task: the input (name) to each processor is a distinct member of a set S of size m, and the output must be a distinct member of a smaller set of size k. In the comparison-based model, the input to a processor is not a vertex of the input complex I but instead is some value associated with the vertex. Different vertices may have the same value. Similarly there are values associated with the vertices of the output complex. This generalization in

the definition of tasks is necessary for the comparison-based model to be different from the model we have considered so far; otherwise, when a processor gets as input a vertex of the input complex I, it can immediately determine its color and the rank of its ID. This suggests a trivial reduction from task-solvability to the comparison-based model task-solvability: Given a task $T \subset I \times O$, construct a comparison-based model task T' with the same input-output tuples where the value of each vertex is the vertex itself. Then all values are distinct, and a processor can infer its color from its input. It follows that T is solvable iff T' is solvable in the comparison-based model.

COROLLARY 2. *The task-solvability problem for three or more processors in the comparison-based model is undecidable.*

Recently, Herlihy and Rajsbaum [8] proposed an interesting extension of Theorem 2 to the models of resiliency and set-consensus. Using the contractibility problem, they showed that the task-solvability problem for these models is also undecidable in general.

5. Conclusion. Let us define the size of a task to be the number of its input-output tuples. Theorem 2 implies that for *any recursive function* $f(s)$, there are solvable tasks of size s whose protocols require at least $f(s)$ steps. This is indicative of the difficulty involved in developing a robust complexity theory for asynchronous distributed computation. The analogy for traditional complexity theory would be that the finite languages, a proper subset of regular languages, are nonrecursive! However, it may still be possible to develop a notion of complexity of distributed tasks that is independent of the task size. An intriguing open problem is finding a solvable "natural" task whose protocol requires, for example, exponential number of steps. Of course, one could use the reductions given in this paper to produce such a task, but that task could not be considered natural.

The Herlihy–Shavit condition (despite the title of [10]) is not constructive. Our results here cast some doubt on its applicability as a necessary and sufficient condition for task solvability. On the one hand, the best way to show that a task is solvable is to provide a distributed algorithm that solves the given task. On the other hand, showing that a task is not solvable often employs other weaker conditions that are easier to apply than the Herlihy–Shavit condition (e.g., Sperner's lemma or homology). However, our work does show how powerful the Herlihy–Shavit condition is, because no weaker condition would enable us to derive the results of this paper. Ironically, although our work exposes the inherent weakness of the Herlihy–Shavit condition, to our knowledge our work is the only work that makes full use of it.

Acknowledgments. The possibility that the Herlihy–Shavit condition might not be constructive was suggested by Shlomo Moran in a conversation with the first author in 1994. We would like to thank Geoffrey Mess of the UCLA Mathematics Department for providing useful pointers to the literature.

REFERENCES

[1] O. BIRAN, S. MORAN, AND S. ZAKS, *A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor*, in Proc. 7th ACM Symposium on Principles of Distributed Computing, Toronto, Ont., Canada, ACM, New York, 1988, pp. 263–275.

[2] E. BOROWSKY, *Capturing the Power of Resiliency and Set Consensus in Distributed Systems*, Ph.D. thesis, University of California, Los Angeles, 1995.

[3] E. Borowsky and E. Gafni, *Generalized FLP impossibility result for t-resilient asynchronous computations*, in Proc. 25th ACM Symposium on the Theory of Computing, San Diego, CA, ACM, New York, 1993, pp. 91–100.

[4] M. Dehn, *Über die topologie des dreidimensional raumes*, Math. Ann., 69 (1910), pp. 137–168 (in German); *Papers on Group Theory and Topology*, Springer-Verlag, New York, 1987 (in English).

[5] T. K. Dey and S. Guha, *Optimal algorithms for curves on surfaces*, in Proc. 36th Annual IEEE Symposium on Foundations of Computer Science, Milwaukee, WI, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 266–274.

[6] M. Fischer, N. Lynch, and M. Paterson, *Impossibility of distributed consensus with one faulty process*, J. Assoc. Comput. Mach., 32 (1985), pp. 374–382.

[7] E. Gafni and E. Koutsoupias, 3-*processor tasks are undecidable*, in Proc. 14th Annual ACM Symposium on Principles of Distributed Computing, Ottawa, Ont., Canada, ACM, New York, 1995, p. 271 (abstract).

[8] M. Herlihy and S. Rajsbaum, *The decidability of distributed decision tasks*, in Proc. 29th ACM Symposium on the Theory of Computing, El Paso, TX, ACM, New York, 1997, pp. 589–598.

[9] M. Herlihy and N. Shavit, *The asynchronous computability theorem for t-resilient tasks*, in Proc. 25th ACM Symposium on the Theory of Computing, San Diego, CA, ACM, New York, 1993, pp. 111–120.

[10] M. Herlihy and N. Shavit, *A simple constructive computability theorem for wait-free computation*, in Proc. 26th ACM Symposium on the Theory of Computing, Montreal, Quebec, Canada, ACM, New York, 1994, pp. 101–110.

[11] M. Herlihy and N. Shavit, *The Topological Structure of Asynchronous Computability*, Tech. report CS-96-03, Department of Computer Science, Brown University, Providence, RI, 1996.

[12] P. Jayanti and S. Toueg, *Some results on the impossibility, universality and decidability of consensus*, in Proc. 6th Workshop on Distributed Algorithms, Haifa, Israel, 1992, pp. 68–84.

[13] W. S. Massey, *Algebraic Topology: An Introduction*, Harcourt, Brace and World, New York, 1967.

[14] J. R. Munkres, *Elements of Algebraic Topology*, Addison-Wesley, Reading, MA, 1984.

[15] P. S. Novikov, *On the Algorithmic Unsolvability of the Word Problem in Group Theory*, Trudy Mat. Inst. im. Steklov. 44, Izdat. Akad. Nauk SSSR, Moscow, 1955, (in Russian). English translation available as *On the algorithmic insolvability of the word problem in group theory,* AMS Trans., 2 (1958), pp. 1–122.

[16] J. J. Rotman, *An Introduction to the Theory of Groups*, 4th ed., Grad. Texts in Math., Springer-Verlag, New York, 1995.

[17] M. Saks and F. Zaharoglou, *Wait-free k-set agreement is impossible: The topology of public knowledge*, in Proc. 26th ACM Symposium on the Theory of Computing, San Diego, CA, ACM, New York, 1993, pp. 101–110.

[18] H. Seifert and W. Threlfall, *A Textbook in Topology*, Academic Press, New York, 1980.

[19] J. Stillwell, *Classical Topology and Combinatorial Group Theory*, 2nd ed., Grad. Texts in Math., Springer-Verlag, New York, 1993.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 984–1003

SIMPLE ALGORITHMS FOR ROUTING ON BUTTERFLY NETWORKS WITH BOUNDED QUEUES*

BRUCE M. MAGGS† AND RAMESH K. SITARAMAN‡

Abstract. This paper examines several simple algorithms for routing packets on butterfly networks with bounded queues. We show that for any greedy queuing protocol, a routing problem in which each of the N inputs sends a packet to a randomly chosen output requires $O(\log N)$ steps, with high probability, provided that the queue size is a sufficiently large, but fixed, constant. We also show that for any deterministic nonpredictive queuing protocol, there exists a permutation that requires $\Omega(N/q \log N)$ time to route, where q is the maximum queue size. We present a new algorithm for routing $\log N$ packets from each input to randomly chosen outputs on a butterfly with bounded-size queues in $O(\log N)$ steps, with high probability. The algorithm is simpler than the previous algorithms of Ranade and Pippenger because it does not use ghost messages, it does not compare the ranks or destinations of packets as they pass through switches, and it cannot deadlock. Finally, using Valiant's idea of random intermediate destinations, we generalize a result of Koch's by showing that if each wire can support q messages, then for any permutation, the expected number of messages that succeed in locking down paths from their origins to their destinations in back-to-back butterflies is $\Omega(N/(\log N)^{1/q})$. The analysis also applies to store-and-forward algorithms that drop packets if they attempt to enter full queues.

Key words. interconnection networks, parallel computers, communication protocols, routing algorithms, circuit-switching, performance analysis

AMS subject classifications. 68Q20, 68Q22, 68Q25, 68M07, 68M20

PII. S0097539796311818

1. Introduction.

Many commercial and experimental parallel computers, including the NYU Ultracomputer [9], the IBM RP3 [19], the BBN Butterfly [5], and NEC's Cenju [18], use butterfly networks to route packets between processors. Several proposed designs for the switching fabric of scalable high-speed ATM networks use butterfly and other closely related multistage interconnection networks [26]. Although many routing algorithms with provably good performance have been devised for butterfly networks [2, 15, 20, 23, 24, 31, 32, 33, 34], simpler algorithms are often used in practice. Typically, packets are sent along shortest paths through the network, and simple queuing protocols such as first-in first-out (FIFO) are used to determine which packets to transmit at each step. In addition, the queues at the switches can usually hold only a small number of packets. The performance of these simple algorithms has proven surprisingly difficult to analyze. For example, the only previously known upper bound on the expected time required for each input of an N-input butterfly

*Received by the editors November 5, 1996; accepted for publication May 23, 1997; published electronically January 29, 1999. A preliminary version of this paper appeared in the *Proceedings of the* 24th *Annual ACM Symposium on Theory of Computing* (*STOC*), New York, 1992, pp. 150–161. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of AFMC, ARPA, CMU, UMASS, or the U.S. Government.

http://www.siam.org/journals/sicomp/28-3/31181.html

†School of Computer Science, Carnegie Mellon University Pittsburgh, PA 15213 (bmm@cs.cmu.edu). The work of this author was supported in part by the Air Force Materiel Command (AFMC) and ARPA under contract F196828-93-C-0193, by ARPA contracts F33615-93-1-1330 and N00014-95-1-1246, and by NSF National Young Investigator Award CCR-94-57766, with matching funds provided by NEC Research Institute.

‡Department of Computer Science, University of Massachusetts, Amherst, MA 01003 (ramesh@cs.umass.edu). The work of this author was supported in part by NSF Research Initiation Award CCR-94-10077 and by NSF CAREER Award CCR-97-03017.

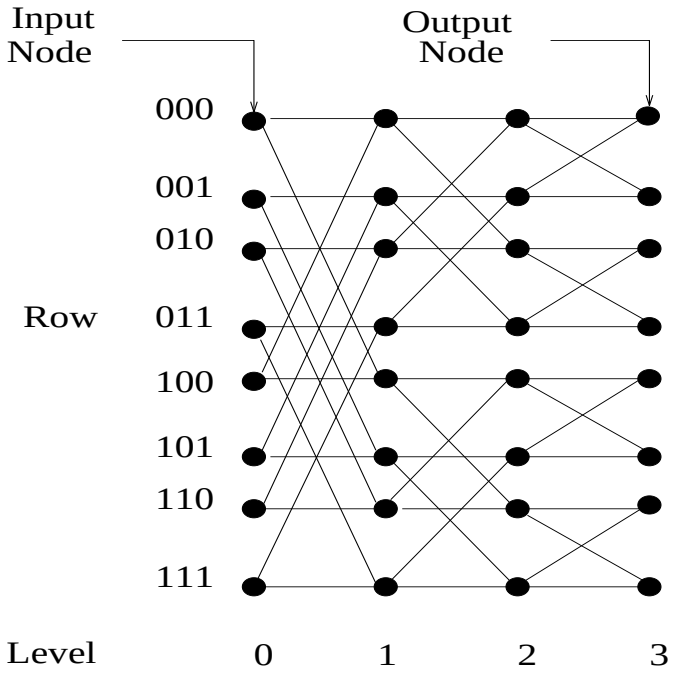


FIG. 1.1. *An 8-input butterfly network.*

network with constant-size FIFO queues to route a packet to a random destination was $O(N)$. In this paper, we show that the routing time is actually $O(\log N)$, with high probability. We also analyze the performance of several other simple algorithms for routing on butterflies with bounded queues.

1.1. Butterfly networks. An example of an N-input butterfly ($N = 8$) with depth $\log N$ ($\log N = 3$) is shown in Figure 1.1. *All logarithms in this paper are base* 2. The edges of the butterfly are directed from the node in the smaller numbered level to the node in the larger numbered level. The nodes in this directed graph represent switches, and the edges represent communication links. We use the terms node and switch interchangeably in the rest of the paper. Each node in a butterfly has a label $\langle l, c_0 \cdots c_{\log N-1}\rangle$, where the *level*, l, ranges from 0 to $\log N$, and the *row*, $c_0 \cdots c_{\log N-1}$, is a $\log N$-bit binary string. The switches on level 0 are called *inputs*, and those on level $\log N$ are called *outputs*. For $l < \log N$, node $\langle l, c_0 \cdots c_l \cdots c_{\log N-1}\rangle$ is connected to node $\langle l+1, c_0 \cdots c_l \cdots c_{\log N-1}\rangle$ by a *straight* edge, and to node $\langle l+1, c_0 \cdots \overline{c_l} \cdots c_{\log N-1}\rangle$ by a *cross* edge. (The notation $\overline{c_l}$ denotes the complement of bit c_l.) At each time step, each switch is permitted to transmit one packet along each of its outgoing edges.

In a butterfly network, packets are typically sent from the inputs on level 0 to the outputs on level $\log N$. One of the nice properties of the butterfly is that there is a unique path of length $\log N$ between any input and any output, and there is a simple rule for finding that path. In particular, when a packet with origin $\langle 0, a_0 \cdots a_{\log N-1}\rangle$ and destination $\langle \log N, d_0 \cdots d_{\log N-1}\rangle$ reaches level l, it passes through the node labeled $\langle l, d_0 \cdots d_{l-1} a_l \cdots a_{\log N-1}\rangle$. If $d_l = a_l$, then it takes the straight edge to $\langle l+1, d_0 \cdots d_l a_{l+1} \cdots a_{\log N-1}\rangle$. Otherwise, if $d_l \neq a_l$, then the packet takes the cross

edge from $\langle l, d_0 \cdots d_{l-1} a_l \cdots a_{\log N - 1} \rangle$ to $\langle l+1, d_0 \cdots d_{l-1} d_l \cdots a_{\log N - 1} \rangle$. This path selection algorithm is called *source oblivious* [6] because, at each node, the next edge taken by a packet depends only on its current location and its destination, and not on its source, or on the locations or paths taken by any of the other packets. All of the routing algorithms discussed in this paper are source oblivious.

1.2. Queuing protocols. This paper studies two broad classes of queuing protocols: greedy protocols and nonpredictive protocols. In a *greedy* queuing protocol, at each step, each switch with one or more packets in its queue selects a packet and then sends it to the next level, unless the queue that the packet wishes to enter is already full. A switch is not prohibited from sending more than one packet at each step, provided that they use different edges. Nonpredictive protocols are a subclass of greedy protocols. In a *nonpredictive* queuing protocol [13, section 3.4.4], [23], at each step, each switch selects one packet from its queue *without examining the destinations of any of the packets in its queue*, and sends the packet to the next level, unless the queue that it wishes to enter is full. If the queue is full, then the switch must select the same packet at the next step. The switch is not permitted to examine the destinations of any other packets until the selected packet has been successfully transmitted. Many easily implementable as well as conceptually simple queuing protocols like FIFO and fixed-priority scheduling are nonpredictive.

1.3. Previous work. A number of different routing problems have been studied on butterfly networks. If each input of an N-input butterfly sends a single packet, we say that the network is *lightly loaded*. A specific type of routing problem of interest is the *permutation routing problem*. In a permutation routing problem, each input of the butterfly sends exactly one packet to some output of the butterfly and each output receives exactly one packet from some input of the butterfly. If each input of an N-input butterfly sends $\log N$ packets, we say that the network is *fully loaded*.

The first important butterfly routing algorithm is due to Batcher [4], who showed that an N-input butterfly network can sort, and hence route, any permutation of N-packets in $O(\log^2 N)$ steps.

The next breakthrough came more than a decade later when Valiant [32] and Valiant and Brebner [34] observed that any permutation routing problem can be transformed into two random problems by routing the packets first to random intermediate destinations and then on to their true destinations. He also showed that an N-node hypercube (or N-input butterfly) can route N packets to random destinations (or from random origins) in $O(\log N)$ time using queues of size $O(\log N)$, with high probability. As a consequence, the hypercube or butterfly can route any permutation in $O(\log N)$ time, with high probability.

Valiant's result was improved in a succession of papers by Aleliunas [2], Upfal [31], Pippenger [20], Ranade [24], and Leighton et al. [14]. All of these papers use Valiant's idea of first routing to random intermediate destinations. Aleliunas and Upfal increased the number of packets that can be routed in $O(\log N)$ time. They developed the notion of a *delay path* and showed how to route N packets in any permutation on an N-node shuffle-exchange graph and $N \log N$ packets in any permutation on an N-input butterfly network, respectively, in $O(\log N)$ steps, using queues of size $O(\log N)$. Pippenger devised an ingenious algorithm for routing with bounded size queues. He showed how to route $N \log N$ packets in any permutation on a variant of the butterfly in $O(\log N)$ steps with queues of size $O(1)$. Finally, Ranade developed a simpler algorithm for routing with bounded queues that could also efficiently combine multiple packets with the same destination. As a consequence of Ranade's al-

gorithm, it is possible to simulate one step of an $N \log N$-processor CRCW PRAM on an N-input butterfly in $O(\log N)$ steps. Neither Pippenger's algorithm nor Ranade's algorithm are greedy.

Stamoulis and Tsitsiklis [27] consider the problem of dynamic routing in butterflies and hypercubes with unbounded queues. Unlike the static routing problems that we have seen so far, they assume that the packets with random destinations are generated at each input according to a Poisson process. They show that if the load factor on the network is less than one, then the network is stable in the steady state; the average delay is $O(\log N)$, and the average queue size is $O(1)$.

Recently, Broder, Frieze, and Upfal [7] have addressed the problem of dynamic routing in butterfly and other networks with constant-size queues. They develop a powerful method to reduce the steady state analysis of dynamic routing to the better understood problem of static routing analysis. They extend the results of section 2 to the dynamic setting, to provide a greedy algorithm that routes packets in expected $O(\log N)$ time on an N-node butterfly with constant-size buffers, assuming that packets with random destinations arrive at each input with expected interarrival time $\Omega(\log N)$. Further, they extend the results of section 4 to provide an algorithm that routes packets in expected $O(\log N)$ time on an N-node butterfly with constant-size buffers, assuming that packets with random destinations arrive at each input with expected interarrival time greater than some absolute constant.

Although the performance of greedy algorithms in butterflies with bounded queues has proven difficult to analyze, attempts have been made to approximately model [29, 17] or empirically determine [30] their performance.

Finally, there have been several papers that analyze algorithms that drop packets when there is contention. The BBN Butterfly algorithm has been studied by Kruskal and Snir [12] and Koch [11]. Koch showed that for a random problem the number of packets that succeed in locking down paths from their origins to their destinations in an N-input butterfly is $\Theta(N/\log^{\frac{1}{q}} N)$, with high probability, where q is the maximum number of packets that any wire can support.

The results presented for the BBN Butterfly algorithm also hold when packets are routed in a store-and-forward fashion, with each switch having a buffer of size q, and all packets attempting to enter a full buffer are dropped. Recently, there has been progress in extending these results to the dynamic case, where packets arrive at each input with a certain interarrival distribution. Rehrmann et al. [25] show that if one packet arrives at each input of an N-input butterfly at every time step, and each switch has a buffer of size 1 at each incoming edge, then the expected network throughput is $\Theta(N/\sqrt{\log N})$ packets per time step.

1.4. Our results. In section 2 we show that for any greedy queuing protocol, routing a random problem on a lightly loaded N-input butterfly requires $O(\log N)$ steps, with high probability, provided that the queue size is a sufficiently large fixed constant. Previously, only the trivial upper bound of $O(N)$ was known. An intriguing problem left open in this section is to bound the number of steps taken by a greedy queuing protocol when the butterfly is fully loaded.

In section 3 we show that for any deterministic nonpredictive queuing protocol, there exists a one-to-one routing problem (permutation) that requires $\Omega(N/q \log N)$ time to route, where q is the maximum queue size. Previously, no lower bound greater than $\Omega(\sqrt{N})$ was known. The $\Omega(\sqrt{N})$ bound is based on the congestion and is independent of the way the packets are scheduled. This section shows that greater delays can occur due to the way packets interact in the network.

Section 4 presents a simple, but nongreedy, algorithm for routing a random problem on a fully loaded N-input butterfly with bounded-size queues in $O(\log N)$ steps, with high probability. The algorithm is simpler than the previous algorithms of Ranade and Pippenger because it does not use ghost messages, it does not compare the ranks or destinations of packets as they pass through switches, and it cannot deadlock.

Finally, in section 5 we analyze routing algorithms that drop packets when there is contention. Examples of machines that drop packets are NEC's ATOM switch [28] and the BBN Butterfly [5]. The BBN Butterfly algorithm has been studied by Kruskal and Snir [12] and Koch [11]. Koch showed that for a random problem the number of packets that succeed in locking down paths from their origins to their destinations is $\Theta(N/\log^{\frac{1}{q}} N)$, with high probability, where q is the maximum number of packets that any wire can support. By routing the packets to randomly (but not independently) chosen intermediate destinations, we show that for *any fixed permutation* the expected number of packets that reach their destinations is $\Omega(N/\log^{\frac{1}{q}} N)$.

2. Greedy queuing protocols. In this section, we study the performance of greedy queuing protocols. In section 2.1, we analyze the average case behavior of any routing algorithm with a greedy queuing protocol. We show that if every input sends a packet to a randomly chosen output, then the time required for all of the packets to reach their destinations is $O(\log N)$, with high probability. In section 2.2, we show how any specific permutation routing problem on the butterfly can be routed in $O(\log N)$ steps using Valiant's idea of splitting a routing problem into two random routing problems.

2.1. Average case behavior. We first define a few terms. A *delay tree* is a rooted tree that is a subgraph of the butterfly. Its root is a level 0 node and the tree contains a (directed) path, which we call the *spine*, from the root to a node in level $\log N$ of the butterfly. The tree "grows out" from the spine so that there is a unique directed path in the tree from the root to each node in the tree. A *full node* is defined to be a node through which the paths of at least q packets pass, where q is the maximum size of the queue in each node. Note that in the course of the routing, a full node may never have a full queue since the packets could arrive at different times. However a nonfull node can never have a full queue. A *full delay tree* is a delay tree for which every node of the tree that is not on the spine is a full node. A *maximal full delay tree* is a full delay tree that is not properly contained in any other full delay tree. The number of *packets on a delay tree* is defined to be the sum over all nodes of the tree of the total number of packets passing through each node. Note that this number is different from the number of *distinct packets on a delay tree.* In the former, if a particular packet hits (i.e., passes through) many nodes of a tree it is counted many times in the sum. The significance of the above definitions becomes clear in Theorem 2.1 below.

THEOREM 2.1. *The maximum delay of any packet is less than or equal to the maximum number of packets on a full delay tree.*

Proof. Let the path of some packet p, from its source to destination, be denoted by P. Now consider the maximal full delay tree with the path P as its spine and the source of the packet p as its root. We will refer to this maximal full delay tree as the maximal full delay tree of packet p. We will bound the delay of p by the number of packets on its maximal full delay tree. Since the tree is maximal, every nontree node that is a neighbor of a tree node is not a full node. We will now show that at each time step t until packet p reaches its destination, some packet in its maximal full

delay tree moves. At every time step t there are three cases.

(a) The packet p moves.

(b) Some other packet queued at the same node as p moves.

(c) No packet queued at the same node as p moves.

In the first two cases, it is evident that some packet in the tree moves. Since the queuing protocol is greedy, case (c) necessarily means that the packet selected to be sent at time step t by the node that contains p could not move because the queue in the node n at the next level that it wanted to enter was full. Note that node n belongs to the maximal full delay tree since it has at least q packets passing through it. Now if some packet in node n moved at time step t we are done. If not we look at the packet selected by node n and repeat the argument again. Note that case (c) cannot apply at the leaves of a tree since it does not have any neighbors with full queues. So we must encounter either case (a) or (b) before we leave the tree. Therefore the delay of packet p is at most the number of packets on its maximal full delay tree. Thus the maximum delay of any packet is at most the maximum of the number of packets on a full delay tree. □

We will use the following property of the butterfly network in the proofs in this section.

OBSERVATION 2.2. *A packet can enter a delay tree contained in the butterfly at exactly one point, and once the packet leaves the tree it can never return to it.*

We state without proof a result due to Hoeffding [10] and a Chernoff-type bound [3] and [21, p. 56].

LEMMA 2.3 (Hoeffding). *Let X be the number of successes in r independent Bernoulli trials where the probability of success in the ith trial is p_i. Let S be the number of successes in r independent Bernoulli trials where the probability of success in each trial is $p = \frac{1}{r}\sum_{1\le i\le r} p_i$. Then $E(X) = E(S) = rp$, and for α such that $\alpha E(S) \ge E(S) + 1$,*

$$\Pr[X \ge \alpha E(X)] \le \Pr[S \ge \alpha E(S)].$$

LEMMA 2.4. *Let S be the number of successes in r independent Bernoulli trials where each trial has probability p. The $E(S) = rp$, and for $\alpha > 2e$,*

$$\Pr[S \ge \alpha E(S)] \le 2^{-\alpha E(S)}.$$

THEOREM 2.5. *Let constant q be the maximum queue size. Then the maximum delay of any packet is at most $\gamma \log N$ with probability at least $1 - \frac{1}{N}$ for sufficiently large but constant γ and q.*

Proof. We will show that if there is a packet with large delay, then there must be a delay tree with a large number of packets on it, which in turn we will show to be an unlikely event. Assume that some packet p has a delay of $\gamma \log N$ or more. Consider the maximal full delay tree of this packet. Let D denote the number of packets on the maximal full delay tree of p. By Theorem 2.1, we know that $D \ge \gamma \log N$. Also since every nonspine node of this delay tree is necessarily a full node, the maximum number of nodes of this maximal full delay tree is at most $\frac{D}{q} + \log N$. Let n be a node on any level l of the butterfly. The average number of packets passing through n is 1, because there are 2^l possible packets that can pass through n and each of these packets has a probability of 2^{-l} of passing through it. Therefore, the expected number of packets on the delay tree of p is at most $\frac{D}{q} + \log N$. The gist of the remainder of the proof is to show that the number of packets on a delay tree is clustered around its expected

value. Therefore, a delay tree is unlikely to have D packets on it for sufficiently large constants q and γ.

The number of *hits* made by a packet on a delay tree is the number of nodes of the tree through which the packet passes. Let us divide the hits on a delay tree into two types: b-hits (for big hits), which are hits made by packets that make at least c hits on the tree, and s-hits (for small hits), which are hits made by packets that make fewer than c hits on the tree, where c is some constant. It must be the case that either the total number of b-hits on some delay tree is greater than or equal to $\frac{D}{2}$ (call this event E_b) or the total number of s-hits on some delay tree is greater than or equal to $\frac{D}{2}$. The latter possibility also implies that there are at least $\frac{D}{2(c-1)}$ distinct packets hitting some delay tree (call this event E_s), since each packet making s-hits can make at most $c-1$ hits on the tree. Thus, the probability that some packet has delay d is at most $\Pr(E_b) + \Pr(E_s)$.

The intuitive reason as to why b-hits are unlikely is as follows. If you imagine packets running backward in time from destination to source, once a packet enters the tree, it can remain in the tree at the next step only if it takes the unique edge to its ancestor in the tree. So, at every step, it has approximately a probability of $\frac{1}{2}$ of making another hit. This exponentially decreasing probability for making more and more hits gives us the bound. Thus, this bound uses the tree structure in a crucial way. The bound we will prove for E_s, on the other hand, is valid for any set of $\frac{D}{q} + \log N$ nodes.

Bounding the big hits. Let us suppose that event E_b occurs; i.e., there exists a delay tree of size at most $\frac{D}{q} + \log N$ with a total of at least $\frac{D}{2}$ b-hits. To bound the probability of this event we will enumerate all the possible ways it can happen. The maximum value that D can take is $N \log N$, since each packet can contribute at most $\log N$ hits and there are a total of N packets. Therefore, the number of ways of choosing D is at most $N \log N$. The number of ways of choosing the root for the delay tree is N. A binary tree of size at most $\frac{D}{q} + \log N$ can be represented by indicating the number of children (no children, left son only, right son only, both sons) in breadth-first-search order. Thus the total number of ways of choosing the delay tree is at most $N 4^{\frac{D}{q} + \log N}$. The number of different packets causing these b-hits is at most $\frac{D}{c}$, since each packet causes at least c hits and there are a total of at most D hits on the tree. Let us assume that there is some arbitrary fixed ordering of the nodes in the tree, e.g., the breadth-first-search ordering of the tree. We will now pick a nondecreasing sequence (with respect to our ordering) of $\frac{D}{c}$ nodes in the tree, $n_1, n_2, \ldots, n_{\frac{D}{c}}$. Note that each node of the tree can occur more than once in this sequence. Node n_i is the last node on the tree through which the ith packet passed. The number of ways of choosing this sequence is at most

$$\left(\begin{array}{c} \frac{D}{q} + \log N + \frac{D}{c} \\ \frac{D}{c} \end{array} \right) = \left(\begin{array}{c} \frac{D}{q} + \log N + \frac{D}{c} \\ \frac{D}{q} + \log N \end{array} \right).$$

Let node n_i of the sequence be at level l_i of the butterfly. For every n_i, we now associate a nonnegative integer h_i denoting the number of hits made by a packet p_i before leaving node n_i. The number of ways of distributing at most D hits over $\frac{D}{c}$ elements of the sequence is at most

$$\left(\begin{array}{c} D + \frac{D}{c} \\ \frac{D}{c} \end{array} \right).$$

We can ignore any n_i with $h_i = 0$ in this. Since the packet p_i must have made exactly h_i hits before leaving the tree at n_i, the number of choices for p_i is $2^{l_i - h_i}$. Here we have used Observation 2.2. The total number of ways of choosing packets for all elements in the sequence is at most $\prod_i 2^{l_i - h_i}$. (We are overcounting a little since packets have to be distinct.) Now, we have chosen a particular tree, a sequence of nodes n_i, and the associated packets p_i. The probability that all the packets p_i pass through their corresponding nodes n_i is simply the product of the probabilities that each individual packet p_i passes through node n_i which equals $\prod_i 2^{-l_i}$. (We can multiply probabilities because each packet chooses its path independently.) Putting it all together, we have

$$\begin{aligned}
\Pr(E_b) &\le N \log N \cdot N 4^{\frac{D}{q} + \log N} \cdot \binom{\frac{D}{q} + \log N + \frac{D}{c}}{\frac{D}{q} + \log N} \\
&\quad \cdot \binom{D + \frac{D}{c}}{\frac{D}{c}} \cdot \prod_i 2^{l_i - h_i} \cdot \prod_i 2^{-l_i} \\
&\le N^5 2^{2\frac{D}{q}} \cdot \left(\frac{(\frac{D}{q} + \log N + \frac{D}{c})e}{\frac{D}{q} + \log N} \right)^{\frac{D}{q} + \log N} \cdot \left(\frac{(D + \frac{D}{c})e}{\frac{D}{c}} \right)^{\frac{D}{c}} \cdot 2^{-\sum_i h_i} \\
&\le 2^{5\frac{D}{\gamma}} \cdot 2^{2\frac{D}{q}} \cdot \left(\left(1 + \frac{q}{c}\right) e \right)^{\frac{D}{q} + \frac{D}{\gamma}} \cdot 2^{\log((c+1)e)\frac{D}{c}} \cdot 2^{-\frac{D}{2}} \qquad (2.1)
\end{aligned}$$

using the inequality $\binom{x}{y} \le x^y / y!$ to bound the combinatorial coefficients and using the fact that $D \ge \gamma \log N$ and $\sum_i h_i \ge \frac{D}{2}$. Note that the multiple of D in the exponent of the first four factors decreases with an increase in the values of c, q, and γ. So for some suitably large values for the constants c, q, and γ the expression in (2.1) is at most 2^{-kD} for some constant $k > 0$. We can use the fact that $D \ge \gamma \log N$ to bound the value of this expression (and hence $\Pr(E_b)$) to be at most

$$2^{-kD} \le 2^{-k\gamma \log N} = \frac{1}{N^{k\gamma}} \le \frac{1}{2N}$$

as long as the value of γ is chosen to be at least $2/k$.

Bounding the small hits. Let us suppose event E_s occurs; i.e., there is a tree of size at most $\frac{D}{q} + \log N$ with at least $\frac{D}{2(c-1)}$ different packets hitting the tree for some value of $D \ge \gamma \log N$. The number of ways of choosing a value for D is at most $N \log N$. The number of ways of choosing such a tree is at most $N 4^{\frac{D}{q} + \log N}$. Let X denote the total number of distinct packets hitting a tree of size at most $\frac{D}{q} + \log N$. X is a sum of N Boolean random variables, $X_i, 1 \le i \le N$. Each X_i is 1 if the packet originating at input i hits the tree and 0 otherwise. The expected number of distinct packets on the tree is at most the expected number of packets on the tree. Therefore, $E(X) \le \frac{D}{q} + \log N$. Using Lemmas 2.3 and 2.4 to derive the second inequality, we have

$$\begin{aligned}
\Pr(E_s) &\le N \log N \cdot N 4^{\frac{D}{q} + \log N} \cdot \Pr\left(X \ge \frac{D}{2(c-1)} \right) \\
&\le N \log N \cdot N 4^{\frac{D}{q} + \log N} \cdot 2^{-\frac{D}{2(c-1)}} \qquad (2.2)
\end{aligned}$$

as long as $\alpha = \frac{\frac{D}{2(c-1)}}{E(X)} > 2e$. Using the fact that $D \geq \gamma \log N$, we have

$$\alpha \geq \frac{\frac{D}{2(c-1)}}{\frac{D}{q} + \log N} \geq \frac{q\gamma}{2(c-1)(q+\gamma)}. \tag{2.3}$$

Let c_0, q_0, and γ_0 be values of c, q, and γ, respectively, for which $\Pr(E_b)$ was shown to be at most $\frac{1}{2N}$. We choose the values of c, q, and γ such that *both* $\Pr(E_b)$ and $\Pr(E_s)$ are at most $\frac{1}{2N}$ as follows. First we choose $c = c_0$. Next we choose constants q and γ such that $q = \gamma$. Let τ denote the value of q and γ. We choose τ such that $\tau \geq \max(q_0, \gamma_0)$. We make $\alpha > 2e$ by choosing τ large enough such that the right-hand side of (2.3) which equals $\tau/4(c-1)$ is greater than $2e$. Furthermore, τ is chosen large enough such that

$$4^{\frac{D}{q}} \cdot 2^{-\frac{D}{2(c-1)}} = 4^{\frac{D}{\tau}} \cdot 2^{-\frac{D}{2(c-1)}} \leq 2^{-j\frac{D}{c-1}}$$

for some constant $j > 0$. Since $D \geq \gamma \log N$,

$$2^{-j\frac{D}{c-1}} \leq 2^{-j\frac{\gamma \log N}{c-1}} = 2^{-j\frac{\tau \log N}{c-1}}.$$

Finally, the value of τ is made large enough such that

$$\Pr(E_s) \leq N \log N \cdot N 4^{\log N} \cdot 2^{-j\frac{\tau \log N}{c-1}} \leq \frac{1}{2N}.$$

Note that since $c = c_0$ and $q = \gamma = \tau \geq \max(q_0, \gamma_0)$, $\Pr(E_b)$ is at most $\frac{1}{2N}$ for the chosen values of c, q, and γ. It now follows that the probability that a packet has delay greater than $\gamma \log N$ is at most $\frac{1}{2N} + \frac{1}{2N}$, which equals $\frac{1}{N}$. □

2.2. Routing a fixed permutation. The results of section 2.1 deal with the routing delay of an average routing problem. What can we say about routing a fixed permutation? We can show that we can route any fixed permutation in $O(\log N)$ steps with high probability using Valiant's idea of routing in two phases. In Phase A, each packet is routed from its source in level 0 to a random intermediate destination in level $\log N$. For simplicity, we will assume that the butterfly network has wrap-around; i.e., each node in level $\log N$ is identified with the node in level 0 in its row. The packets are queued up at the end of Phase A, and in Phase B each packet is routed to its actual destination.

THEOREM 2.6. *Any fixed permutation can be routed such that the delay is* $O(\log N)$ *with probability* $\geq 1 - \frac{2}{N}$.

Proof. Phase A is precisely the same problem as that studied in section 2.1. In Phase B, each packet is routed from its intermediate destination to its final destination. For convenience, we will denote the level of its final destination as 0 and that of the intermediate destination as level $\log N$. This phase is different from the one we studied in section 2.1 in that the starting points are random while the destinations are fixed. But the same proofs for the delay will work with small modifications. It is perhaps best to imagine the packets running backward from level 0 (final destinations) to random nodes in level $\log N$ (intermediate destinations). In the proof for bounding the b-hits, the sequence n_i will now represent switches through which packets that hit the tree *entered* the tree (running backward in time). The number of ways of associating a packet with n_i in level l_i is 2^{l_i}. The probability that the packet makes h_i hits is now $2^{-(l_i+h_i)}$, since it must leave the tree at the unique ancestor of n_i in level $l_i - h_i + 1$. The rest of the calculation is the same as before. The proof for bounding the s-hits is identical. □

3. Difficult permutations. In this section, we prove that for any deterministic nonpredictive queuing protocol, there exists a permutation that requires $\Omega(N/q \log N)$ time to route on a butterfly network. Previously, the best lower bound for routing on a butterfly with queues of any size was $\Omega(\sqrt{N})$. The $\Omega(\sqrt{N})$ bound is proved by observing that certain permutations, such as the bit-reversal permutation, force $\Omega(\sqrt{N})$ packets to pass through a single switch [13, section 3.4.2]. (It is also not very difficult to prove that if the queue size is not bounded, then $O(\sqrt{N})$ is an upper bound on the time to route any permutation using any greedy protocol.) Because the $\Omega(\sqrt{N})$ bound is based on congestion only, it applies to any queuing protocol. The results in this section indicate that the manner in which packets are scheduled can potentially cause much greater delays. The proof involves a careful examination of the interaction of the packets as they route through the network.

To simplify the presentation in this section, we will assume that each switch has a single queue, and that at each step, its two neighbors at the previous level may each send a packet into the queue provided that, at the beginning of the step, the queue held at most q packets. We call q the *queue threshold* of the switch. Since a queue can receive 2 packets when it already has q, it may contain as many as $q+2$ packets but no more.

THEOREM 3.1. *For any deterministic nonpredictive queuing protocol, there exists a permutation π that requires $\Omega(N/q \log N)$ steps to route on a butterfly with queue threshold q.*

Proof. The proof is by induction. We will assume that there are two edges leading into each butterfly input, and we begin by computing the time, $t_d(r)$, required for a depth d butterfly to accept $r/2$ packets on each of the 2^{d+1} edges into its inputs. (For simplicity, we assume without loss of generality that r and q are even.) We will assume that at time step 1 and at each time step thereafter, 1 packet is available for transmission along each of these edges until $r/2$ packets have crossed the edge. Furthermore, we will assume that each output switch can transmit one packet at each step.

We begin by examining a 1-input butterfly, which consists of a single switch, s. Suppose that at the beginning of time step 1, the queue at switch s is empty. We would like to know how long it takes for s to receive $r/2$ packets from each of its incoming edges, where $r > q$. On time steps 1 through q, s receives one packet along each of its two incoming edges. During steps 2 through q, s transmits one packet at each step. Thus, after q steps, $2q$ packets have been received, $q-1$ have been transmitted, and the queue contains $q+1$ packets. Since the queue is full, s does not receive any packets on step $q+1$, but it does transmit one. Thereafter, s receives two packets on every other step and transmits one packet on every step, until a total of r packets have been received, which occurs on step $q + (r - 2q) = r - q$. Thus, $t_0(r) = r - q$.

Next, let us compute the time required for each input of a depth-d butterfly to receive $r/2$ packets along each of its incoming edges. In order for an input to receive r packets, it must transmit at least $r-(q+2)$ packets. Using the assumption that the queuing protocol is nonpredictive, we will choose the paths of these $r-(q+2)$ packets so as to maximize the delay. Since a switch cannot look at a packet's destination until it has been selected for transmission, we can wait until a packet has been selected and then decide if it should take a cross edge or a straight edge to the next level. The first $(r-(q+2))/2$ packets selected by each input switch $\langle 0, c_0 c_1 \cdots c_{d-1} \rangle$ will be sent to the switches labeled $\langle 1, 0 c_1 \cdots c_{d-1} \rangle$. These switches are the inputs of a depth-$(d-1)$

subbutterfly. The second $(r-(q+2))/2$ packets will be sent to the depth-$(d-1)$ subbutterfly whose inputs are labeled $\langle 1, 1c_1 \cdots c_{d-1} \rangle$.

The inputs of the first subbutterfly start accepting packets at step 2. By induction, the time required for each input to receive $r-(q+2)$ packets is $t_{d-1}(r-(q+2))$. Thus, the first subbutterfly receives packets during steps 2 through $t_{d-1}(r-(q+2))+1$. In the meantime, no packets are sent to the inputs of the second subbutterfly. The first packets arrive there on step $t_{d-1}(r-(q+2))+2$ and continue to arrive until step $2t_{d-1}(r-(q+2))+1$, at which point each input has received $r-(q+2)$ packets. Thus, $t_d(r) = 2t_{d-1}(r-(q+2))+1$. Solving this recurrence yields

$$\begin{aligned} t_d(r) &\geq 2^d t_0(r-(q+2)d) \\ &\geq 2^d(r-(q+2)(d+1)). \end{aligned}$$

The lower bound on $t_d(r)$ that we have just derived requires $r > (q+2)(d+1)$ packets to pass through each butterfly input. In a permutation routing problem, however, only one packet originates at each input. In order to use the bound, we will force r packets through each input of an N/r^2-input subbutterfly that spans levels $\log r$ through $\log N - \log r$. We call this subbutterfly the *busy subbutterfly*. It has depth $d = \log N - 2\log r$. Each input of this subbutterfly is the root of a depth-$\log r$ complete binary tree whose leaves are butterfly inputs on level 0. Call these trees the *input trees*. Each output is the root of a $\log r$-depth complete binary tree whose leaves lie on level $\log N$. Call these trees the *output trees*. All of these trees are completely disjoint. The r packets that originate at the leaves of an input tree will all be sent through the root of that tree. Each output of the busy subbutterfly receives exactly r packets. These packets are distributed among the r leaves of the corresponding output tree so that they each receive exactly one packet. Note that between levels $\log r$ and $\log N - \log r$, the only switches and edges used for routing are those in the busy subbutterfly.

All that remains is to choose appropriate values of r and d. From the construction of the busy subbutterfly, we know that $d = \log N - 2\log r$. In order for our lower bound on t_d to be greater than zero, we need $r > (q+2)(d+1)$. Choosing $r = 2(q+2)(\log N + 1)$ yields $t_d(r) \geq 2^d(q+2)(\log N+1) = (N/r^2)(q+2)(\log N+1) = N/(4(q+2)(\log N+1))$. Thus the delay is $\Omega(N/q\log N)$. □

Note that the maximum number of packets passing through any node (the congestion) for the worst-case permutation constructed in this section is only $O(q\log N)$. This implies that there are other more complex routing algorithms such as that of Ranade [24] which can route this permutation in $O(q\log N)$ steps!

4. A simple routing algorithm. In this section we present a simple, but nongreedy, algorithm for routing on butterfly networks. With high probability, the algorithm requires $O(k+\log N)$ time to route packets with random destinations, where k is the number of packets that originates at each input. The algorithm is simpler than the algorithms of Pippenger [20] and Ranade [24] because it does not use ghost messages, it does not compare the ranks or destinations of packets as they pass through a switch, and it cannot deadlock. Unlike the algorithm of Ranade, however, it does not combine packets with the same destination.

The routing algorithm begins by breaking the packets into *waves*. Each input contributes one packet to each wave. The waves of packets are separated by waves of *tokens*. Unlike the ghost messages in Ranade's algorithm, a token carries no informa-

tion other than its type, which requires $O(1)$ bits to represent.[1] Initially, there are k packets at each input and a token is placed between each pair of successive packets and after the last packet. For $0 \leq i \leq k-1$, the ith packet at each input is assigned to wave $2i$, and the ith token is assigned to wave $2i+1$. Thus, the packets belong to the even waves, and the tokens belong to the odd waves. Throughout the course of the routing, the algorithm maintains the following important invariant. For $i < j$, no packet or token in the jth wave leaves a switch before any packet or token in the ith wave. Furthermore, packets within the same wave pass through a switch in the increasing order of their row numbers of origin. (A row $c_0 \cdots c_{\log N-1}$ is viewed as a binary number where c_0 is the *lower* order bit.)

A switch labeled $\langle l, c_0 \cdots c_{\log N-1} \rangle$ has two edges into it, one from the switch labeled $\langle l-1, c_0 \cdots c_{l-2} 0 c_l \cdots c_{\log N-1} \rangle$, and the other from the switch labeled $\langle l-1, c_0 \cdots c_{l-2} 1 c_l \cdots c_{\log N-1} \rangle$. We call the first edge the 0-*edge*, and the other the 1-*edge*. At the end of each of these edges is a FIFO queue that can hold q packets or tokens. We call these queues the 0-queue and the 1-queue, respectively.

The behavior of each switch is governed by a simple set of rules. By "forward" a packet or token we mean send it to the appropriate queue in the next level. If that queue is full, the switch tries again in successive time steps until it succeeds. A switch can either be in 0-mode or in 1-mode and is initialized to be in 0-mode. In 0-mode, a switch forwards packets in the 0-queue in FIFO fashion, until a token is at the head of the 0-queue. It then changes to 1-mode. In 1-mode, a switch forwards packets in the 1-queue in FIFO fashion, until a token is at the head of the 1-queue as well. Now the switch waits until both the queues at its outgoing edges have room to receive a token and then simultaneously sends one token to each of them. After doing this, the switch changes back to 0-mode.

Note that at each step a switch is required to perform only $O(1)$ bit operations in order to determine which packet, if any, to send out. In the algorithms of Pippenger and Ranade, the switches must perform more complicated operations, such as comparing the destinations of two packets as they pass through a switch. In the succeeding sections, we show that our algorithm requires $O(k + \log N)$ steps, which is asymptotically optimal.

4.1. Delay sequences. The proof that the algorithm requires $O(k + \log N)$ time uses a delay sequence argument similar to those in [1, 14, 24]. A (w, f)-delay sequence consists of four components: a path P from an output to an input; a sequence $s_1, \ldots, s_w$ of w, not necessarily distinct switches which appear in order on the path; a sequence $h_1, \ldots, h_w$ of w distinct packets and tokens; and a nonincreasing sequence of wave numbers $r_1, \ldots, r_w$. The path P may trace any edge of the network in either direction. When the path traces an edge from some level l to level $l+1$, we call the edge a *forward* edge. The number of forward edges in the path is denoted by f. The length, L, of P is equal to the distance from an output to an input ($\log N$) plus two times the number of forward edges on P, $L = \log N + 2f$. We say that a delay sequence *occurs* if, for $1 \leq i \leq w$, packet or token h_i belongs to wave r_i and passes through switch s_i. The following lemma shows that if some packet is delayed, then a delay sequence must have occurred.

LEMMA 4.1. *If some packet arrives at its destination at step* $\log N + d$ *or later, then a* $(d + (q-2)f, f)$*-delay sequence must have occurred for some* $f \geq 0$. *Further-*

[1] Tokens are used in a similar fashion in a bit-serial algorithm for routing on the hypercube in [1]. It turns out, however, that tokens are not really needed in that algorithm. Ranade's proof of the equivalence of different queuing disciplines [23] implies that a FIFO queuing protocol will suffice.

more, no two tokens in the sequence belong to the same wave.

Proof. Before we begin the proof, we need some definitions. Let the *lag* of a switch s at time t on level l be $t-l$. Also, let the *rank* of a packet h be a 2-tuple consisting of h's wave number and the row number of the input in which it originated. Ranks are examined by first comparing wave numbers, and then, if there is a tie, comparing row numbers. A row $c_0 \ldots c_{\log N-1}$ is viewed as a binary number where c_0 is the low-order bit. Note that each packet has a distinct rank. Every token belonging to the same wave has the same rank. This rank is strictly less than all the packets in the wave above it but strictly greater than the packets in the wave below it. Note that ranks are used only as a tool for the analysis and not by the algorithm itself.

The algorithm maintains several important invariants. As mentioned before, the packets and tokens leave each switch in order of nondecreasing wave number. Furthermore, each edge transmits exactly one token from each odd wave. Finally, within an even wave, the packets that arrive at a switch via its 0-edge have smaller ranks than the packets that arrive via its 1-edge. As a consequence, each switch sends out packets and tokens in order of strictly increasing rank.

The delay sequence begins with the last packet to arrive at its destination. Suppose that some packet h_1 arrives at its destination, s_1, at step τ_1, where $\tau_1 \geq \log N + d$. Then s_1 has lag at least d at step τ_1. We will construct the delay sequence by spending lag points. We begin the sequence with h_1, s_1, and r_1, where r_1 is the wave number of h_1. Next, we follow h_1 back in time until the step at which it was last delayed.

In general, suppose that we have followed some packet or token h_i back in time from some switch s_i at time step τ_i until it was last delayed, at some switch s'_{i+1} at time step τ_{i+1}. As we follow h_i back in time, the nodes that h_i passes through are added to path P. Because h_i is delayed at s'_{i+1} at step τ_{i+1}, the lag at s'_{i+1} at step τ_{i+1} is one less than the lag of s_i at step τ_i. There are three possible reasons for the delay of h_i at switch s'_{i+1}.

First, if s_{i+1} selects another packet or token, h_{i+1}, to send instead of h_i, then h_{i+1} must have a strictly smaller rank than h_i. In this case, h_{i+1}, $s_{i+1} = s'_{i+1}$, and r_{i+1} are inserted into the sequence, where r_{i+1} is the wave number of h_{i+1}. We then follow h_{i+1} back in time until it was last delayed. We have spent one lag point and inserted one packet or token into the sequence.

Second, if s'_{i+1} doesn't send h_i because the queue at the end of one of its outgoing edges is full, then we extend the path, P, forward along that edge to the switch at its head, s''_{i+1}. The lag of switch s''_{i+1} at time τ_{i+1} is two less than the lag of s_i at step τ_i. However, the queue must contain a total of q packets and tokens, all of which have smaller rank than h_i. We insert these packets and tokens into the sequence. We then follow the packet or token at the head of the queue back in time until it was last delayed.

If neither of these cases is true, it must be the case that in switch s'_{i+1} at time τ_{i+1} either of the following occurs.

(a) h_i is a packet, it is at the head of the 1-queue, and the 0-queue is empty, or

(b) h_i is a token, it is at the head of one of the queues, and the other queue is empty.

In either case, we go back to the switch at the tail of the empty queue at the previous time step. Note that we do not lose any lag by this process. We continue to do this as long as we can find an empty queue at the current switch. Suppose we do it m times and we are at a switch s''_{i+1} at time $\tau_{i+1} - m$. Switch s''_{i+1} has packets or tokens at the heads of both of its queues but did not send anything through one of its edges

at time $\tau_{i+1} - m$. If one of the heads of its queues is a packet, we add it and switch s''_{i+1} to the delay sequence and continue to follow this packet back in time. Note that in case (a), this packet belongs to the same wave as h_i but has rank strictly less than h_i since the first edge we followed back from s'_{i+1} is a 0-edge. In case (b), the packet belongs to a wave earlier than that of token h_i and hence has a strictly smaller rank. In either case, we have added a packet of strictly smaller rank for the cost of one lag point. Now suppose that both the heads of queues are tokens. The only reason the tokens were not sent at time $\tau_{i+1} - m$ is that one of the outgoing edges of s''_{i+1} had a full queue. In this case we extend the path P forward to the switch at the head of the queue, insert all of the packets and tokens in that queue into the delay sequence, and follow the packet or token at the head of the queue back in time. Now we have added q packets and tokens for the cost of two lag points.

For each lag point spent, at least one new packet or token is inserted into the delay sequence. Furthermore, for each forward edge on the path P, an additional $q - 2$ packets and tokens are inserted. Let f be the number of forward edges on P. Since we had d lag points to spend, we must insert at least $d + (q-2)f$ packets and tokens. Since we are inserting packets or tokens in strictly decreasing order of rank, at most k of these can be tokens. The length of P is $\log N + 2f$. □

4.2. Bunched delay sequences. We have now established that if some packet is delayed, then a delay sequence occurs. To simplify the rest of the argument, we will restrict our attention to delay sequences in which the packets can be partitioned into *bunches* of size b such that all of the packets in each bunch pass through the same switch on the sequence and have the same wave number. We call such a delay sequence a *bunched* delay sequence. Note that a bunched delay sequence cannot contain tokens. The following lemma shows that if a delay sequence occurs, then a bunched subsequence also occurs.

LEMMA 4.2. *If a* $(d + (q-2)f, f)$ *delay sequence occurs, then a* (bg, f) *bunched delay sequence occurs, where*

$$g = \left\lceil \frac{d + (q-2b)f - bk - (b-1)\log N}{b} \right\rceil.$$

Proof. Suppose that a $(d + (q-2)f, f)$ delay sequence occurs. We will describe an algorithm for finding a bunched subsequence.

Starting at the first switch on the sequence, s_1, form a bunch of size b of packets with wave number $2(k-1)$. If successful, then form another bunch of packets with wave number $2(k-1)$. Otherwise, if there are fewer than b remaining packets with wave number $2(k-1)$, then there are two cases to consider. First, if there are other packets on the sequence that pass through s_1, then discard the remaining packets with wave number $2(k-1)$ and begin forming bunches out of packets with the next smaller even wave number. Since the wave number can decrease at most k times, this case can happen only k times. Each time, we may discard as many as $b-1$ packets from the original delay sequence. Second, if no other packets on the sequence pass through s_1, then move on to the second switch, s_2. This case can happen at most $\log N + 2f$ times, since the path has length $L = \log N + 2f$. As in the first case, we may discard $b-1$ packets from the original sequence.

Since the original sequence contains at least $d + (q-2)f - k$ packets, and we discard a total of at most $k(b-1) + (\log N + 2f)(b-1)$ packets, at least $d + (q - 2b)f - bk - (b-1)\log N$ packets are placed in bunches. Thus, there are at least $g = \lceil \frac{d+(q-2b)f-bk-(b-1)\log N}{b} \rceil$ bunches. □

4.3. The counting argument. We are now in a position to prove that, with high probability, every packet reaches its destination within $O(k + \log N)$ steps.

THEOREM 4.3. *For any c_2, there exist constants c_1 and $q > 0$ such that the probability that any packet is delayed for more than $d = c_1(k + \log N)$ steps is at most $1/N^{c_2}$, where k is the number of packets per input of the butterfly.*

Proof. Note that, by Lemma 4.1, if some packet suffers delay d, then a $(d + (q - 2)f, f)$-delay sequence must occur for some $f \geq 0$. Therefore, using Lemma 4.2, a (bg, f) bunched delay sequence must occur, where

$$g = \left\lceil \frac{d + (q - 2b)f - bk - (b - 1)\log N}{b} \right\rceil.$$

To prove this theorem we will enumerate all possible bunched delay sequences and show that it is unlikely that any of them occurs.

The number of different bunched delay sequences is at most

$$N \cdot 4^L \cdot \binom{L+g}{g} \cdot \binom{g+k}{g} \cdot \prod_{i=1}^{g} \binom{2^{d_i}}{b}, \tag{4.1}$$

where d_i is the level of the switch through which the ith bunch passes. The factors in this product are explained as follows. There are N choices for the output switch at which the path P in the delay sequence originates. At each of the L switches on the path, there are at most four choices for the next switch on the path. There are at most $\binom{L+g}{g}$ ways of choosing the g (not necessarily distinct switches) on the path that the g bunches pass through, and at most $\binom{g+k}{g}$ ways of choosing (not necessarily distinct) wave numbers for the g bunches. Finally, given a switch with level d_i and wave number w, there are $\binom{2^{d_i}}{b}$ ways of choosing b packets with wave number w that can pass through the switch.

Whether or not a particular delay sequence occurs depends entirely on the random destinations chosen by the packets in the delay sequence. It is important to note that every packet on the delay sequence is distinct. Therefore, the events regarding any two packets on the delay sequence are independent. Thus, the probability that all of the packets pass through their corresponding switches is $\prod_{i=1}^{g} \frac{1}{2^{bd_i}}$, since each of the b packets in the ith bunch has probability $1/2^{d_i}$ of passing through any particular switch on level d_i.

We can bound the probability that *any* delay sequence occurs by summing the probabilities of each individual delay sequence occurring, which is equivalent to multiplying (4.1) by $\prod_{i=1}^{g} \frac{1}{2^{bd_i}}$. Using the inequality $\binom{x}{y} \leq (ex/y)^y$ to bound $\binom{L+g}{g}$ and $\binom{g+k}{g}$, and using $\binom{x}{y} \leq x^y/y!$ to bound $\binom{2^{d_i}}{b}$, the product is at most

$$2^{3\log N + 4f} \cdot (e(L+g)/g)^g \cdot (e(g+k)/g)^g \cdot (1/b!)^g,$$

where $g = \lceil \frac{d+(q-2b)f-(b-1)k-(b-1)\log N}{b} \rceil$. First, we choose b such that $b! \geq 16e^2$. We can make g larger than $L = \log N + 2f$, k, and $3\log N + 4f$, by making q large compared with b (but still constant), and d large compared with $b(k + \log N)$ (but still $c_1(k + \log N)$, where c_1 is a constant).

In this case, the product is at most $(8e^2/b!)^g \leq 2^{-g}$. By making g large enough, we can make this product less than $1/N^{c_2}$ for any constant c_2. ∎

5. Algorithms that drop packets. In this section, we consider queuing protocols that resolve contention by dropping packets. Two examples of machines that use this kind of protocol are the BBN Butterfly [5] and the NEC ATOM switch [28].

The ATOM switch routes packets in a store-and-forward manner. At every time step, each switch examines the head of its input queue and forwards a packet to the appropriate output queue. If a queue receiving packets is already full, then it discards packets in excess of its maximum queue size. In the BBN Butterfly, each packet tries to lock down a path between its source and destination. We will assume that each edge of the butterfly can sustain up to q such paths. Therefore, if more than q packets want to use the same edge only q succeed, and the rest fail to lock down their paths. Packets that succeed in locking down paths are transmitted along these paths to their respective destinations. This method of routing is known as circuit-switching.

In both of these queuing protocols, a natural question to ask is how many of the packets reach their destinations. The ATOM switch has not been studied in this context before. Kruskal and Snir [12] and Koch [11] studied the average case performance of the BBN Butterfly algorithm. Koch showed that if each packet independently chooses a random destination, then the expected number of packets that get through is $\Theta(N/\log^{\frac{1}{q}} N)$. However, there are permutations that arise from natural problems in which only $O(\sqrt{N})$ packets get through. To combat this we show how to route any *fixed* permutation in either of the above-mentioned queuing protocols such that the expected number of packets that reach their destinations is $\Omega(N/\log^{\frac{1}{q}} N)$. As an aside, this section also provides an elementary proof of the fact that the expected number of packets that get through for a random routing problem on the butterfly is $\Omega(N/\log^{\frac{1}{q}} N)$. As mentioned earlier, this was first proved by Koch [11].

The idea for routing any fixed permutation is based on Valiant's idea of routing to random intermediate destinations. Consider two back-to-back butterflies, i.e., two butterflies whose level $\log N$ nodes are identified. The source of the packets is level 0 of one of the butterflies, and each packet has a destination in level 0 of the other butterfly. In the first stage, the packets route from level 0 to level $\log N$ of the first butterfly. In the second stage, the packets route from level $\log N$ to level 0 of the second butterfly. In the first stage we use a scheme for sending packets to random but not independent destinations. Ranade [22] was the first to use this scheme in order to reduce the amount of the randomness needed to send packets to intermediate destinations in a packet switching algorithm. The scheme is as follows. At time step i every level i switch receives two packets, one from each of its incoming butterfly edges. The switch selects a random outgoing edge for one of the packets and routes the other packet through the remaining outgoing edge. Therefore, in the first stage no packets are dropped. In the second stage, every packet is routed from this intermediate destination to its actual destination in level 0 of the second butterfly. In this stage, packets are dropped according to the rules of BBN Butterfly routing or that of the ATOM switch. We will assume that each packet picks a random rank from 1 to $r = \log^{\frac{1}{q}} N$. When packets must be dropped, packets with the least rank are dropped in favor of those with a higher rank. We will now show that the expected number of packets that reach their destinations is $\Omega(N/\log^{\frac{1}{q}} N)$.

Let n be a node at level l of the second butterfly. Consider any k packets whose final destinations are reachable from this node. We bound the probability that all k packets pass through this node.

LEMMA 5.1. *The probability that any k packets all pass through a node n in level l of the second butterfly is at most $\frac{1}{2^{lk}}$.*

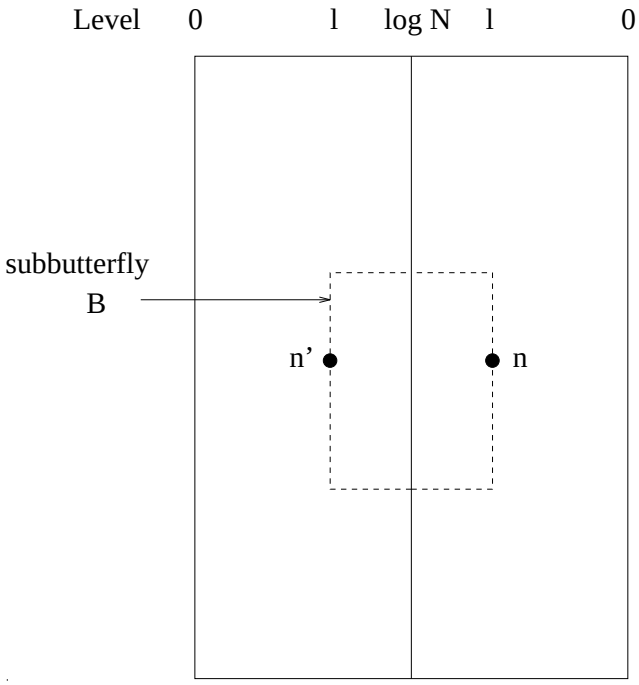


FIG. 5.1. *Subbutterfly B.*

Proof. Let the node in the first butterfly that corresponds to node n be n'. Let the subbutterfly from level l to $\log N$ of the first butterfly that contains n' be B (see Figure 5.1). Note that the k packets pass through the given node n if and only if all of these packets pass through some node of subbutterfly B. Consider the sources of the k packets in level 0 of the first butterfly and the unique shortest paths from each of the sources to subbutterfly B. If any two of them intersect before reaching the subbutterfly, these two packets cannot both hit subbutterfly B, since at the node of intersection only one packet can take the path to the subbutterfly. If no two paths intersect, then the probability of each packet hitting B is independent of the others and equals $\frac{1}{2^l}$. Thus in this case the probability of all of them hitting the subbutterfly is exactly $\frac{1}{2^{lk}}$. Therefore, given any k packets the probability of all k of them passing through the given node or equivalently hitting the subbutterfly is at most $\frac{1}{2^{lk}}$. □

THEOREM 5.2. *The expected number of packets that reach their destinations is* $\Omega(N/\log^{\frac{1}{q}} N)$.

Proof. Consider the path of a particular packet p in the second butterfly. We will now evaluate the probability that packet p reaches its destination. Note that with probability $\frac{1}{r}$ packet p will have the highest rank r. In this case, packet p can be dropped only if there is a node on its path with at least q packets going through it, all with rank r. We will now show a lower bound on the probability that there exist no such q packets. First let's bound E_q, the expected number of q-tuples of packets incident on a node at level l of the second butterfly,

$$E_q \leq \left(\begin{array}{c} 2^l \\ q \end{array} \right) \frac{1}{2^{ql}} \leq \frac{1}{q!},$$

since there are $\binom{2^l}{q}$ ways of choosing q packets that can pass through a node on level l, and by Lemma 5.1, the probability that these packets actually pass through the node is at most $1/2^{ql}$.

The expected total number of q-tuples incident on some node on the path of packet p is at most $\log N/q!$, since the path has length $\log N$. The expected number of such q-tuples with all packets having rank r is at most $\log N/(q!r^q)$ which equals $1/q!$, since $r = \log^{\frac{1}{q}} N$. Since $1/q! \leq 1$, the probability that no such q-tuple exists anywhere on the path of packet p is at least $1 - 1/q!$. (A slightly larger choice for r would make the arguments work for $q = 1$.) This implies that packet p reaches its destination with a probability of at least $(1 - 1/q!)(1/r)$, since the probability that packet p gets rank r is $1/r$. Therefore, the expected number of packets to reach their destinations is at least $(1 - 1/q!)(N/r)$ which is $\Omega(N/\log^{\frac{1}{q}} N)$. □

The proof that the expected number of packets that reach their destinations is $\Omega(N/\log^{\frac{1}{q}} N)$ also holds for a random routing problem in which each packet chooses independently a random destination. Lemma 5.1 is true because the probability that a packet passes through a node n in level l is $1/2^l$. Every packet chooses its path independently and hence the probability that all of them pass through the node exactly equals $1/2^{lk}$. The rest of the proof is the same as before. Koch [11] has observed that the expected number of packets that get through is not affected by the rule that is used to decide which messages to keep and which messages to drop, as long as the destinations of the packets are not used to make this decision. Therefore, for this problem, random ranks are necessary only as a tool for analysis and any other nonpredictive rule would exhibit the same average case behavior.

6. Open questions. The most vexing problem left open by this paper is to determine the average number of time steps required to route a random problem on a fully loaded N-input butterfly with constant-size FIFO queues. If fewer than $\Omega(\log N)$ packets may be queued at a node, then the only known upper bound on the time to route is $O(N \log N)$. This trivial upper bound is proven by showing that after $\log N$ steps, at least one packet arrives at the outputs at every time step until the routing is completed. Simulations show that the true time is closer to $O(\log N)$.

Another open question concerns the algorithm of section 4 for routing on a fully loaded butterfly with constant-size queues. We know from section 2 that a single wave of packets with random destinations can be routed using a greedy queuing protocol in $O(\log N)$ time, but when the waves are pipelined, as in section 4, the analysis requires us to use a simple, but not greedy, protocol to route each wave. It would be interesting to show that even if each individual wave was routed with a greedy protocol, the total time to route $\log N$ waves was $O(\log N)$.

Acknowledgments. The authors would like to thank Danny Krizanc, Christos Kaklamanis, Tom Leighton, Satish Rao, Nick Pippenger, Alan Frieze, and Thanasis Tsantilas for many helpful discussions.

REFERENCES

[1] W. A. Aiello, F. T. Leighton, B. M. Maggs, and M. Newman, *Fast algorithms for bit-serial routing on a hypercube*, Math. Systems Theory, 4 (1991), pp. 253–271.

[2] R. Aleliunas, *Randomized parallel communication*, in Proceedings of the ACM SIGACT–SIGOPS Symposium on Principles of Distributed Computing, 1982, pp. 60–72.

[3] D. Angluin and L. G. Valiant, *Fast probabilistic algorithms for Hamiltonian circuits and matchings*, J. Comput. System Sci., 18 (1979), pp. 155–193.

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

[5] *Butterfly*^{TM} *Parallel Processor Overview*, BBN report 6148, Version 1, Bolt, Beranek, and Newman, Cambridge, MA, 1986.

[6] A. Borodin and J. E. Hopcroft, *Routing, merging, and sorting on parallel models of computation*, J. Comput. System Sci., 30 (1985), pp. 130–145.

[7] A. Z. Broder, A.M. Frieze, and E. Upfal, *A general approach to dynamic packet routing with bounded buffers*, in Proceedings of the ACM Symposium on Theory of Computing, 1996, ACM, New York, pp. 390–399.

[8] S. Felperin, P. Raghavan, and E. Upfal, *A theory of wormhole routing in parallel computers*, in Proceedings 33rd IEEE Symposium on Foundations of Computer Science, IEEE Press, Piscataway, NJ, 1992, pp. 563–572.

[9] A. Gottlieb, *An overview of the NYU Ultracomputer Project*, in Experimental Parallel Computing Architectures, J. J. Dongarra, ed., Elsevier Science Publishers, B. V., Amsterdam, The Netherlands, 1987, pp. 25–95.

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

[11] R. R. Koch, *Increasing the size of a network by a constant factor can increase performance by more than a constant factor*, in Proceedings of the 29th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 221–230.

[12] C. P. Kruskal and M. Snir, *The performance of multistage interconnection networks for multiprocessors*, IEEE Trans. Comput., C–32(12) (1983), pp. 1091–1098.

[13] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, San Mateo, CA, 1992.

[14] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao, *Randomized Routing and Sorting on Fixed-Connection Networks*, J. Algorithms, 17 (1994), pp. 157–205.

[15] T. Leighton, B. Maggs, and S. Rao, *Universal packet routing algorithms*, in Proceedings of the 29th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, 1988, pp. 256–271.

[16] B. M. Maggs and R. K. Sitaraman, *Simple algorithms for routing on butterfly networks with bounded queues*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 150–161.

[17] A. Merchant, *A Markov chain approximation for the analysis of banyan networks*, in Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, ACM, New York, 1991.

[18] T. Nakata, S. Matsushita, N. Tanabe, N. Kajihara, H. Onozuka, Y. Asano, and N. Koike, *Parallel programming on Cenju: A multiprocessor system for modular circuit simulation*, NEC Research & Development, Princeton, NJ, 1991, pp. 421–429.

[19] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss, *An introduction to the IBM Research Parallel Processor Prototype* (*RP*3), in Experimental Parallel Computing Architectures, J. J. Dongarra, ed., Elsevier Science Publishers, B. V., Amsterdam, The Netherlands, 1987, pp. 123–140.

[20] N. Pippenger, *Parallel communication with limited buffers*, in Proceedings of the 25th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, 1984, pp. 127–136.

[21] P. Raghavan, *Lecture Notes on Randomized Algorithms*, Res. report RC 15340 (#68237), IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY, 1990.

[22] A. G. Ranade, *Constrained Randomization for Parallel Communication*, Tech. report YALEU/DCS/TR-511, Department of Computer Science, Yale University, New Haven, CT, 1987.

[23] A. G. Ranade, *Equivalence of Message Scheduling Algorithms for Parallel Communication*, Tech. report YALE/DCS/TR-512, Department of Computer Science, Yale University, New Haven, CT, 1987.

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

[25] R. Rehrmann, B. Monien, R. Luling, and R. Diekmann, *On the communication throughput of buffered multistage interconnection networks*, in Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1996, pp. 152–161.

[26] R. Rooholamini, V. Cherkassky, and M. Garver, *Finding the right ATM switch for the market*, IEEE Comput., 27 (1994), pp. 16–28.

[27] G. D. Stamoulis and J. N. Tsitsiklis, *The efficiency of greedy routing in hypercubes and butterflies*, in Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1991, pp. 248–259.

[28] H. Suzuki, H. Nagano, T. Suzuki, T. Takeuichi, and S. Iwasaki, *Output-buffer switch architecture for asynchronous transfer mode*, in Proceedings of the 1989 IEEE International Conference on Communications, IEEE Press, Piscataway, NJ, 1989, pp. 99–103.

[29] T. Szymanski and S. Shaikh, *Markov chain analysis of packet-switched banyans with arbitrary switch sizes, queue sizes, link multiplicities and speedups*, in Proceedings of the IEEE INFOCOM '89, IEEE Press, Piscataway, NJ, 1989, pp. 960–971.

[30] A. M. Tsantilas, *Communication Issues in Parallel Computation*, Ph.D. thesis, Harvard University, Cambridge, MA, 1990.

[31] E. Upfal, *Efficient schemes for parallel communication*, J. Assoc. Comput. Mach., 31 (1984), pp. 507–517.

[32] L. G. Valiant, *A scheme for fast parallel communication*, SIAM J. Comput., 11 (1982), pp. 350–361.

[33] L. G. Valiant, *General purpose parallel architectures*, in Handbook of Theoretical Computer Science, J. van Leeuwen, Ed., Elsevier Science Publishers, B. V., Amsterdam, The Netherlands, 1990, pp. 943–971.

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

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 1004–1020

FAST AND SIMPLE ALGORITHMS FOR RECOGNIZING CHORDAL COMPARABILITY GRAPHS AND INTERVAL GRAPHS*

WEN-LIAN HSU† AND TZE-HENG MA†

Abstract. In this paper, we present a linear-time algorithm for substitution decomposition on chordal graphs. Based on this result, we develop a linear-time algorithm for transitive orientation on chordal comparability graphs, which reduces the complexity of chordal comparability recognition from $O(n^2)$ to $O(n+m)$. We also devise a simple linear-time algorithm for interval graph recognition where no complicated data structure is involved.

Key words. chordal graph, triangulated graph, interval graph, analysis of algorithms, graph theory, substitution decomposition, modular decomposition, cycle-free poset, transitive orientation, graph partitioning, cardinality lexicographic ordering, graph recognition

AMS subject classifications. 68Q25, 68R10

PII. S0097539792224814

1. Notation.

All graphs in this paper are simple and have no self-loops. We also assume all graphs are connected. (For the purposes of this paper, the components of a disconnected graph can always be processed independently.) Let $G = (V, E)$ be a graph. An undirected edge between vertices u and v is denoted by uv. A directed edge from u to v is written as (u, v). For undirected graphs, the neighborhood of a vertex v, $N(v)$, is $\{w \in V : vw \in E\}$. For a set S of vertices, $N(S)=\cup_{v\in S}N(v) \setminus S$. The *degree* of a vertex v, *d(v)*, is the cardinality of $N(v)$. The *closed neighborhood* of vertex v, $N[v]$, is $\{v\} \cup N(v)$. Let $m = |E|$, $n = |V|$.

A *chordal graph* is a graph with no induced subgraph isomorphic to a cycle C_k, $k \geq 4$. Chordal graphs have been studied extensively. They are also called *triangulated*, *rigid-circuit*, and *perfect elimination* graphs. There are several subclasses of chordal graphs which have gained a lot of attention, e.g., interval graphs, split graphs, strongly chordal graphs, and chordal comparability graphs. The latter are the comparability graphs of *cycle-free* partial orders.

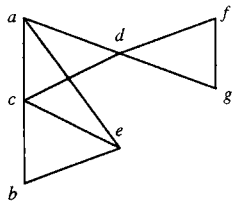
A *module* in an undirected graph $G = (V, E)$ is a set of vertices $S \subseteq V$ such that for every vertex $v \in V \setminus S$, v is adjacent to all vertices in S or no vertex in S. A module is *nontrivial* if $1 < |S| < |V|$. A *substitution decomposition* of a graph is the process of substituting a nontrivial module in the graph with a marker vertex and doing the same recursively for the module and the substituted graph. It is also called a *modular decomposition*. The process of a substitution decomposition on a graph leads to a construction of a *decomposition tree*, where each subtree represents a nontrivial module marked by its root. An example of a substitution decomposition is shown in Fig. 1.1.

For general graphs, substitution decomposition takes $O(\min(n^2, m\alpha(m, n)))$ time [15], [21]. In this paper, we call a graph *prime* if it does not contain a nontrivial module.

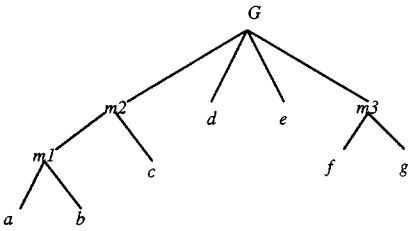
*Received by the editors January 17, 1992; accepted for publication February 13, 1998; published electronically January 29, 1999. A preliminary version of this paper appeared as *Substitution Decomposition on Chordal Graphs and Applications*, in ISA'91, Algorithms, Taipai, 1991, Lecture Notes in Comput. Sci. 557, Springer-Verlag, Berlin, 1991, pp. 52–60.

http://www.siam.org/journals/sicomp/28-3/22481.html

†Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan 11529, Republic of China (hsu@iis.sinica.edu.tw, mada@iis.sinica.edu.tw).



(a) A graph G.



(b) A substitution decompsition tree for G.
($m1$, $m2$, $m3$ are marker vertices.)

FIG. 1.1.

A directed graph $G = (V, E)$ is *transitive* if for all $u, v, w \in V$, $(u, v), (v, w) \in E \Rightarrow (u, w) \in E$. For a transitive graph G, since G has no self-loops, G must be acyclic. An undirected graph is a *comparability graph* if we can give each edge a direction such that the resultant directed graph is transitive. This process is called a *transitive orientation*. The complement of a comparability graph is called a *co-comparability graph*.

A *clique* is a set completely connected vertices. A clique is *maximal* if it is not an induced subgraph of any larger clique.

2. Introduction. A linear-time algorithm for the substitution decomposition on chordal graphs is given in this paper, which results in linear-time recognition algorithms for chordal comparability graphs and interval graphs. In the next section, we present an $O(n + m)$ algorithm for substitution decomposition on chordal graphs. Our algorithm uses a special ordering to force vertices in the same module to occur consecutively in this ordering. This algorithm decomposes chordal graphs into prime components. A prime graph has the following properties: (i) if it is a comparability graph, there is a unique transitive orientation [19] (up to the reversal of the directions of all edges); (ii) if it is an interval graph, there is a unique interval representation for the graph [13] (by which we mean there is a unique linear maximal clique arrangement up to the reversal of the interval model).

A *chordal comparability graph* is a graph which is both a chordal graph and a comparability graph. The fastest algorithm [20] for recognizing a comparability graph involves two stages. First, the input graph is transitively oriented, which can be done in $O(n^2)$ time. Then we test whether this directed graph is transitive. The fastest algorithm for the latter problem takes time proportional to that of multiplying two $n \times n$ Boolean matrices, which is currently $O(n^{2.376})$ [3]. Recently, an $O(n + m)$ algorithm has been developed to test whether a directed chordal graph is transitive

[16]. This brings the complexity of recognizing chordal comparability graphs down to $O(n^2)$. The new bottleneck is the transitive orientation of chordal graphs. In section 4, we present an algorithm which transitively orients a prime chordal comparability graph in $O(n + m)$ time. Combined with previous results, this yields an $O(n + m)$ algorithm for chordal comparability graph recognition.

A graph $G = (V, E)$ is called an *interval graph* if it is the intersection graph of a set F of closed intervals on the real line [14]. In other words, there is a one-to-one mapping between the vertices in G and the intervals in F such that two vertices are adjacent iff their corresponding intervals overlap. Interval graphs are exactly the chordal co-comparability graphs [10]. This class of graphs has a wide range of applications (cf. [11]).

Booth and Lueker [1] devised the first linear-time algorithm to recognize interval graphs using a complicated data structure called a PQ-tree. Korte and Möhring [11] simplified the operations on a PQ-tree by carrying out an incremental algorithm based on a lexicographic ordering. In section 5, we present a linear-time algorithm for recognizing prime interval graphs without using a PQ-tree. Combined with the decomposition algorithm, this yields a linear-time algorithm for interval graph recognition. We consider our algorithm to be much simpler than previous ones since there is no complicated data structure involved and the approach is intuitively appealing.

3. Substitution decomposition on chordal graphs. A vertex is *simplicial* if its neighbors form a clique. A necessary and sufficient condition for a graph to be chordal is that it admits a *perfect elimination scheme*, which is a linear ordering of the vertices such that, for each vertex v, the neighbors of v that are ordered after v form a clique. A perfect elimination scheme of a chordal graph can be obtained by taking the reverse of a *lexicographic ordering*, which can be carried out in linear time [18].

A lexicographic ordering can be considered a special kind of breadth-first ordering. Imagine there is a label of n digits, initially filled with zeros, associated with each vertex. After the ith vertex in the ordering is chosen, put a "1" into the ith digit of the labels of the neighbors of the vertex. The $(i + 1)$th vertex is then chosen among the unchosen vertices with the greatest label (with the first digit being the most significant digit). This ordering guarantees that if x is ordered before y and there is a vertex ordered before x which is a neighbor of one but not both of x, y, the first vertex added to the ordering with such property must be a neighbor of x. The linear-time algorithm is built upon a partitioning procedure. One implementation of a lexicographic ordering is shown in Fig. 3.1.

The output ordering π is the reverse of a *perfect elimination scheme* iff the input graph is chordal [18]. Perfect elimination schemes play a central role in most algorithms concerning chordal graphs.

A *cardinality lexicographic ordering* is just a lexicographic ordering with the vertices sorted by their degrees in descending order prior to the partitioning. This extra step ensures that when more than one vertex is eligible to be included to the ordering, the tie is broken by choosing a vertex with maximum degree. Since we can count the degrees and bucket sort the vertices of a graph in linear time, cardinality lexicographic ordering can also be done in linear time.

LEMMA 3.1. *Let S be a module in a chordal graph $G = (V, E)$. Either S is a clique or $N(S)$ is a clique.*

Proof. Suppose neither S nor $N(S)$ is a clique. There are $w, x \in S$, $y, z \in N(S)$, $wx, yz \notin E$. Since S is a module, both w and x are adjacent to y and z. Therefore, w, x, y, z form a C_4, which contradicts the assumption that G is chordal. □

```
procedure Lexicographic(G);
    create a list L of sets with V as the only set in L;
    /* each set in L is kept as a doubly linked list */
    for i := 1 to n do
        begin
            v := the first element of the first set in L;
            remove v;
            π(i) := v;
            split and replace each L_j ∈ L into N(v) ∩L_j and L_j \ N(v);
                /* put N(v) ∩L_j in front of L_j \ N(v) */
            discard empty sets;
        end;
end Lexicographic;
```

FIG. 3.1.

If module S is a clique, every vertex in S has the same closed neighborhood. Such a module is called a type I module. All type I modules can be located in $O(n+m)$ time by partitioning the vertices using the closed neighborhoods of all vertices. By Lemma 3.1, if there is a set with more than one vertex at the end of the partitioning process, it is a type I module.

LEMMA 3.2. *If $N[u] \subset N[v]$, then $\pi^{-1}(u) > \pi^{-1}(v)$ in every cardinality lexicographic ordering.*

Proof. Initially, v is ordered before u since v has greater degree than u. Since $N[u] \subset N[v]$, no partitioning can pull u in front of v. Therefore, v will be included in an ordering before u. □

We call a module type II if it is connected but not type I. After all type I modules are removed, a cardinality lexicographic ordering will put the vertices in a type II module consecutively and the neighborhood of the module will be ordered before the module.

LEMMA 3.3. *Let S be a connected module in a chordal graph G with no type* I *module. If π is a cardinality lexicographic ordering on G, then*

(i) $\pi^{-1}(v) < \pi^{-1}(u)$ $\forall v \in N(S)$, $u \in S$, *and*

(ii) *all vertices in S are ordered consecutively in π.*

Proof. (i) Since S is not type I, $N(S)$ is a clique. If $v \in N(S)$ and $u \in S$, then $N[v] \supseteq S \cup N(S) \supseteq N[u]$. $N[v] \neq N[u]$, otherwise u, v form a type I module. By Lemma 3.2, $\pi^{-1}(v) < \pi^{-1}(u)$.

(ii) Suppose there exists $\pi^{-1}(x) < \pi^{-1}(y) < \pi^{-1}(z), y \notin S, x, z \in S$, and $\pi^{-1}(x)$, $\pi^{-1}(y)$ are smallest possible. Since every vertex in $N(S)$ is ordered before x, and x must be the first element of S in π, when x is selected, it has a lexicographic value caused by $N(S)$. At the same moment, y, as well as all vertices in S, are in the first set in L. While we are adding vertices in S into the ordering, y is never placed into a set in front of a set containing a vertex in S since (by part (i)) $y \notin N(S)$. Since S is connected, we have a pair of adjacent vertices $x', z' \in S$, $\pi^{-1}(x') < \pi^{-1}(y) < \pi^{-1}(z')$. However, when x' is selected, z' will be placed into a set before y. Therefore, $\pi^{-1}(z') < \pi^{-1}(y)$, a contradiction. □

After getting the cardinality lexicographic ordering π, we scan the ordering from the last position. By Lemma 3.3, if there is a type II module, the vertices in the module must be in consecutive positions with all neighbors ordered before the module.

The algorithm to discover all type II modules in π tries to find the existence of such configurations. We use a "stack of stacks" to store scanned vertices. Each stack can be viewed as a candidate for a module. There are two conditions we have to enforce. First, no vertex in a stack has a neighbor in another previously created stack, since by Lemma 3.3, all neighbors of a type II module will be ordered before the module. Second, the neighborhoods of vertices in the stack should agree outside the stack. Each time a new vertex v is scanned, we try to start a new stack. If there is an edge extended from the top stack down to a lower stack, all boundaries between these two stacks must be broken. With each stack, we store the size of the stack, the common neighborhood of vertices in the stack, and the minimum $\pi^{-1}(w)$, where w is the neighbor of some but not all vertices in the stack. For a stack to be eligible to be a module, w must be included in the stack. After v is processed, if the size of the top stack is greater than 1 and the neighborhoods of vertices in the top stack agree on all the unscanned vertices, we conclude this stack forms a module.

Each time a module is reported, a minimal module is found. We then replace the top stack by a marker vertex, whose neighborhood is the CommonNeighbors of the stack. Afterwards, this marker vertex is treated the same as all other vertices. Hence, all modules will be reported in a recursive fashion. A pseudocode implementation of the algorithm is shown in Fig. 3.2. An example for the algorithm is shown in Fig. 3.3.

LEMMA 3.4. *ChordalSubstDecomp correctly finds all type* II *modules in a chordal graph in* $O(n+m)$ *time.*

Proof. Correctness: Suppose there is a type II module. By Lemma 3.3, all its vertices are in consecutive position in π; let v be the one with the greatest $\pi^{-1}(v)$ and u be the one with the smallest $\pi^{-1}(u)$ of the module. Since no vertex in this module has a neighbor after v in the ordering, after u is merged into the top stack v will still be the bottom vertex. After u is merged into this top stack, the MinDisagree of this stack is greater than or equal to $\pi^{-1}(u)$ since their neighborhoods agree on all vertices before u in π. This stack will be reported as a module.

Conversely, whenever a stack S is declared to be a module, no vertex is S has a neighbor beyond the bottom of S; otherwise, the stack would have been merged with that vertex. Moreover, since we check that the MinDisagree value is greater than or equal to the top of the stack, all vertices yet to be processed must agree on all elements of the stack. Hence the stack is in fact a module.

Time complexity: ChordalSubstDecomp steps through π and spends at constant on each vertex if we ignore the time for subroutine calls. (Since MergeTopTwoStacks can be called at most $O(n)$ times, the total cost of entering the while-loop is bounded by $O(n)$.) Each call of CreateStack(v) costs $O(|N(v)|)$ units of time. Overall, CreateStack takes $O(n+m)$ time. We assume the neighborhoods of each vertex are sorted by their indices in π (also the CommonNeighbors). The cost of MergeTopTwoStacks is proportional to the sizes of the CommonNeighbors of the top two stacks. Since the size of the CommonNeighbors of a stack is never larger than the degree of any member in the stack, we can charge this cost to the neighborhoods of the bottom vertex in the top stack and the top vertex in the second top stack. After a merge, the bottom of the top stack will never be a bottom and the top of the second top stack will never be a top. Therefore, the neighborhood of each vertex will never be charged more than twice. The total cost for MergeTopTwoStacks is then $O(n+m)$. Since all the costs are bounded by $O(n+m)$, the complexity of this algorithm is $O(n+m)$. □

```
ChordalSubstDecomp(G, π);
    i := 0; /* the number of stacks; an index */
    for j := n to 2 do
        begin
            v := π(j);
            CreateStack(v);
            while v has a neighbor in a lower stack do
                MergeTopTwoStacks;
            if the size of STACK(i) > 1 and STACK(i).MinDisagree ≥ j then
                report that vertices in STACK(i) form a module
        end;
end ChordalSubstDecomp;

CreateStack(v);
    i := i + 1;
    STACK(i).MinDisagree := n + 1;
        /* the value n + 1 indicates there is no disagreed neighbor */
    STACK(i).CommonNeighbors := N(v);
end CreateStack;

MergeTopTwoStacks;
    S := STACK(i).CommonNeighbors ∩ STACK(i − 1).CommonNeighbors;
    STACK(i − 1).MinDisagree := min( STACK(i).MinDisagree,
        STACK(i − 1).MinDisagree,
        (π^{-1}(v), v ∈ STACK(i).CommonNeighbors ∪
            STACK(i − 1).CommonNeighbors, v ∉ S) );
    STACK(i − 1).CommonNeighbors := S;
    i := i − 1
end MergeTopTwoStacks;
```

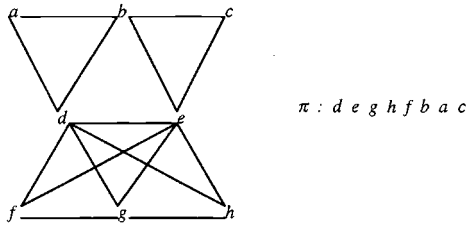
FIG. 3.2.

After we replace every type I and II module by a marker vertex, all remaining modules must be independent sets with the same neighborhoods. These modules, call them type III modules, can be found in linear time by partitioning the vertex set using their neighborhoods (as we did for finding type I modules). In conclusion, we have the following theorem.

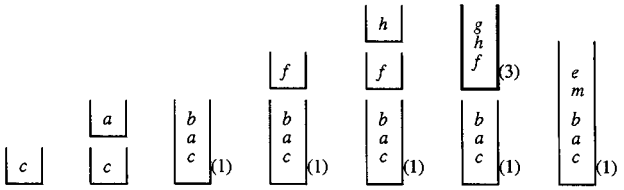
THEOREM 3.5. *The substitution decomposition on a chordal graph can be carried out in $O(n + m)$ time.*

Proof. We find type I, II, and III modules in stages. Each stage takes $O(n + m)$ time. We have to show only that at each stage, if a module of another type is generated, it will also be found.

During stage III, if a type I or II module is generated after we replace a type III module by a marker, then we must have had a connected module before the type III module is replaced by a marker. This module would have been detected in previous stages. After stage II, if a type I module S is generated, there must be some vertices in S which are markers of type II modules, otherwise S would have been detected in stage I. Suppose $v \in S$ is a marker of the last of these type II module. The module



(a) Input graph G and a cardinality lexicographic ordering.



(b) The changes of the stack configurations when π is being scanned backward. The number in the parenthesis for each nontrivial stack is its MinDisagree. Note that the stack for g,h,f is a module and is replaced by a marker vertex m.

FIG. 3.3. *An example for ChordalSubstDecomp.*

replaced by v together with $S \setminus \{v\}$ form a type II module. Therefore, by Lemma 3.4, S should be reported as a module at stage II. □

The decomposition tree can be easily constructed along with the decomposition process. Whenever a module S is reported, we replace S by a marker v with the same neighborhood as that of S. For the decomposition tree, create a tree rooted at v whose children are those vertices in S.

4. Transitive orientation on chordal graphs. In this section, we present a linear-time algorithm to transitively orient a prime chordal graph. Since a prime comparability graph admits a unique transitive orientation [19], the uniqueness provides us extra power to improve the efficiency. Together with the linear-time algorithms for substitution decomposition and transitive verification [16] on chordal graphs, we can thus recognize chordal comparability graphs in linear time.

We call $P = (X, R)$ a *partially ordered set (poset)* if X is a set and R is an irreflexive transitive binary relation on X. We say x *dominates* y if $(x, y) \in R$. If (u, v) or $(v, u) \in R$, we say that u, v are comparable. A poset $P = (X, R)$ can be viewed as a transitive graph $G = (V, E)$ by taking X as the vertex set V, R as the edge set E. Chordal comparability graphs, when transitively oriented, become a class of poset called *cycle-free* posets. For more about characteristics on cycle-free posets, see [5], [16].

A poset can be expressed by its Hasse diagram, which is an undirected graph with a minimum number of edges where there is an upward path from a to b iff a dominates b. A *chain* of a poset is a path on its diagram whose vertices are pairwise comparable. For brevity, all chains mentioned hereafter are assumed maximal unless

otherwise stated. The chains of a poset correspond to the maximal cliques of its comparability graph. Therefore, in a diagram, the vertices which appear in exactly one chain are simplicial in the poset's comparability graph.

To better understand the algorithm, the readers should keep in mind an imaginary diagram—call it the *target* diagram—which represents the poset resulting from the unique orientation of the input graph. Unlike the traditional transitive orientation algorithms, our algorithm initially does not actually orient the edges of the input graph. Instead, we explore the relative positions of vertices in the target diagram. In the end, each edge is then oriented according to the relative positions of its two endpoints.

We call a vertex w in a diagram *high* (resp., *low*) if it is dominated by (resp., dominates) a pair of incomparable vertices x, y. In a chordal comparability graph, a simplicial vertex is neither high nor low and a nonsimplicial vertex must be either high or low but not both, since if w is made high by u and v, and low by x and y, then $\{u, v, x, y\}$ induces a cycle of length 4. Thus, it can be observed that on a chain in the diagram of a cycle-free poset, the high (resp., low) vertices are above (resp., below) all simplicial and all low (resp., high) vertices. We say vertex x is higher (resp., lower) than y when there exists a chain containing both x and y where x is above (resp., below) y.

One of the most widely used characterizations of chordal graphs is that the maximal cliques of a chordal graph can be connected to form a tree T such that for each vertex v, the subgraph induced on T by the maximal cliques containing v is connected [2], [8]. (Call it the *connectivity property*.) Our algorithm applies a partitioning technique on the clique tree structure for the input chordal graph. At the end, all nonsimplicial vertices are marked either high or low and a topological sort for the target diagram is generated to provide the basis for a transitive orientation.

In our following discussion, we assume that all graphs under investigation are prime. Therefore, for any two vertices x and y, $N[x] \neq N[y]$ and $N(x) \neq N(y)$. The basic idea of our algorithm is to take a confirmed high (or low) vertex x in a certain chain and try to force the common vertices in a neighboring chain (i.e., another chain which has nonempty intersection with this one) to be low (or high). Formally, we claim the following.

LEMMA 4.1. *Let C_i and C_j be two chains with intersection S. If x is highest (resp., lowest) in C_i, $x \notin S$, then every vertex in S must be low (resp., high).*

Proof. Suppose v is the highest vertex in S. Since $x \notin C_j$, there exists $y \in C_j$, x, y are incomparable, y higher than v. Therefore, v is low and so is every vertex in S. □

To take advantage of the property of Lemma 4.1, we need to carry out our partitioning by the order where "extreme" vertices are considered first. This requirement can be easily met by the following observation.

LEMMA 4.2. *For two adjacent high (resp., low) vertices x, y, $N[x] \supset N[y]$ iff x is higher (resp., lower) than y.*

Proof. ($\Rightarrow$) Suppose $N[x] \supset N[y]$ but x is lower than y. (Note that since the graph is prime, $N[x] \neq N[y]$.) There must be a vertex z which is adjacent to x but not y. z must be higher than x since otherwise transitivity and the fact that y is higher than x would imply that y is higher than z, contradicting the fact that y and z are incomparable. y, z together with the two vertices u, v which make x high form a C_4, a contradiction.

```
procedure CliqueTree(G, π)
    create a clique C_1=π(1);
    for i := 2 to n do /* assuming the graph is connected */
        begin
            v := π(i);
            find u ∈ N(v) with maximum π^{-1} and π^{-1}(u) < i;
                /* since G is connected, u must exist. */
            let C_u be the clique generated or expanded when u is processed;
            if v and C_u form a clique, then expand C_u by adding v to it;
            else create a new clique C_j containing v and its neighbors with
                less π^{-1}, and link C_j to C_u;
        end
end CliqueTree;
```

FIG. 4.1.

($\Leftarrow$) If x is higher than y, but there exists a $z \in N[y]$ which is incomparable to x, z must be higher than y. Again, this implies the existence of a C_4. □

For an input graph G, a clique tree representation can be constructed by the algorithm CliqueTree (see Fig. 4.1). We assume that the input graph is connected. The algorithm goes through a lexicographic ordering starting from the first picked vertex. Each time a new vertex v (it must be simplicial in the subgraph induced by the previously processed vertices) is processed, we either add it to an existing clique or create a new clique for v and link the new clique to an existing one.

THEOREM 4.3. *CliqueTree creates a clique tree representation (i.e., links all maximal clique to form a tree T such that for each vertex v, the maximal cliques containing v induce a connected subtree) in $O(m+n)$ time.*

Proof. If we keep the record of the size of each clique, checking if v and C_u form a clique can be done in $|N(v)|$ time. Since each step can be done in $|N(v)|$ time as v is processed, CliqueTree can be done in $O(m+n)$ time.

For correctness, the following three conditions must be satisfied.

(i) All maximal cliques will be generated.

This can be proved by induction. The hypothesis is that the vertices $\pi(i)\pi(i-1)...\pi(1)$ processed by CliqueTree will generate all maximal cliques among them. This is trivially true when $i = 1$. When $\pi(i+1)$ is then processed, since it is simplicial, it can be in only one maximal clique. This clique is either created or expanded from an old one.

(ii) These maximal cliques are connected as a tree.

Since each new clique (except the first one) will link to exactly one existing clique, a tree structure of all existing maximal cliques is always kept.

(iii) The tree constructed has the connectivity property.

Again, we use induction. Suppose the tree constructed on $\pi(i)\pi(i-1)...\pi(1)$ is a legitimate clique tree. If $\pi(i+1)$ is added to an existing clique, there is no problem. Suppose a new clique C_p is created by $p = \pi(i+1)$, and C_p is linked to C_q which contains a neighbor q with maximum π^{-1} (which is smaller than $i+1$) and C_q is created or expanded when q is processed. Note that all the neighbors of q with smaller π^{-1} will appear in C_q. If there exists a vertex r which is in C_p and C_r but not in C_q, since $\pi^{-1}(r) < \pi^{-1}(q)$, and q and r must be adjacent (as they are both neighbors of the simplicial vertex p), r must in C_q. Therefore, the connectivity on the clique containing any vertex is kept after p is processed. □

```
procedure Orientation(G, T)
    v := a vertex of largest degree in G;
    Q := a queue containing only v;
    mark(v) := 1;
    /* a vertex u is high if it is marked 1, low if marked −1
        initially all vertices are marked 0 */
    while Q is not empty do
        begin
            remove the first vertex v from Q;
            for all tree edges e between an element of T_v and T \ T_v do
            begin
                    L := an empty list;
                    for all vertices w in e do
                        if mark(w) = 0 then
                            begin
                                mark(w) := -mark(v);
                                add w to L;
                            end;
                    sort L into order of decreasing degree;
                    append the elements of L, in order, to Q;
                    remove e from T;
            end;
        end;
end Orientation;
```

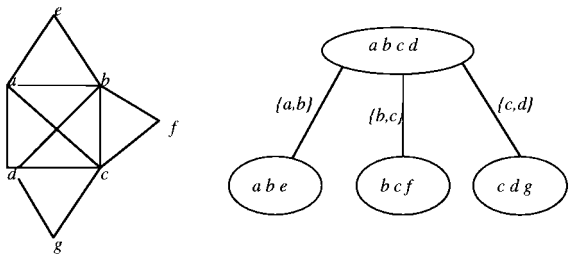
FIG. 4.2.

An example of the construction is included in Fig. 5.3. We also record the intersection of two adjacent maximal cliques on the edge connecting these maximal cliques. All these can be obtained in linear time. With this information, we are ready to perform a transitive orientation on a chordal comparability graph. The input is a chordal graph G with one of its clique tree T. Let T_v be the subtree induced by all maximal cliques containing v on T. We also assume each edge of T points to a set of vertices which are in both maximal cliques connected by that edge.

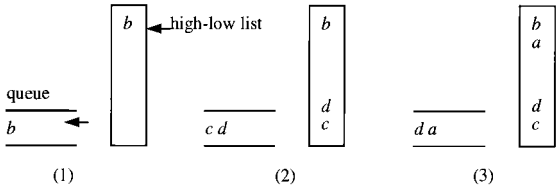
To avoid wasting time looking into an empty edge, we remove an edge on T when everything recorded in that edge is processed. This will not affect the correctness of the algorithm since our subsequent traversal is not going to pass beyond this edge anyway.

An ordering to be imposed on the vertices is constructed during our algorithm by filling in an array of length n from both ends. Whenever a vertex enters the queue, if it is marked high, it is inserted into the highest-indexed empty position in the array. Otherwise, it is inserted into the lowest-indexed empty position. Once procedure Orientation (see Fig. 4.2) has completed, we insert the remaining vertices, which are simplicial, arbitrarily into the remaining empty positions in the array. We can then orient an edge from the endpoint with a lower position to the endpoint with a higher position on the list. If the input graph is transitive, this orientation yields a partial order whose diagram is exactly characterized by the high-low relations generated by our algorithm. An example is shown in Fig. 4.3.

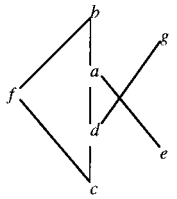
We are now ready to state the main theorem of this section.



(a) A chordal graph G and its clique tree representation. The intersections of adjacent cliques are shown in braces.



(b) The contents in the queue and the high-low list after: (1) b is selected as the initial extreme vertex and marked high; (2) b is processed (b is deleted from the braces $\{a,b\}$ and $\{b,c\}$); (3) c is processed.



(c) The target diagram constructed from the high-low list.

FIG. 4.3. *An example for the transitive orientation of a chordal graph.*

THEOREM 4.4. *A prime chordal graph, $G = (V, E)$, can be transitively oriented in $O(n + m)$ time.*

Proof. Throughout Orientation, we claim that our high-low assignments admit a feasible transitive orientation if the input graph is comparable; and the vertex x at the front of the queue must be maximal (i.e., no other vertex is higher (resp., lower) than x if x is high (resp., low)) among all unprocessed vertices. By Lemma 4.2, we know these conditions hold when we enter the while-loop. When a high vertex x is moved into the queue, all vertices higher than x are either already in the queue or they are entering the queue at the same time since, by Lemma 4.2, they exist on the same edge of the clique tree when x is detected. Since each vertex being processed is maximal among the unmarked vertices, by Lemma 4.1 and the sorting before putting vertices into the queue, we know these two conditions still hold after each operation.

The main point we need to prove is that if G is a comparability graph, every nonsimplicial vertex gets a high-low assignment when the algorithm ends. Suppose this is not true. Let x be a vertex which is high in the unique orientation after we fix the first vertex, but which is not marked high by the algorithm. Let S be the connected component containing x in the graph induced by $V \setminus M$, where M is the set of vertices marked by the algorithm. We claim that S contains more than one vertex.

Assume for a contradiction that S contains only one vertex. Then of the two vertices y, z which make x high, one (say, y) must be marked before z (y and z cannot be both simplicial, otherwise a module containing y and z can be found). There is a maximal clique containing x, y and another containing x, z, and they are connected by a path in the clique tree. When y is being processed, this path is traversed. There must be a time when y is not shared by two adjacent maximal cliques while x is. Therefore, x would have been marked opposite to y at that moment. This contradiction shows that S has at least two vertices. We claim S is a module. Suppose $u \in M$, which is not simplicial, is adjacent to v but not w, $v, w \in S$. (u must exist; otherwise $V \setminus M$ is a module.) We can find $v', w' \in S$ such that u is adjacent to v' but not w' and v', w' are adjacent. Again, there is a path in the clique tree connecting a maximal clique containing u, v' and another containing v', w'. Similar to what we have observed on x, y, z earlier, this implies that v' should have been marked opposite to u when u is used for partitioning. This contradicts $v' \in S$. Since G is prime, S cannot exist and every nonsimplicial vertex must be marked by the algorithm.

The $O(n + m)$ complexity comes from an amortized analysis [22]. The clique tree and the intersection of all pairs of adjacent maximal cliques can be obtained in linear time. When we traverse the clique tree, whenever an edge is passed, every vertex recorded in that edge will be processed (either ignored or moved to a list) in constant time and the edge of the clique tree is then deleted. Therefore, the cost of the traversal can be charged to the number of vertices stored in the edges, which is $O(n + m)$. When a set of vertices is put into the queue, we charge the cost of sorting to the edges in the clique induced by this set of vertices. Since there are quadratically many edges in a clique, we have plenty of time for sorting. After all nonsimplicial vertices are marked, the orientation can be performed in constant time per edge. The overall complexity is thus $O(n + m)$. □

COROLLARY 4.5. *Chordal comparability graphs can be transitively oriented in $O(n + m)$ time.*

Proof. We can incorporate the orientation algorithm with the substitution decomposition tree for a chordal graph. It is well known that the orientation within a module is independent to the orientation outside that module [9]. Formally, a graph is comparability iff each of its modules (including the prime graph represented at the root of a decomposition tree) can be transitively oriented. Since each module of a chordal graph is also chordal, our linear-time algorithm for prime chordal graph can be applied to each module when it is found. The overall time complexity is still linear since each edge is oriented once. □

Note that during the execution of our algorithm, we never check if an illegal orientation occurs. We only make sure that if the input graph is comparability, it will be transitively oriented correctly. With the linear-time algorithm for transitive verification for chordal graphs [16], we have the following result.

COROLLARY 4.6. *Chordal comparability graphs can be recognized in $O(m + n)$ time.*

5. Interval graph recognition. Interval graphs have been a very useful model for many applications. Interested readers are referred to [6], [9]. The fastest algorithm to recognize interval graphs relies on the following property.

THEOREM 5.1 (See [7]). *A graph G is an interval graph iff its maximal cliques can be linearly ordered such that, for each vertex v, the maximal cliques containing v occur consecutively.*

This linear ordering of the maximal cliques actually admits a clique tree which

is a path. Unfortunately, although we can generate a clique tree for a chordal graph in linear time, the clique tree constructed for an interval graph by the algorithm is not necessarily a path. Booth and Lueker [1] created a data structure called PQ-tree to capture the consecutive property of a set of intervals. Based on this data structure, a linear-time algorithm is devised to recognize interval graphs. However, their algorithm is quite involved. Korte and Möhring [11] devised a simpler algorithm to recognize interval graphs which also runs in linear time. They observed that if the input vertices follow a lexicographic ordering, the operations on the PQ-tree can be simplified.

Hsu [13] proved that an interval graph has a unique maximal clique arrangement if the graph is prime. In this section, we provide a linear-time algorithm to find the linear maximal clique arrangement of a prime interval graph. Together with the linear-time substitution decomposition algorithm, we have yet another linear-time algorithm for interval graph recognition. Our algorithm uses only basic techniques such as graph partitioning and lexicographic ordering; no special data structure such as PQ-tree is required. We consider it to be much simpler than the previous algorithms. However, we want to remind the readers that Booth and Lueker's algorithm is much more flexible in the sense that it can test the consecutive 1's property of a Boolean matrix and can operate in an "on-line" fashion. In contrast, our algorithm deals only with interval graphs and must operate in an "off-line" fashion. The readers are encouraged to make comparisons among these algorithms.

In order to better understand our algorithm, we ask the readers to keep in mind a geometric model which is the unique linear maximal clique arrangement for the input graph. Our algorithm is based on a graph partitioning idea. Initially, we obtain the clique tree representation of the input chordal graph as we did in the last section. Partition the maximal cliques into two sets such that there is a linear maximal clique arrangement (if G is an interval graph) where all the maximal cliques in one set are at the left of the maximal cliques in the other set. We then further refine our partition based on the following observation.

LEMMA 5.2. *Let A and B be two sets of maximal cliques where A is at the left of B in a linear arrangement. Suppose vertex v is shared by $C_A, C_B, C_A \in A, C_B \in B$. If X is a set of maximal cliques at the right of A, all maximal cliques in X containing v must be at the left of those not containing v. Symmetrically, if Y is a set of maximal cliques at the left of B, all maximal cliques in Y containing v must be at the right of those not containing v.*

The proof to this lemma, which is omitted here, can be observed from the geometric model of an interval graph. The refinement process is repeatedly picking a vertex which is in two maximal cliques in different sets of a partition. We can then further partition these sets by Lemma 5.2. To start the partitioning process, we need an initial partition which admits a feasible linear maximal clique arrangement if the input graph is interval. The following lemma provides an easy way to find such a configuration.

Before we prove Lemma 5.3, we investigate some basic behaviors on a lexicographic ordering. A lexicographic ordering is a breadth-first ordering. Therefore, the traversed vertices always form a connected component. Moreover, the traversal always adds vertices of a particular maximal clique until it is completely traversed. For a lexicographic ordering π on G, we say that maximal clique C_a is traversed before C_b if every vertex in C_a is included in π before the last ordered vertex in C_b.

If we focus on interval graphs with a particular geometric model, a lexicographic

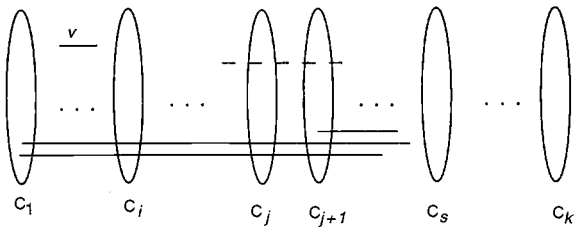


FIG. 5.1. *The scheme for proving Lemma* 5.3. *The dashed line represent an interval which cannot exist.*

ordering always picks a maximal clique to traverse first, then picks another which has the greatest lexicographic value defined by the traversed vertices, and so on. Intuitively, the traversal starts at a maximal clique and then expands toward both ends of a linear maximal clique arrangement. The maximal cliques might not be traversed consecutively as it is possible that there are more than one maximal clique with vertices which are tied in their lexicographic value.

LEMMA 5.3. *For any lexicographic ordering on a prime interval graph G, the maximal clique containing the last vertex (which is simplicial and thus contained by only one maximal clique) included in the ordering must be leftmost or rightmost on the linear maximal clique arrangement for G.*

Proof. Suppose G has k maximal cliques. Since G is prime, these maximal cliques have a unique ordering (up to reversal) $C_1, C_2, ..., C_k$, from left to right. (See Fig. 5.1.) Let C_s be the first traversed maximal clique. We claim that either C_1 or C_k will be the last traversed maximal clique. It suffices to prove that if $s > 1$, C_1 will be the last traversed maximal clique among $C_1, C_2, ..., C_s$. Suppose C_i, instead of C_1, is the last traversed maximal clique, $1 < i < s$. Keep in mind that during a lexicographic ordering, the lexicographic value of each vertex (as an interval in the linear maximal clique arrangement) is decided by its connection to the intervals that had been ordered. Let v be the first vertex which lies to the left of C_i being included in the ordering with lexicographic value α. It is clear that every unordered vertex in C_i also has the same lexicographic value. Let C_j be the maximal clique with greatest index value such that every unordered vertex in C_j also has a lexicographic value α. There is no interval connecting C_j and C_{j+1} except those that extend all the way to C_1. Otherwise, such a vertex will be ordered before v. Therefore, the set of vertices M whose interval representations end before C_{j+1} form a module. M is not trivial. Since C_1 has a simplicial vertex, and since C_i is maximal, there must be a vertex in C_i but not in C_1. This contradicts our assumption that G is prime. □

The clique tree structure provides enough information for us to partition a prime interval graph into a unique linear maximal clique arrangement. We now present a detailed description of the algorithm in Fig. 5.2. An example for this algorithm is presented in Fig. 5.3.

We now prove the main theorem of this section.

THEOREM 5.4. *Given a prime interval graph G, a linear maximal clique arrangement of G can be obtained in $O(m + n)$ time.*

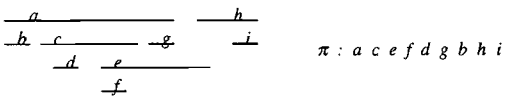
Proof. Lemmas 5.2 and 5.3 assure that LinearMaxCliqueArrange never partitions the maximal cliques incorrectly. What we need to show is that when the execution is over, a linear maximal clique arrangement can be trivially derived from the partitioning. In other words, each set of maximal cliques contains exactly one

```
procedure LinearMaxCliqueArrangement(G)
    find a clique tree of G;
    create a initial partition P which consists all maximal cliques in G;
    partition the maximal clique containing the last vertex of a lexicographic
        ordering from the rest of the maximal cliques;
    move (and delete from the clique tree) the edges in the clique tree crossing
        these two sets of maximal cliques into CrossingEdge;
    while CrossingEdge ≠ ∅ do begin
        pick an edge C_iC_j from CrossingEdge and remove it from CrossingEdge;
        for each unprocessed v ∈ C_i ∩ C_j do
            begin
                for all sets S_i in P, partition S_i into maximal cliques containing
                    v, S_i', and maximal cliques not containing v, S_i'';
                if S_i' ≠ ∅ and S_i'' ≠ ∅ then do
                    begin
                        replace S_i by S_i', S_i'' according to Lemma 5.2;
                        for each edge e in the clique tree between a member of S_i'
                            and a maximal clique not in S_i', remove e from the
                            clique tree and add it to CrossingEdge;
                    end;
            end;
    end; {while}
end Partitioning;
```

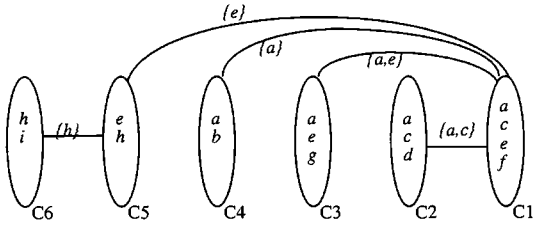
FIG. 5.2.

maximal clique. If we suppose this is not the case, then we have more than one maximal cliques in the same set S after every edge in CrossingEdge has been processed. Let V_S denote the union of vertices in the maximal cliques in S. We claim that $M = \{x|N(x) \subset V_S, x \in V_S\}$ is a module. To show this, we show that, for any vertex $u \notin M$, on the interval model, u either meets all the maximal cliques in S or none of them. If this is not true, there must exist a vertex v, $v \in C_x, C_z; v \notin C_y; C_x, C_y \in S; C_z \notin S$. From the property of a clique tree, there is a path on the subtree induced by v connecting C_x and C_z. Traversing on the path from C_x, we can find an edge on the clique tree, $C'_xC'_z$, such that $C'_x \in S$, $C'_z \notin S$. When C'_x and C'_z were first partitioned into different sets, this edge must have been put into CrossingEdge and v would have been used to partition C_x, C_y into different sets before the algorithm ends. Thus M is a module. To see that it is nontrivial, note that S must contain two distinct maximal cliques C and C', so there must be a vertex $x \in C \setminus C'$ and a vertex $y \in C' \setminus C$. Since all vertices outside M meet all or none of S, x and y must be in M, so M is a nontrivial module, which contradicts the fact that G is prime. Therefore, no set contains more than one maximal clique when the partitioning is done.

A bipartite graph H connecting the vertex set and the maximal cliques can be constructed by scanning all the maximal cliques once, such that vertex v is connected to C iff $v \in C$. The size of this graph is in $O(n + m)$. To find a clique tree and the intersections of all adjacent maximal cliques on the tree can be done in $O(n + m)$ time. The partitioning caused by vertex v can be performed in $N_H(v)$ time since all we need to do is to mark the maximal cliques containing v and split a set into



(a) The interval model of input graph G and a lexicographic ordering.



(b) The clique tree representation for G built from π.

The initial configuration:
[C6] [C5 C4 C3 C2 C1] CrossEdge = { C6C5 }

after h on C6C5 is used for partitioning...
[C6] [C5] [C4 C3 C2 C1] CrossEdge = { C5C1 }

after e on C5C1 is used for partitioning...
[C6] [C5] [C3 C1] [C4 C2] CrossEdge = { C4C1, C2C1 }

after a,c on C2C1 are used for partitioning...
[C6] [C5] [C3] [C1] [C2] [C4] CrossEdge = { C4C1, C3C1 }

FIG. 5.3. *An example for LinearMaxCliqueArrangement.*

two if necessary. To decide the left-right relation of two sets, we maintain a counter in each partitioned set recording the number of maximal cliques to the right of the set. Whenever a partition is caused by v, we examine whether v is connected to a left maximal clique (one with greater counter number) or to a right maximal clique. Since whenever an edge is moved into CrossingEdge, the edge is deleted from the clique tree, the total time needed for these movements is bounded by the number of edges in the graph. In our traversal on the maximal cliques of S'_i in order to find the crossing edges, the time we wasted on traversing noncrossing edges when a partition is caused by vertex v is bounded by $N_H(v)$. Therefore, the overall complexity is in $O(n+m)$. □

An interval model for the input graph G can be constructed from the linear maximal clique arrangement. It's easy to test in linear time whether this model is consistent with G. Therefore, we have a linear-time algorithm to recognize prime interval graphs. We can incorporate this algorithm with the substitution decomposition algorithm for chordal graphs to get a linear-time algorithm for recognizing interval graphs. Whenever a module S is identified, we test whether S is an interval graph. By Lemma 3.1, we can always replace the interval of a marker vertex v by the interval model of the module it represents and still have an interval model. If all the modules (including the last prime graph representing the input graph G) are interval graphs, G is an interval graph. We have thus proved the following corollary.

COROLLARY 5.5. *An interval graph can be recognized in linear time.*

Remark. After we submitted our paper [12], McConnell and Spinrad [17] have developed a linear-time algorithm to perform modular decomposition for general graphs. They also discussed the transitive orientation on a number of problems. Their time bound for recognizing chordal comparability graphs is $O(n + mlogn)$.

Yet another linear-time algorithm for recognizing interval graph without using PQ-tree is developed by Corneil, Olariu, and Stewart [4].

Acknowledgment. The authors thank the referees who made many helpful suggestions and a clearer presentation for procedure Orientation and the paper. We also want to thank Spinrad who pointed out an error on the clique tree construction in an earlier manuscript.

REFERENCES

[1] K. S. Booth and G. S. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 335–379.
[2] P. Buneman, *A characterization of rigid-circuit graphs*, Discrete Math., 9 (1974), pp. 205–212.
[3] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, in Proc. 19th Annual ACM Symposium on the Theory of Computation, ACM, New York, 1987, pp. 1–6.
[4] D. Corneil, S. Olariu, and L. Stewart, *The ultimate interval graph recognition algorithm*?, in Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1998, pp. 175–180.
[5] D. Duffus, I. Rival, and P. Winkler, *Minimizing setups for cycle-free ordered sets*, Proc. Amer. Math. Soc., 85 (1982), pp. 509–513.
[6] P. C. Fishburn, *Interval Orders and Interval Graphs*, Wiley, New York, 1985.
[7] D. R. Fulkerson and O. A. Gross, *Incidence matrices and interval graphs*, Pacific J. Math., 15 (1965), pp. 835–855.
[8] F. Gavril, *The intersection graphs of subtrees in trees are exactly the chordal graphs*, J. Combin. Theory B, 16 (1974), pp. 47–56.
[9] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
[10] P. C. Gilmore and J. J. Hoffman, *A characterization of comparability graphs and of interval graphs*, Canad. J. Math., 16 (1964), pp. 539–548.
[11] N. Korte and R. H. Möhring, *An incremental linear-time algorithm for recognizing interval graphs*, SIAM J. Comput., 18 (1989), pp. 68–81.
[12] W. L. Hsu and T. H. Ma, *Substitution decomposition on chordal graphs and applications*, in ISA '91, Algorithms: 2nd International Symposium on Algorithms, Lecture Notes in Comput. Sci. 557, Springer-Verlag, Berlin, 1991, pp. 52–60.
[13] W. L. Hsu, *O(m · n) algorithms for the recognition and isomorphism problems on circular-arc graphs*, SIAM J. Comput., 24 (1995), pp. 411–439.
[14] C. G. Lekkerkerker and J. Boland, *Representation of a finite graph by a set of intervals on the real line*, Fund. Math., 51 (1962), pp. 45–64.
[15] J. H. Muller and J. Spinrad, *Incremental modular decomposition*, J. Assoc. Comput. Mach., 36 (1989), pp. 1–19.
[16] T. H. Ma and J. Spinrad, *Cycle-free partial orders and chordal comparability graphs*, Order, 8 (1991), pp. 175–183.
[17] R. McConnell and J. Spinrad, *Linear-time modular decomposition and efficient transitive orientation of comparability graphs*, in Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, 1994, SIAM, Philadelphia, pp. 536–545.
[18] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.
[19] L. N. Shevrin and N. D. Filippov, *Partially ordered sets and their comparability graphs*, Siberian Math. J., 11 (1970), pp. 497–509.
[20] J. Spinrad, *On comparability and permutation graphs*, SIAM J. Comput., 14 (1985), pp. 658–670.
[21] J. Spinrad, *P_4 trees and substitution decomposition*, Disc. Appl. Math., 39 (1992), pp. 263–291.
[22] R. E. Tarjan, *Amortized computational complexity*, SIAM J. Alg. Disc. Math., 6 (1985), pp. 306–318.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 1021–1034

FAST CONNECTED COMPONENTS ALGORITHMS FOR THE EREW PRAM*

DAVID R. KARGER†, NOAM NISAN‡, AND MICHAL PARNAS‡

Abstract. We present fast and efficient parallel algorithms for finding the connected components of an undirected graph. These algorithms run on the exclusive-read, exclusive-write (EREW) PRAM. On a graph with n vertices and m edges, our randomized algorithm runs in $O(\log n)$ time using $(m + n^{1+\epsilon})/\log n$ EREW processors (for any fixed $\epsilon > 0$). A variant uses $(m + n)/\log n$ processors and runs in $O(\log n \log\log n)$ time. A deterministic version of the algorithm runs in $O(\log^{1.5} n)$ time using $m + n$ EREW processors.

Key words. connected components, parallel algorithms, graph algorithms, random walks

AMS subject classifications. 68Q22, 68Q25, 68R10, 05C40, 05C85, 60J15

PII. S009753979325247X

1. Introduction.

Perhaps the most basic algorithmic problem involving an undirected graph is to find its connected components. In this problem, the input to the algorithm is an undirected graph $G = (V, E)$, with $|V| = n$ vertices and $|E| = m$ edges. The output is the connected components of the graph. There are various ways to represent the solution; the one we shall use is to label each vertex with the largest numbered vertex to which it is connected. Connected components can be found in linear sequential time by breadth-first search or depth-first search methods. However, these methods do not parallelize easily. Parallel algorithms for connected components have been known for quite some time ([HCS79], [CLC82], or see the survey in [KR90]). Until recently the best known algorithms required $O(\log n)$ time on CRCW PRAMs and $O(\log^2 n)$ time on CREW PRAMs (recall that CR (CW) PRAMs allow multiple processors to concurrently read (write) to the same memory location, while ER (EW) PRAMs allow only one processor to read (write) at a time). The number of processors used by the best of those algorithms is nearly optimal in the deterministic case [SV82, CV91, AS87] and completely optimal in the randomized case [Gaz91].

In their survey, Karp and Ramachandran [KR90] raised the question of the existence of $o(\log^2 n)$-time algorithms for connected components on exclusive-write PRAMs. Recently, Johnson and Metaxas [JM91] developed a CREW algorithm that runs in $O(\log^{1.5} n)$ time and uses $m+n$ processors. An $O(\log^{1.5} n)$-time algorithm for the EREW PRAM is described in [NSW92], but this algorithm uses a large polynomial number of processors.

*Received by the editors March 6, 1993; accepted for publication (in revised form) February 16, 1997; published electronically January 29, 1999. A preliminary version of this paper appeared in *Proc. 4th Annual ACM-SIAM Symposium on Parallel Algorithms and Architectures*, 1992, pp. 562–572.

http://www.siam.org/journals/sicomp/28-3/25247.html

†MIT Laboratory for Computer Science, 545 Technology Square, Room NE43-321, Cambridge, MA 02139 (karger@theory.lcs.mit.edu). This research was supported by a National Science Foundation Graduate Fellowship, by NSF grant CCR-9010517, and grants from Mitsubishi and OTL.

‡Institute of Computer Science, Hebrew University of Jerusalem, Jerusalem 91904, Israel (noam@cs.huji.ac.il, michalp@cs.huji.ac.il). The research of the second author was supported by the Wolfson Research Awards administered by the Israel Academy of Sciences and Humanities and by U.S.A.-Israel BSF 89-00126.

In this paper we present improved EREW algorithms for connected components. One contribution is the first (randomized) algorithm that runs in $O(\log n)$ time. It is based on the parallelization of random walk techniques studied in [AKL*79], where it is shown that a relatively short random walk will visit all the vertices in a graph.

THEOREM 1.1. *The connected components of an undirected graph can be computed on a randomized EREW PRAM in $O(\log n)$ time with high probability*[1] *using $(m + n^{1+\epsilon})/\log n$ processors for any fixed $\epsilon > 0$.*

The running time of this algorithm is optimal, as the results of [CDR86] and [DKR94] imply a lower bound of $\Omega(\log n)$ time even on a randomized CREW PRAM. For graphs that are not too sparse, i.e., with $\Omega(n^{1+\epsilon})$ edges, the processor costs are optimal as well, because the total work remains linear in the input size. For sparse graphs, using a linear number of processors slightly increases the running time.

THEOREM 1.2. *The connected components of an undirected graph can be computed on a randomized EREW PRAM in $O(\log n \log\log n)$ time with high probability with $(m + n)/\log n$ processors.*

This is of course within an $O(\log\log n)$ factor of being work optimal. An important related open problem is to design a *deterministic* $O(\log n)$-time algorithm. We have made some progress in this direction.

THEOREM 1.3. *The connected components of an undirected graph can be computed on a deterministic EREW PRAM in $O(\log^{1.5} n)$ time using $m + n$ processors.*

The running time of this deterministic algorithm matches those of [JM91] and [NSW92]. It improves upon [JM91] by working in the more restricted EREW model instead of in the CREW model. It improves upon [NSW92] by requiring only a linear number of processors instead of a large polynomial number of processors. This last result was proved independently (using a different method) by [JM92].

After publication of the preliminary version of this paper [KNP92], several improvements were given. Chong and Lam [CL95] gave an $O(\log n \log\log n)$-time deterministic algorithm that uses $m + n$ processors. Halperin and Zwick improved our methods to yield first [HZ94] an optimal randomized algorithm for connected components that runs in $O(\log n)$ time with a linear number of processors, and subsequently [HZ6] an optimal randomized algorithm for finding a spanning forest of the graph (note that our algorithm does not find spanning forests).

In the following sections, we present the connectivity algorithm. To simplify the exposition, we first present a randomized algorithm that uses $m + n$ rather than $(m+n)/\log n$ processors. We give a general overview of the algorithm and then fill in the details and provide proofs of correctness. We then discuss the changes needed to make the algorithm deterministic. The modifications needed to reduce the processor cost by an additional factor of $\log n$ are somewhat complex and are left to a later section.

2. Overview of the algorithm. The algorithm is based on a simple and well-known idea: repeatedly find groups of connected vertices in the graph and contract (i.e., merge) each group into a single vertex, finishing when each connected component is contracted to a single vertex. The question lies in how to find these connected groups.

Suppose we are fortunate and the minimum degree of the graph is large. Let $N(i)$ be the set of vertices adjacent to i (including i). The following procedure can

[1] By "high probability" we mean that the probability of the event not happening can be made at most $n^{-\delta}$ for any fixed $\delta > 1$ without affecting the orders of run times.

be applied, using a processor for each vertex and a processor for each edge.

1. Each vertex looks at all vertices within distance two of itself, i.e., $N(N(i))$. If it finds a larger numbered vertex than itself, it makes this vertex its *parent*. Any vertex that fails to find a parent becomes a *leader*.
2. The selection of parents has created a group of trees with leaders at the roots (note that the tree edges need not be graph edges). Each tree is now contracted to a single vertex.

Assuming that the minimum degree is large, this process yields a much smaller graph, as the following lemma shows.

LEMMA 2.1 (neighborhoods). *If all neighborhoods $N(i)$ have size at least s, then at most n/s leaders can exist.*

Proof. The distance between two leaders must exceed 2. Thus the neighborhoods of two leaders are disjoint, and therefore at most n/s leaders remain. □

The contraction of the trees does not change the connected components, as can be seen from the following lemma.

LEMMA 2.2. *In the contracted graph, two leaders are connected iff they were connected in the old graph.*

Proof. Note that two leaders are adjacent iff each had a descendant such that the descendants were adjacent. The lemma then follows from the fact that all vertices are necessarily connected to their leaders. □

The running time of the algorithm will depend on the number of rounds needed to contract every connected component into a single vertex. Since the reduction is based on neighborhood size, this number of rounds depends on the minimum degree s of the graph. The problem is that the minimum degree of the graph may be small, and therefore the procedure described above may fail to reduce the size of the graph significantly. We will show how to solve this problem by "imagining" additional edges in the graph in order to make the neighborhoods large. As long as the imaginary edges connect vertices that are connected in the original graph, the two lemmas given above continue to hold. Similar ideas are explored in [BR91] and [NSW92]. The remainder of this paper is dedicated primarily to the question of how to construct quickly and in parallel a large neighborhood for each vertex in the graph.

Our approach to this question began with the following observation. It is known that an EREW PRAM with a polynomial number of processors can simulate any logspace algorithm in $O(\log n)$ time (see [KR90]). This extends to the fact that a randomized EREW PRAM can simulate randomized-logspace algorithms.[2] Since a randomized-logspace algorithm for connectivity is known [AKL*79], a randomized EREW algorithm follows.

Unfortunately, the parallelization of the random walk algorithm of [AKL*79] requires $\Omega(mn^2)$ processors. The reason so many processors are needed is that a random walk of length $\Omega(mn)$ must be taken to be sure of covering the entire graph. Thus the approach of [AKL*79] does not directly suggest an efficient algorithm. However, an important idea can be extracted from this approach, namely, that a random walk visits a large number of vertices relatively quickly. We thus explore the use of *short* walks on the graph.

Consider taking a walk of some length p from each vertex of the graph by traveling along edges of the graph. Using the vertices encountered along each walk as the

[2] In fact, the results in [Nis93] imply that any randomized-logspace algorithm with bounded two-sided error can be simulated with *zero error* by a randomized EREW PRAM in $O(\log n)$ time using a polynomial number of processors.

neighborhood of the walk's starting vertex, we will apply the contraction procedure described above to reduce the size of the graph by a significant factor. This procedure will be called a *walk phase of length p*. In the randomized algorithm, the walk is a random walk; in the deterministic algorithm, the walk is based on a deterministic traversal sequence. A walk phase takes $O(\log n)$ time to simulate in parallel, but since the walk phases construct large neighborhoods, a very small number of phases suffices to complete the algorithm. In more detail, consider the following procedure.

1. From each vertex $1 \leq i \leq n$, take a walk of length p. Let $W(i)$ be the *itinerary* of i, i.e., the set of vertices seen on the walk that starts at i.
2. Consider the edges defined by the walks, so that there is an edge $\{i, j\}$ if $j \in W(i)$ or $i \in W(j)$. These edges clearly connect vertices that are connected in G.
3. Use these "walk edges" to define the vertex neighborhoods. Each vertex examines its neighborhoods, as was described at the start of this section, to find a parent or become a leader.
4. To contract the resulting collection of trees, each vertex finds the leader in its tree of parents and transfers all its edges to the leader (i.e., replaces each edge $\{i, j\}$ in G by an edge from i's leader to j's leader).

When a walk phase is finished, we have a new graph G' whose vertices are the leaders in the old graph. Lemmas 2.1 and 2.2 tell us that G' is smaller than G and that G' embodies the same connectivity information.

The connectivity algorithm is to repeat the walk phases until the resulting graph has no edges. Each remaining vertex then represents the connected component containing that vertex. Every vertex that does not remain will have dropped out after selecting some vertex as its leader and giving that vertex its edges. The leader choices of the vertices form a forest—the root of each tree is one of the connected component representatives, and the vertices in each tree are a single connected component of the graph. Tree contraction can now be used to let each vertex identify its connected component representative.

To make the algorithm run quickly, we need to finish in a small number of walk phases. From this description, it can be seen that all we need in order to implement the algorithm are

- a walk that visits a large number of vertices, and
- a way to simulate a walk phase quickly in parallel.

We now show in detail how to achieve these two goals.

3. Implementing a walk phase. In the course of the following discussion of the implementation of the algorithm, assume that G is totally connected. The results we wish to prove then follow by independently considering the action of the algorithm on each connected component of the graph.

The key question that must be solved is how to construct a walk that visits a large number of vertices. Using randomization, the solution is straightforward. Knowing that a random walk expects to cover all n vertices of a graph in time $n^{O(1)}$, we will deduce that a random walk of length p visits $p^{\Omega(1)}$ vertices with high probability. This reduces the implementation of a walk phase to the problem of simulating a random walk from each vertex in parallel.

It is well known that CRCW can be simulated on an EREW machine with an $O(\log n)$ slowdown in running time and no increase in processor cost [KR90, pp. 894–895]. Since we wish the walk phase to have running time $O(\log n)$, we feel free to say that "processors concurrently read or write," so long as this occurs only a constant

number of times.

We now discuss the details of implementing a walk phase of length p using $m+pn$ processors.

3.1. Data structures and processor allocation. Each vertex i has a list L_i of edges leaving i. The edge connecting i and j appears as (i,j) in L_i and as (j,i) in L_j. The edge lists are stored contiguously in one array L of length $2m$, sorted by order of lists $L_1, L_2, \ldots, L_n$.

Each vertex i also has two variables, $first_i$ and $last_i$, which indicate the beginning and end of the list L_i in L. It is easy to determine the number of neighbors of i by computing $last_i - first_i + 1$.

The algorithm uses an $n \times p$ array $WALK$ to simulate random walks of length p. Two more arrays, MAX of dimensions $n \times p$ and $PARENT$ of length n, are used to find the leaders of the graph.

We use $O(\log n)$ time to redistribute the processors at the beginning of each walk phase.

- p processors are assigned to each vertex. Let $P_{i,1}, P_{i,2}, \ldots, P_{i,p}$ be the processors assigned to vertex i for $i = 1, \ldots, n$.
- One processor is assigned to every edge. These processors will be called *edge processors*. Let P_k, where $k = 1, \ldots, m$, be these processors assigned to edge k.

Therefore, the total number of processors is $m+pn$. Notice that after each walk phase the number of vertices of the graph reduces, and therefore in each walk phase we can allocate more processors per vertex. This will allow us to increase the length p of a walk phase and thus to contract the connected components even faster.

We now turn to the details involved in executing the four steps outlined in the overview of the algorithm.

3.2. Step 1: Simulating the random walk. We wish to simulate the process of taking a random walk of length p simultaneously from all vertices of G. For each vertex $1 \le i \le n$ and each $1 \le t \le p$, processor $P_{i,t}$ chooses a neighbor of vertex i uniformly at random and writes it into $WALK[i,t]$. Each processor does so using three concurrent reads of $first_i$, $last_i$, and L.

Consider the random variables u_t^i defined by

$$\begin{aligned} u_1^i &= i, \\ u_{t+1}^i &= WALK[u_t^i, t] \quad \text{for} \quad t = 1, \ldots, p. \end{aligned}$$

By the choice of the $WALK[i,t]$ values, the random variable u_t^i is a random walk starting at vertex i for each $1 \le i \le n$. The random walks with different sources are not independent, but this will not affect the analysis. As mentioned in the overview, let the itinerary $W(i)$ be the set of vertices encountered on the random walk from i. Figure 1 shows a filled $WALK$ array in which, for example, $W(8) = \{7, 8, 9\}$ (in the algorithm, each walk step moves from one vertex to a different vertex, but in our picture, for the sake of clarity, we have drawn some horizontal edges that imply walk steps that stand still). We will show later that all the $W(i)$ are large, i.e., of size exceeding $p^{\Omega(1)}$.

3.3. Step 2: Finding neighborhoods. As stated in step 2 of the outline, consider the *walk edges* defined by including (i,j) if $i \in W(j)$ or $j \in W(i)$. In step 3 each vertex looks for a parent among vertices up to two walk edges away. These

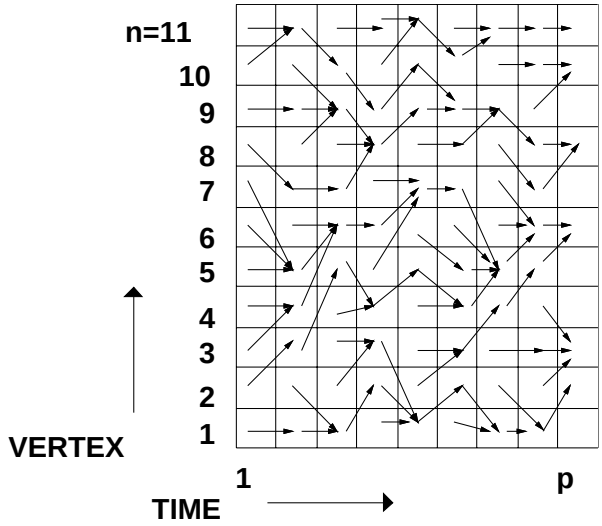


FIG. 1. *The WALK array.*

edges are not actually constructed; instead, each vertex deduces the information it needs directly from the *WALK* array. Furthermore, the edges considered are actually a superset of the walk edges, which will define larger neighborhoods $N(i) \supset W(i)$. But it will still be true that i is connected to all vertices in $N(i)$. Since larger neighborhoods cause a greater reduction in the size of the graph, this use of more edges can only help.

The values placed in the walk array in step 1 can be seen to define a collection of trees (the values provide parent pointers). We let the neighborhood of a vertex be the vertices to which it is connected by one of these trees. More formally, for each $i = 1, \ldots, n$, let $T_{i,t}$ be the set of array entries $[j, t']$ that are in the tree containing the entry $[i, t]$ and let $T_i = \bigcup_t T_{i,t}$. Define $N(i) = \{j \mid (\exists t)[j, t] \in T_i\}$ to be the neighborhood of vertex i. In other words, imagine an edge from i to j whenever i and j share a tree. Note that if the random walk from i encounters j then $[j, t] \in T_i$ for some t, so $W(i) \subset N(i)$. Also, since each tree edge corresponds to a step in a random walk, and thus to an edge in G, all members of $N(i)$ are necessarily connected to i.

Let H denote the graph with vertex set V but with edges defined by the neighborhoods $N(i)$. This is the graph that will be used to find connected sets of vertices to contract.

3.4. Step 3: Choosing leaders. We now implement the process of choosing a maximum parent of distance at most two in H, as described in the overview of section 2. This is achieved by calling twice the following procedure *Max-Neighbor*. The first call to this procedure chooses for each vertex i the maximum vertex in $N(i)$ (which corresponds to finding the maximum vertex of distance one from each vertex on the graph H). The second call finds the maximum vertex chosen by any vertex in $N(i)$ (which corresponds to finding the maximum vertex of distance two from each vertex on the graph H). Initialize the array $PARENT$ to $PARENT[i] = i$, and then call the following procedure twice.

Procedure *Max-Neighbor:*

1. For each $i = 1, \ldots, n$ and $t = 1, \ldots, p$, set $MAX[i, t] = \max_{[j,t'] \in T_{i,t}} PARENT[j]$.

2. For each $i = 1, \ldots, n$, set $PARENT[i] = \max_t MAX[i, t]$.

At the first iteration this process labels each vertex with its largest "neighbor," so the second iteration labels vertices with their largest neighbor at distance two. In other words, at the end of this process $PARENT[i]$ contains the parent of vertex i. Vertex i is a leader if $PARENT[i] = i$.

Implementing *Max-Neighbor* is straightforward. Step 2 is trivial. Step 1, maximizing over a tree, can be implemented using Euler tour techniques on the array *WALK* in time $O(\log n)$ using np processors (see [KR90, pp. 879–883]). The only nonstandard detail is that our filling up of the walk array has created trees with unidirectional edges, while the Euler tour method requires bidirectional edges. To build these edges, proceed as follows. Copy the *WALK* array, and then sort the edges $([i, t], [j, t+1])$ in the *WALK* array according to their second endpoints. We can do this in $O(\log np)$ time using np processors, either by applying Cole's sorting algorithm [Col88] or via a simple bucket sort using $(np)^2$ space. After the sort, all the edges that point to a particular position in the *WALK* array are grouped together for application of the Euler tour technique.

3.5. Step 4: Create the new graph G′. In this final step we must construct the new graph G' whose vertices are the leaders in the graph G. The selection of parents in procedure *Max-Neighbor* created a group of trees (not to be confused with the trees in the *WALK* array) with leaders at the roots. Each vertex now finds the leader at the root of its tree by using Euler tours as before [KR90]. We can now create the new smaller graph $G' = (V', E')$. The set of vertices V' is the set of leaders, i.e., $V' = \{leader(i) \mid i \in V\}$. The set of edges of G' is obtained by transforming each edge (i, j) of E to an edge $(leader(i), leader(j))$, i.e., $E' = \{(leader(i), leader(j)) \mid (i, j) \in E\}$.

To construct the set E', each edge processor P_k for $k = 1, \ldots, m$ concurrently reads the leaders of each of its endpoints and renames its edge appropriately. If P_k is handling edge (i, j), then P_k checks if $leader(i) = leader(j)$. If so, this edge has been contracted and is now useless, so P_k writes 0 at $L[k]$. If not, it writes the edge $(leader(i), leader(j))$ at $L[k]$.

Next sort L lexicographically by left and right endpoints in $O(\log n)$ time using Cole's sorting algorithm [Col88] or a bucket sort (for the bucket sort, a binary tree atop an array of size m can be used to let all items with a particular key merge into a list by walking up the tree). The renaming may yield multiple copies of some edges. These must be removed because otherwise the random walk becomes "biased" toward visiting vertices that are connected by many edges; our analysis requires that the random walk be unbiased. To remove these multiple edges, each edge processor P_k looks to its left at L. If $L[k] = L[k-1]$, then P_i writes a 0 at $L[k]$. Now compact the array L using standard parallel compaction [KR90, pp. 875–876].

Next it is necessary to update the $first_i$ and $last_i$ variables. To do so, first set $first_i = -1$ (using the processors $P_{i,1}$). Then each edge processor P_k looks to its left (right) at L, and if it is at the beginning (end) of the edge list of some processor i, it updates $first_i$ ($last_i$). Afterward, any vertex that still has $first_i = -1$ must have no incident edges. Such isolated vertices are marked as the representatives of connected components and removed. It should be noted that the new graph G' still has at most m edges.

Clearly, all the operations described above can be done in $O(\log n)$ time. Thus we have proved Lemma 3.1.

LEMMA 3.1. *A walk phase of length p can be implemented in $O(\log n + \log p)$*

time using $m + pn$ EREW processors.

4. Iterating the walk phase. Now that we have shown the required time and processor bounds, it remains to show that the new graph G' has significantly fewer vertices, and that as a consequence the algorithm terminates in a small number of walk phases. We require the following corollary to the known results regarding the cover time of random walks on graphs. This lemma was first observed by Linial [Lin]. For completeness we sketch the proof.

LEMMA 4.1. *Let G be an undirected graph. Let v be any vertex in G that is contained in a connected component of at least t vertices. Then the expected time needed for a random walk starting from v to see t vertices is $O(t^4)$.*

Proof. Define the random variable X_t to be the time it takes a random walk that starts at v to see t vertices. Assume that by time X_t we saw the set of vertices C_t. Let $w \notin C_t$ be a vertex that is adjacent to some vertex in C_t. Then the expected time to cover the graph $C_t \bigcup w$ (and thus see a new vertex) is $O(t^3)$ if we do not leave $C_t \bigcup w$. If we do leave it, then we shall see a new vertex even sooner. Hence $E(X_{t+1}) = E(X_t) + O(t^3) = O(t^4)$. □

It was shown by Barnes and Feige [BF93] that $O(t^3)$ expected time is sufficient to see t vertices. We can now obtain Lemma 4.2

LEMMA 4.2. *After a walk phase of length p, for every vertex $i = 1, \ldots, n$, the itineraries satisfy $|W(i)| = \Omega((\frac{p}{\log n})^\alpha)$ with high probability, where $\alpha = 1/4$.*

Proof. Consider the walk to be a composition of $\Omega(\log n)$ "subwalks" of equal length $\Omega(\frac{p}{\log n})$. Call each subwalk *good* if it visits $\Omega((\frac{p}{\log n})^\alpha)$ vertices. By the Markov inequality and Lemma 4.1, each subwalk has a constant probability of being good. This is true even if we condition on the outcomes of previous subwalks. Thus all the subwalks fail to be good with polynomially small probability. □

The result of [BF93] lets us take $\alpha = 1/3$, thus improving the constant factors in the following analysis.

COROLLARY 4.3. *A walk phase of length p reduces the number of vertices in a graph by a factor of $\Omega((\frac{p}{\log n})^\alpha)$ with high probability.*

Proof. Consider the above two lemmas and the fact that $W(i) \subset N(i)$. Now apply Lemma 2.1. □

We can now analyze the running time of our connected component algorithm.

LEMMA 4.4. *Using $m + pn$ processors, with high probability we can identify the connected components of the graph with $O(\log(\log n / \log p))$ walk phases.*

Proof. Assume for now that $p > \log^2 n$. The hypothesis gives us at least p processors per vertex. Running a walk phase of length p yields a graph of $O(n(\frac{\log^\alpha n}{p^\alpha}))$ vertices. On this graph redistribute the processors to get

$$\frac{np}{n \log^\alpha n / p^\alpha} = \frac{p^{1+\alpha}}{\log^\alpha n} > p^{9/8}$$

processors per vertex. Thus the number of processors per vertex after t walk phases is described with high probability by the recurrence

$$p_{t+1} > p_t^{9/8}$$

with solution

$$p_t > p^{(9/8)^t}.$$

Thus p_t exceeds pn within $O(\log(\log n/\log p))$ steps. Since this implies that all the processors are assigned to one vertex, the algorithm must be finished at this point. Therefore, this is the maximum expected number of walk phases needed.

There remains the detail of what to do if initially $p < \log^2 n$. To handle this case, note that even if $p = 1$, so that the random walks are in fact just inspections of a single neighbor, the neighborhoods still have size two. Thus the size of the graph is still reduced by a factor of two in each walk phase. Therefore, $O(\log(\log^2 n/p)) = O(\log(\log n/\log p))$ walk phases suffice to raise p to $\log^2 n$ and thus reduce to the previous case. □

Since each walk phase is simulated in $O(\log n)$ time, the overall running time of the algorithm is $O(\log n \log(\log n/\log p))$. Theorems 1.1 and 1.2 follow immediately, up to a factor of $\log n$ in the processor count that is removed in section 7.

5. Using fewer random bits. Randomness is used in our algorithm only to construct random walks. We show how to restrict this use of randomness to $O(n^\epsilon)$ bits for any $\epsilon > 0$. Note first that once we have n^ϵ processors per vertex and can simulate random walks of length n^ϵ, there is no need to reassign processors to vertices, since an additional $O(1/\epsilon)$ walk phases of length n^ϵ will finish the problem. Therefore, assume that random walks never exceed length n^ϵ. Now observe that a walk phase of length p needs only $p \log n$ random bits. Two entries in the *WALK* array need be independent only if it possible for a walk defined in the array to encounter both of them. Therefore, entries $WALK[i,t]$ and $WALK[i',t]$, $i \neq i'$, can use the same random seed in selecting edges, since a particular walk is only at one place at any particular time.

COROLLARY 5.1. *Connected components can be found in*

$$O(\log n \log(\log n/\log p) + (\log n)/\epsilon)$$

time using $m + pn$ processors and $O(n^\epsilon)$ random bits.

6. The deterministic version. Our techniques can also be used to obtain a deterministic algorithm for the EREW PRAM that runs in $O(\log^{1.5} n)$ time using $m + n$ processors. This improves on the deterministic $O(\log^{1.5} n)$-time algorithm of [NSW92] and matches an independent result of [JM92]. As in [NSW92], we use a universal sequence instead of a random walk. It will be convenient to consider a generalization of the universal sequences of [AKL*79] to allow walks on nonregular graphs.

DEFINITION 6.1. *A graph G with at most r vertices will be called* r-labeled *if the edges adjacent to each vertex are labeled with unique numbers from $\{1, 2, \ldots, r\}$.*

DEFINITION 6.2. *Given a string $\sigma \in \{1, 2, \ldots, r\}^*$ and an r-labeled graph G, a* walk according to σ *starting from a given vertex will follow an edge labeled i at step j if $\sigma_j = i$. If $\sigma_j = i$ and none of the edges leaving the current vertex are labeled i, the walk will remain in that vertex.*

DEFINITION 6.3. *A string $\sigma \in \{1, 2, \ldots, r\}^*$ is called an* r-universal sequence *if for every graph G with at most r vertices and any r-labeling of G a walk according to σ visits all the vertices of G regardless of the starting vertex.*

By following the proofs of [AKL*79], [BNS92], and [Nis92], it is not difficult to see that the construction of [Nis92] yields an r-universal sequence of length $r^{O(\log r)}$ in our general sense. We need only the following two properties.

THEOREM 6.4 (see [Nis92]). *An r-universal sequence of length $l = r^{O(\log r)}$ can be generated by an EREW PRAM in $O(\log l)$ time using $O(l \log l)$ processors.*

LEMMA 6.5. *For any undirected connected graph G with at least r vertices, and for any vertex v in G, a walk along an r-universal sequence σ starting from v visits at least r vertices of G.*

Proof. Label G so that each vertex of degree d is labeled with the numbers $\{1, 2, \ldots, d\}$. Assume the claim is false and σ visits fewer than r vertices.

Let C_r be the graph induced by all the vertices σ visits. Let $w \notin C_r$ be a vertex adjacent to some vertex $v' \in C_r$ such that the edge (v', w) is labeled with a number less than r (this is possible since v' has at most $r-2$ neighbors in C_r). Then the graph $C_r \cup w$ is an r-labeled graph with at most r vertices, and thus a walk according to σ should cover it and thus visit w, a contradiction. ∎

The deterministic algorithm proceeds as follows: instead of taking a random walk from each vertex, generate an r-universal sequence and then walk along this sequence. The parameter l is chosen such that the length of the resulting universal sequence is p (where p is the number of processors allotted to each vertex in the graph); thus $r = 2^{O(\sqrt{\log p})}$. We are thus assured by Lemma 2.1 that the number of vertices in the graph at the next round shrinks by a factor of at least r.

Letting p_i be the number of processors allotted to each vertex at iteration i, we argue as in the random walk case. We have the following recursion:

$$p_1 = 2,$$
$$p_{i+1} = p_i \cdot 2^{\sqrt{\log p_i}}.$$

LEMMA 6.6. *Let $p_1 = 2$ and $p_{i+1} = p_i \cdot 2^{\sqrt{\log p_i}}$. Then $p_j = n$ for some $j = O(\sqrt{\log n})$.*

Proof. Let $q_i = \log p_i$. Therefore, $q_1 = 1$ and $q_{i+1} = q_i + \sqrt{q_i}$. Then for every i, $q_{(i+\sqrt{q_i})} \geq 2q_i$. Thus the time to reach $q_j = \log n$ is at most $\sqrt{1} + \sqrt{2} + \sqrt{4} + \sqrt{8} + \cdots + \sqrt{\log n} = O(\sqrt{\log n})$. ∎

As a result of this lemma, we can conclude that after $O(\sqrt{\log n})$ walk phases the graph is contracted to a single vertex.

Theorem 1.3 follows immediately. Observe that starting with (polynomially many) more processors does not decrease the running time in this case.

COROLLARY 6.7. *If an n-universal sequence of polynomial length can be generated deterministically in $O(\log n)$ time, then connected components can be found in $O(\log n)$ time deterministically using $m + n^{1+\epsilon}$ processors for any fixed ϵ.*

7. Approaching optimal work. The algorithms described above perform work that exceeds the optimal by a factor of $O(\log n \log\log_p n)$. Here we reduce this factor to $O(\log\log_p n)$, showing how the $O(\log n \log\log_p n)$ running time can be achieved with $(m + pn)/\log n$ processors (this will be optimal for $p = n^\epsilon$). We begin with the assumption that $p > \log^2 n$ and later show how this assumption can be removed. Assume without loss of generality that $m \geq n/2$, since an initial step of the algorithm can use $n/\log n$ processors to remove any vertices with no edges.

7.1. Assuming $p > \log^2 n$. For now, assume that $p > \log^2 n$. Observe first that in this case the difference between using pn and $pn/\log n$ processors can be ignored, since for $p > \log^2 n$, $\log \frac{\log n}{\log p} = \Theta(\log \frac{\log n}{\log(p/\log n)})$. Thus the only need is to perform the m-processor steps with $m/\log n$ processors.

To do so, note that m processors are used for only one purpose: to update the edge list after leaders have been identified. There are three phases in this update process.

1. Replace the edge (i, j) by the edge $(\mathit{leader}(i), \mathit{leader}(j))$.
2. Detect and remove *dead* edges, namely, those that now have the form (i, i) because both endpoints chose the same leader.
3. Sort the remaining edges to remove duplicates and create edge lists for the contracted graph.

The real sticking point in this process is step 3. Since potentially nearly m edges may remain in the contracted graph, and since sorting them requires $\Omega(m \log m)$ work, it is unclear how step 3 can be performed.[3] Getting around this problem is the main topic of this section.

We begin by showing that steps 1 and 2 are easy to perform with $m/\log n$ processors. We allocate the processors according to the following scheme. Break the list of edges into sequential blocks of size $\log n$ and assign one processor to each block. Recall that the edge list is sorted by the first vertex in each edge. Therefore, the ith block contains first some of the edges of some first vertex f_i, then all the edges of some set of vertices V_i, and finally some of the edges of a last vertex l_i.

The advantage of this assignment is that it allows us to simulate, in $O(\log n)$ time, a single concurrent read by each edge (i, j) of some information from vertex i, and similarly, in $O(\log n)$ time, concurrent writes (with any conflict resolution scheme desired) by each edge (i, j) to vertex i. To simulate the read, proceed as follows. First use the standard concurrent read simulation to let processor i read from f_i and l_i into its local memory; these two reads take $O(\log n)$ time. Then each processor updates the $\log n$ edges it is responsible for—these updates now require only exclusive reads from its local copies of f_i and l_i or from the global values in the vertices V_i. The concurrent write simulation is similar.

Exploiting this simulation, we will freely use instructions of the form "each edge (i, j) reads from or writes to its vertex i," with the understanding that each such step actually takes $O(\log n)$ time.

One other small change is that it is necessary for each edge (i, j) to have a pointer to its *twin* edge (j, i) that is maintained as edges are moved around.

It is now easy to perform step 1 in $O(\log n)$ time—each edge (i, j) concurrently reads $\mathit{leader}(i)$ and replaces i by $\mathit{leader}(i)$ in (i, j) and in its twin (j, i). Step 2 can be performed easily by $m/\log n$ processors in $O(\log n)$ time with a standard array compaction algorithm.

It remains to deal with the difficulty of step 3. The approach we take is to ensure that the number of edges remaining after step 2 (counting duplications) is small, so that few processors are needed to perform step 3. We use the following lemma.

LEMMA 7.1. *If each edge of a graph is selected independently with probability q and connected components induced by the selected edges are contracted in the original graph, then with high probability the number of edges of the contracted graph is $O(n \ln n/q)$.*

In [KKT95], the number of remaining edges is shown to be $O(n/q)$ with high probability; this does not improve our application.

Proof. The number of edges in the contracted graph is just the number of edges crossing between the different connected components induced by the selected edges. The number of different arrangements of connected components is certainly no more than the number of ways to partition the set of n vertices into at most n groups, namely, n^n. For any given partition with k edges crossing between the components of the partition, the probability that no crossing edge is chosen is $(1 - q)^k \approx e^{-kq}$. The

[3] Possibly some form of bucket sort could be used to circumvent the sorting lower bound.

probability that k edges cross the partition resulting from the sampling construction is just the probability that for some partition with at least k crossing edges, no one of these k edges is chosen. This is at most $n^n e^{-kq} = e^{n \ln n - kq}$, which is negligible when $kq = \Omega(n \ln n)$. □

We therefore use the following approach: Given an m-edge graph, choose $m/\log n$ edges at random, and using $(m+pn)/\log n$ processors and the basic algorithm, compute connected components in this sampled graph in $O(\log n \log\log_p n)$ time. This labels all vertices in a given connected component of the sampled graph with a single vertex. If we treat the label of a vertex as its choice of a leader, then we can contract the original graph as if a walk phase has been performed. We use $m/\log n$ processors to relabel the edges and remove dead edges as was described at the beginning of this section. Lemma 7.1 shows that at this point $O(n \log^2 n)$ edges remain, so $O(n \log^2 n)$ processors suffice to perform step 3, sorting the edges and removing duplicates. We can finish the calculation by finding connected components in the resulting contracted graph; since the number of edges in the graph is $O(n \log^2 n)$, and since by assumption the number of processors is $pn > n \log^2 n$, this can be done in $O(\log n \log\log_p n)$ time with the available processors.

Thus when $p > \log^2 n$, connected components can be found in $O(\log n \log\log_p n)$ time using $O((m+pn)/\log n)$ processors.

7.2. Removing the assumption. We now handle the case $p < \log^2 n$. With such a value of p, the running time that we must achieve is $O(\log n \log\log n)$. Assume that in fact $p = 1$, since this merely restricts us further.

It suffices to find a procedure that, in $O(\log n)$ time and using $(m+n)/\log n$ processors, reduces the number of vertices by a constant factor. After $O(\log\log n)$ phases of this procedure, the graph will have $n' = O(n/\log^3 n)$ vertices. The algorithm of the previous section can be applied to solve this graph in $O(\log n \log\log n)$ time using $O(m/\log n + n' \log^2 n) = O((m+n)/\log n)$ processors.

We use the same allocation of processors to blocks of $\log n$ edges that was used previously, allowing the same simulation of concurrent reads and writes. The following procedure reduces the number of vertices in the graph by at least half in $O(\log n)$ time using $(m+n)/\log n$ processors.

1. For each vertex, compute the identity of its largest and smallest neighbors. To find the maximum, each edge (i,j) concurrently writes j to vertex i, letting multiple writes yield the maximum value written. Then do the same for the minimum value.
2. It is necessarily the case that either half the vertices have a larger neighbor or that half the vertices have a smaller neighbor. Which case we are in can be determined in $O(\log n)$ time using $n/\log n$ processors and the information from the previous step. Assume the first case; the other can be handled the same way.
3. Call vertices with no larger neighbors *leaders,* as before. The largest vertex j that is a neighbor of a nonleader vertex i becomes the *parent* of i as before. Mark the edge (i,j) in i's edge list, as well as its twin edge (j,i) in j's edge list.
4. Each vertex i can now identify its children: it examines its edges (i,j) and checks which ones were marked in the previous step.
5. Once each vertex has identified both parent and children, use the Euler tour technique to transform each tree of parent pointers into a list of the vertices in the tree and to identify the leader of each vertex.

6. Having identified leaders, relabel the edge list precisely as was done in section 7.1.
7. Since there is a linked list of vertices for each leader, and since each vertex has a contiguous list of its edges, we have an implicit linked list of all the edges that will be incident to a given leader after the graph is contracted. This allows us to use optimal list ranking to count the number of edges belonging to each leader in $O(\log n)$ time with $n/\log n$ processors and to rank them.
8. Take a length n array and write into the kth position the number of edges belonging to k if k is a leader, and 0 otherwise. Then, using array compaction, each leader can find the number of edges belonging to the leaders that precede it in the contracted graph.
9. The list ranking information just described suffices to determine the position each edge should be copied to in the edge array of the contracted graph, so the copying can be done in $O(\log n)$ time.

Note that duplicate edges do not affect this procedure, so we can ignore them until we reduce to the case of $p > \log^2 n$. Lemma 7.1, used to reduce to $m/\log n$ processors, holds even when duplicate edges exist. We do need to remove duplicates from the set of $m/\log n$ edges that we sample for the first application of the random walk algorithm, but the $m/\log n$ processors that we have are sufficient to do this by sorting.

8. Conclusion. Since the publication of the preliminary version of this paper [KNP92], Halperin and Zwick [HZ94] have used it to derive a work- and processor-optimal randomized EREW-connected components algorithm. The obvious remaining open problem is to find a deterministic $O(\log n)$-time EREW algorithm for connected components. One way to work toward this goal is to improve on the bounded space universal traversal sequence construction which is used in our deterministic algorithm, since any improvement in the space needed immediately yields a faster algorithm. There also remains an intriguing gap in the CRCW model, where the best-known algorithm has running time $O(\log n)$, but the best-known lower bound is $\Omega(\log n/\log\log n)$.

Acknowledgments. Thanks to Daphne Koller, who made our collaboration possible. Thanks also to Rajeev Motwani and Serge Plotkin for helpful discussions.

REFERENCES

[AKL*79] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovasz, and C. Rackoff, *Random walks, universal traversal sequences and the complexity of maze problems*, in Proc. 20th Annual Symposium on the Foundations of Computer Science, IEEE Computer Society Press, Piscataway, NJ, 1979, pp. 218–223.

[AS87] B. Awerbuch and Y. Shiloach, *New connectivity and MSF algorithms for shuffle-exchange network and PRAM*, IEEE Trans. Comput., C-36 (1987), pp. 1258–1263.

[BF93] G. Barnes and U. Feige, *Short random walks on graphs*, in Proc. 25th ACM Symposium on Theory of Computing, San Diego, ACM Press, New York, 1993, pp. 728–737.

[BNS92] L. Babai, N. Nisan, and M. Szegedy, *Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs*, J. Comput. System Sci., 45 (1992), pp. 204–232.

[BR91] G. Barnes, and W. L. Ruzzo, *Deterministic algorithms for undirected $s-t$ connectivity using polynomial time and sublinear space*, in Proc. 23rd ACM Symposium on Theory of Computing, New Orleans, ACM Press, New York, 1991, pp. 48–53.

[CDR86] S. Cook, C. Dwork, and R. Reischuk, *Upper and lower bounds for parallel random access machines without simultaneous writes*, SIAM J. Comput., 15 (1986), pp. 87–97.

[CL95] K. Chong and T. Lam, *Finding connected components in $O(\log n \log\log n)$ time on the EREW PRAM*, J. Algorithms, 18 (1995), pp. 378–402.

[CLC82] F. Y. Chin, J. Lam, and I. N. Chen, *Efficient parallel algorithms for some graph problems*, in Comm. ACM, 25 (1982), pp. 659–665.

[Col88] R. Cole, *Parallel merge-sort*, SIAM J. Comput., 17 (1988), pp. 770–785.

[CV91] R. Cole and U. Vishkin, *Approximate parallel scheduling.* II. *Applications to logarithmic-time optimal parallel graph algorithms*, Inform. and Comput. (formerly Information and Control), 92 (1991), pp. 1–47.

[DKR94] M. Dietzfelbinger, M. Kutylowski, and R. Reischuk, *Exact lower time bounds for computing Boolean functions on CREW PRAMs*, J. Comput. System Sci., 48 (1994), pp. 231–254; a preliminary version appeared in SPAA 1992.

[Gaz91] H. Gazit, *An optimal randomized parallel algorithm for finding the connected components of a graph*, SIAM J. Comput., 20 (1991), pp. 1046–1067; a preliminary version appeared in FOCS 1986.

[HCS79] D. S. Hirschberg, A. K. Chandra, and D. V. Sarwate, *Computing connected components on parallel computers*, Comm. ACM, 22 (1979), pp. 461–464.

[HZ94] S. Halperin and U. Zwick, *An optimal randomized logarithmic time connectivity algorithm for the EREW PRAM*, in Proc. 6th Annual ACM-SIAM Symposium on Parallel Algorithms and Architectures, ACM Press, New York, 1994, pp. 1–10.

[HZ6] S. Halperin and U. Zwick, *Optimal randomized EREW PRAM algorithms for finding spanning forests and other basic graph connectivity problems*, in Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM-SIAM, 1996, pp. 438–447.

[JM91] D. B. Johnson and P. Metaxas, *Connected components in $O(\log^{3/2} |V|)$ parallel time for the CREW PRAM*, in Proc. 32nd Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Piscataway, NJ, 1991, pp. 688–697.

[JM92] D. B. Johnson and P. Metaxas, *A parallel algorithm for computing minimum spanning trees*, in Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures, ACM Press, 1992, pp. 363–372.

[KKT95] D. R. Karger, P. N. Klein, and R. E. Tarjan, *A randomized linear-time algorithm to find minimum spanning trees*, J. ACM, 42 (1995), pp. 321–328.

[KNP92] D. R. Karger, N. Nisan, and M. Parnas, *Fast connected components algorithms for the EREW PRAM*, in Proc. 4th Annual ACM-SIAM Symposium on Parallel Algorithms and Architectures, 1992, pp. 562–572.

[KR90] R. M. Karp and V. Ramachandran, *Parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, Vol. A, Jan van Leeuwen, ed., MIT Press, Cambridge, MA, 1990, pp. 869–932.

[Lin] N. Linial, personal communication, 1992.

[Nis92] N. Nisan, *Pseudorandom generators for space-bounded computation*, in Combinatorica, 12 (1992), pp. 449–461; a preliminary version appeared in STOC 1990.

[Nis93] N. Nisan, *On read-once vs. multiple access to randomness in logspace*, Theoret. Comput. Sci., 107 (1993), pp. 135–144; a preliminary version appeared in Proc. 5th IEEE Structure in Complexity Theory Conference, 1990.

[NSW92] N. Nisan, E. Szemeredi, and A. Wigderson, *Undirected connectivity in $O(\log^{1.5} n)$ space*, in Proc. 33rd Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Piscataway, NJ, 1992, pp. 24–29.

[SV82] Y. Shiloach and U. Vishkin, *An $O(\log n)$ parallel connectivity algorithm*, J. Algorithms, 3 (1982), pp. 57–67.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 1035–1050

PRODUCTS AND HELP BITS IN DECISION TREES*

NOAM NISAN†, STEVEN RUDICH‡, AND MICHAEL SAKS§

Abstract. We investigate two problems concerning the complexity of evaluating a function f on k distinct inputs by k parallel decision-tree algorithms.

In the *product problem*, for some fixed depth bound d, we seek to maximize the fraction of input k-tuples for which all k decision trees are correct. Assume that for a single input to f, the best depth-d decision tree is correct on a fraction p of inputs. We prove that the maximum fraction of k-tuples on which k depth-d algorithms are all correct is at most p^k, which is the trivial lower bound. We show that if we replace the restriction to depth d by "expected depth d," then this result need not hold.

In the *help-bits problem*, before the decision-tree computations begin, up to $k-1$ *arbitrary* binary questions (help-bit queries) can be asked about the k-tuple of inputs. In the second stage, for each possible $(k-1)$-tuple of answers to the help-bit queries, there is a k-tuple of decision trees where the ith tree is supposed to correctly compute the value of the function on the ith input, for any input that is consistent with the help bits. The complexity here is the maximum depth of any of the trees in the algorithm. We show that for all k sufficiently large, this complexity is equal to $deg^s(f)$, which is the minimum degree of a multivariate polynomial whose sign is equal to f.

Key words. decision trees, help bits

AMS subject classifications. 68Q05, 68Q25, 68R99, 05D99

PII. S0097539795282444

1. Introduction.

Pick your favorite computation model and complexity measure, e.g., Boolean circuit size, communication complexity, decision-tree depth, interactive proof length, tensor rank, etc. Any attempt to understand such a model and complexity measure requires understanding the ways that an "unreasonable" computation can be more efficient than a "reasonable" one. Of course, what is reasonable changes as our understanding of the model improves.

Suppose we are given several unrelated instances of a problem to solve. The "reasonable" approach is to solve each instance separately; intuitively, any computation that is useful for solving one instance is irrelevant to any of the others. To what extent is this intuition valid in a given model? The following question is the most common way of formalizing this.

The direct-sum problem. Suppose that the complexity of computing some function f is c. Is it true that computing f twice, on two unrelated inputs, requires complexity $2c$? How about computing f on k unrelated inputs?

Versions of this question were first studied in the context of Boolean circuits [Ulig, Paul, GF]. Subsequent work has concerned bilinear circuits [JT, Bsh], Boolean circuits [FKN], communication complexity [KRW, KKN], and interactive proofs (see

*Received by the editors March 6, 1995; accepted for publication (in revised form) March 13, 1997; published electronically January 29, 1999. A preliminary version of this paper appeared in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, IEEE, 1994.
http://www.siam.org/journals/sicomp/28-3/28244.html

†Computer Science Department, Hebrew University, Jerusalem, Israel (noam@cs.huji.ac.il). This research was supported by BSF grant 92-00043 and by a Wolfson award administered by the Israeli Academy of Sciences.

‡Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213. This research was partially supported by NSF grant CCR-9119319.

§Department of Mathematics and RUTCOR, Rutgers University, New Brunswick, NJ 08903 (saks@math.rutgers.edu). This research was supported in part by NSF contracts CCR-9215293 and STC–91–19999, and by DIMACS.

the references contained in [Raz]). In this paper we consider two related problems of a similar flavor.

The product problem. Let f be a function, and suppose that for any allowable computation that has complexity bounded by c and attempts to compute f, the fraction of inputs on which it correctly computes f is at most p. Suppose that we have two independent computations, each taking as input the same ordered pair a, b of inputs to f, where the first computation is trying to compute $f(a)$ and the second is trying to compute $f(b)$. If each of the two computations has complexity at most c, can the fraction of input pairs a, b on which both are correct exceed p^2? What about the analogous question for k independent computations and k inputs?

If the first computation accesses only a and the second accesses only b, then the p^2 upper bound is trivial. Intuition suggests that there is no advantage in having each computation access the input of the other. A variant of this problem, in which we seek to compute f on the two inputs by a single computation, was studied recently in [IRW]. A version of this problem for interactive proofs, the well-known "parallel repetition problem," was recently solved by Raz [Raz].

The help-bit problem. Suppose that the complexity of exactly computing the Boolean function f is c. Suppose that we wish to compute f on two inputs a and b, and are allowed for free one "help bit," i.e., an arbitrary function of the two inputs. Is it possible to choose this help-bit function so that, given the help bit, $f(a)$ and $f(b)$ can each be evaluated by a computation of complexity less than c, and if so, how much can the complexity be reduced below c? How about computing f on k inputs with $k-1$ help bits?

The notion of help bits is essentially the same as that of bounded queries, which were studied in recursion theory [Be87]. The term "help bit" was introduced in the context of constant depth circuits in [Cai] and was also studied in the context of Boolean circuits in [ABG]. The point here is that if we have k inputs, then we can use k help bits to obtain the value of f on each of the inputs, and no further computation is necessary. With only $k-1$ help bits, we can for instance obtain the value of f at $k-1$ inputs, but then we still need complexity c to compute f on the last input. Is there a more effective use of the help bits?

In this paper we consider these problems in the context of Boolean decision-tree complexity—perhaps the simplest computational model. The cost of a computation (decision tree) is simply the number of input variables that are read (the depth of the decision tree); a more precise definition is given in section 2. While it is an easy exercise to see that "direct-sum" holds for decision-tree depth, the other two problems are more difficult. Our answer for the product problem is a qualified "Yes."

THEOREM 1.1. *Let f be a Boolean function and suppose that any depth-d decision tree computes f correctly on a fraction at most p of the inputs. Let $T_1, \ldots, T_k$ be decision trees that each access a set of nk variables corresponding to a k-tuple $\alpha^1, \ldots, \alpha^k$ of inputs to f. If each of the T_i has depth at most d, then the fraction of k-tuples $\alpha^1, \ldots, \alpha^k$ on which each T_i correctly outputs $f(\alpha^i)$ is at most p^k.*

The theorem seems completely obvious; however, readers might test their intuition on the following variation. Suppose that in the above theorem we change the complexity measure from "depth" to "average depth," i.e, the average over all inputs of the depth of the leaf reached by the input. This modified statement of the theorem seems similarly obvious but, as we will see, it is false.

The recent work of [IRW], which was done independently of ours, includes a (substantially different) proof of a weaker variant of this theorem, namely that a

single depth-d tree that tries to compute all k functions can be correct on at most a p^k fraction of the inputs. Our result shows that even if we use k parallel decision trees then we can't do better than this.

For the help-bit problem, the answer is more complicated. Linial [Lin] has shown that the complexity of computing f on two inputs with one help bit is at least $deg(f)$, the degree of the (unique) multilinear real polynomial that is equal to f. Since almost all Boolean functions on n variables have $deg(f) = n$, this says that for most functions one help bit does not help at all in evaluating two instances of the function (the case $k = 2$). It is natural to ask whether there are any functions for which one help bit helps.[1] Here, we show that for sufficiently large k, $k - 1$ help bits usually do help in computing a function on k different inputs. We manage to prove a lower bound that holds for all k, and is always tight when k, the number of instances to be solved, is sufficiently large. We need the following definitions. If f is an n-variate Boolean function, we say that the n variate real polynomial p *sign represents* f if for all inputs a, $f(a) = sgn(p(a))$, where $sgn(z) = 1$ if $z > 0$ and $sgn(z) = -1$ otherwise (here we are taking our Boolean set to be $\{-1, 1\}$). The *sign degree* of f, $deg^s(f)$, is the minimum degree of a polynomial that sign represents f. The sign degree of a function has been studied extensively in connection with threshold circuits and counting complexity classes. See the surveys [Sa93] and [Be93].

THEOREM 1.2. *Let f be an n-variate Boolean function. Then for all $k \geq 1$, any solution to the help-bit problem for f for k inputs and $k - 1$ help bits requires depth at least $deg^s(f)$. Furthermore, for all sufficiently large k, there is a decision-tree algorithm with $k - 1$ help bits whose depth is $deg^s(f)$.*

In the case that f is equal to the product of n variables (which corresponds to the parity function for $\{0, 1\}$-valued variables), $deg^s(f) = n$ and so the lower bound implies that help bits don't help in this case. Actually, this function and its negative are the only functions with $deg^s(f) = n$. Since the ordinary decision-tree complexity of most Boolean functions is n, this means that for large enough k, the complexity of k instances given $k - 1$ help bits is less than the ordinary decision-tree complexity for most functions. In particular, if f is the majority function, then $deg^s(f) = 1$, and the lower bound is trivial, while the upper bound says that for k sufficiently large, it is possible to ask $k - 1$ binary questions so that, given the answers, the value of the function on any one of the k inputs can be computed by probing just one variable. This remarkable savings is not typical; it has been observed by Anthony [Ant] and by N. Alon (personal communication) that almost all Boolean functions f satisfy $deg^s(f) \geq \lfloor n/2 \rfloor$.

In the next section, we review the decision-tree model. In section 3 we give a general formulation for the product problem in decision trees, and prove a generalization (Theorem 3.1) of Theorem 1.1. In section 4, we discuss the help-bit problem and prove Theorem 1.2.

2. Preliminaries. In this section we present some basic definitions and notation. Most of the notions discussed here are very familiar, but in some cases our notation is nonstandard.

2.1. Boolean functions. For the purposes of this paper it will be convenient to use $B = \{-1, 1\}$ as our Boolean set, instead of $\{0, 1\}$ (this choice of B is only significant in section 4). If X is a set, a *Boolean assignment* to X is a map α from X

[1] This question was recently answered in the negative by Beigel and Hirst [BH], who showed more generally that $\lfloor \log_2 k \rfloor$ help bits don't help for computing k functions.

to B. The set of Boolean assignments to X is denoted B^X. We refer to the elements of X as *variables*. We will consider probability distributions over the set of assignments. For a specified distribution D, a random assignment chosen according to D is denoted by placing a $\tilde{}$ above the identifier, e.g., $\tilde{\alpha}$. A *Boolean function* over the variable set X is a function with domain B^X. In this paper, the range of our functions will always be equal to B^k for some integer k.

2.2. Decision trees. All trees in this paper are rooted, ordered, binary trees. For such a tree T every internal node v has exactly two children, and the two children are distinguished as the (-1)-child and $(+1)$-child of v. The depth $d_T(v)$ of a node v is, as usual, the number of edges along the path from v to the root, and the depth d_T of T is the maximum depth of any node in T.

Formally, a *decision tree over* X is a triple (T, p, a), where T is a rooted, ordered, binary tree, p is a map that associates to each internal node v a variable $x = p_v$ in the set X, and a is a map that associates each leaf v with an element a_v of some set R. The label p_v is called the *query* associated with v, and node v is said to *probe* variable p_v. We will generally say that T is a decision tree, keeping the maps p and a implicit. The set of decision trees over X is denoted $\mathcal{T}(X)$, or simply $\mathcal{T}$.

Let T be a decision tree over X. If α is any assignment in B^X, the *computation* of T on α is the unique path $v^0, v^1, \ldots, v^s$ from the root of T to some leaf $v^s = l_T(\alpha)$ as follows: start from the root v^0 and inductively define v^{i+1} for $i \geq 0$ as the $\alpha(p_{v^i})$-child of v^i. The output of the computation is the label $a_{l_T(\alpha)}$. Thus T can be viewed as a Boolean function over X. Trivially, every function f over X is computed by some decision tree.

In the following definitions $\alpha \in B^X$, f is some function over X, T is a decision tree over X, $\mathcal{U}$ is a set of decision trees over X, d is a nonnegative integer, and D is a probability distribution over assignments to X.

- $C(T, \alpha)$, the *cost* of T on α, is the length (number of internal nodes) of the corresponding computation path.
- $C(T)$, the *complexity* or *depth* of T, is the maximum of $C(T, \beta)$ over all assignments β.
- $\mathcal{T}_d = \mathcal{T}_d(X)$ is the set of all decision trees over X of depth at most d.
- $C_D(T)$, the *distributional complexity* of T with respect to D, is the average of $C(T, \tilde{\alpha})$ with respect to D, i.e., $C_D(T) = \sum_{\alpha \in B^X} D(\alpha) C(T, \alpha)$.
- $q_D(f, T)$, the *agreement probability* of T with f relative to D, is the probability that $T(\tilde{\alpha}) = f(\tilde{\alpha})$, with respect to the random assignment $\tilde{\alpha}$ chosen according to D.
- The *decision tree approximation problem* for $(f, D, \mathcal{U})$ is to determine $q_D(f; \mathcal{U})$, which is defined to be the maximum agreement probability $q_D(f; T)$ over all $T \in \mathcal{U}$.
- T/β, the *contraction* of T by β where β is an assignment to some subset Y of X, is the decision tree on the variable set $X - Y$ that corresponds to executing T with the modification that whenever the current node v is labeled by $y \in Y$, follow the $\beta(y)$ branch without probing y. Formally, the tree T/β is obtained from T as follows: for each internal node v whose label p_v belongs to Y, replace the subtree rooted at v by the subtree rooted at the $\beta(p_v)$-child of v.
- A *decision forest* F over X is an ordered sequence $T_1, \ldots, T_k$, where T_i is a decision tree over X. F computes a Boolean function one short whose domain is B^X and whose range is $R = R_1 \times R_2 \times \cdots \times R_k$, where R_i is the range of

the function computed by T_i.

The following example illustrates some of the definitions.

Example 1. Consider the case of four variables, $X = \{x_1, x_2, x_3, x_4\}$, where $f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$. (Since our Boolean set is $\{-1, 1\}$ we treat -1 as the FALSE value here). Consider the distribution D on assignments in which the variables are assigned values independently, with x_1 and x_3 having probability p of being 1 and x_2 and x_4 having probability $1-p$ of being 1, where $p < 1/2$. Further, let $\mathcal{U}$ be the set $\mathcal{T}_2(X)$ of decision trees of depth at most 2. Let T be the following decision tree: First probe x_1. If $\alpha(x_1) = 1$, then probe x_2 and output $\alpha(x_2)$. Otherwise, if $\alpha(x_1) = -1$, probe x_3 and output $\alpha(x_3)$. Then this tree has depth 2, and it is easy to see that this algorithm will output $f(\alpha)$ unless $\alpha(x_1) = \alpha(x_3) = \alpha(x_4) = 1$ and $\alpha(x_2) = -1$, or $\alpha(x_1) = \alpha(x_4) = -1$ and $\alpha(x_3) = 1$. Thus the probability of agreement, $q_D(f; T)$, is $1-p^3(1-p)-p^2(1-p) = 1-p^2+p^4$. This turns out to be the largest agreement probability over all trees in $\mathcal{T}_2(X)$, so $q_D(f; \mathcal{T}_2(X)) = 1 - p^2 + p^4$.

3. The product problem. We now formalize the product problem stated in the introduction for the decision-tree model. Let $X_1, \ldots, X_k$ be pairwise-disjoint sets of variables, and let $D_1, \ldots, D_k$ be, respectively, distributions over assignments to $X_1, \ldots, X_k$. Let $X = X_1 \cup \cdots \cup X_k$. A Boolean assignment β for X will be viewed as a k-tuple $(\beta^1, \ldots, \beta^k)$, where β^i is an assignment for X_i. Let D denote the distribution over assignments to X given by $\mathrm{Prob}_D[\tilde{\alpha} = \beta] = \prod_{i=1}^{k} \mathrm{Prob}_{D_i}[\tilde{\alpha}^i = \beta^i]$, i.e., the product distribution $D_1 \times D_2 \times \cdots \times D_k$.

Now suppose that we have k decision-tree approximation problems $(f_1, D_1, \mathcal{U}_1), \ldots, (f_k, D_k, \mathcal{U}_k)$, where for each i, f_i is a function over X_i, and let $q_i = q_{D_i}(f_i; \mathcal{U}_i)$ be the optimal agreement probability for $\mathcal{U}_i$ with f_i relative to D_i. It will be convenient sometimes to view f_i as a function of the entire variable set X that does not depend on any variables except those in X_i. We consider the problem of *simultaneously approximating* $f_1, \ldots, f_k$ by a decision forest $F = (T_1, \ldots, T_k)$ where $T_i \in \mathcal{U}_i$. The *simultaneous agreement probability* $q_D(f_1, \ldots, f_k; T_1, \ldots, T_k)$ for $T_1, \ldots, T_k$ with $f_1, \ldots, f_k$ denotes the probability, for $\tilde{\alpha}$ chosen according to D, that $(T_1(\tilde{\alpha}) = f_1(\tilde{\alpha})) \wedge (T_2(\tilde{\alpha}) = f_2(\tilde{\alpha})) \wedge \cdots \wedge (T_k(\tilde{\alpha}) = f_k(\tilde{\alpha}))$. For $\mathcal{U}_1, \ldots, \mathcal{U}_k$ where $\mathcal{U}_i$ is a set of trees over X we define (see Example 2)

$$q_D(f_1, \ldots, f_k; \mathcal{U}_1, \ldots, \mathcal{U}_k) = \max\{q_D(f_1, \ldots, f_k; T_1, \ldots, T_k) : T_1 \in \mathcal{U}_1, \cdots, T_k \in \mathcal{U}_k\}.$$

Now, since f_i depends only on X_i, and since under D the assignments $\tilde{\alpha}^1, \ldots, \tilde{\alpha}^k$ to $X_1, \ldots, X_k$ are chosen independently, it would seem that $q_D(f_1, \ldots, f_k; T_1, \ldots, T_k)$ should just be the product of the probabilities $q_{D_i}(f_i; T_i)$. This is clearly the case if each tree T_i only queries variables in X_i. However (as shown by the examples below), if T_i is allowed to query variables outside of X_i, then this need not be the case. Intuitively, it would seem that variables outside of X_i could not help to approximate f_i and indeed this is trivially true, if we are only trying to approximate f_i. But when we seek to approximate all of the functions simultaneously, it is no longer true that such "cross-queries" are irrelevant.

Nevertheless, we might expect that for "reasonable" classes $\mathcal{U}_1, \ldots, \mathcal{U}_k$ of decision trees, the optimal simultaneous agreement probability is attained by a sequence of trees $T_1, \ldots, T_k$ with T_i querying variables only in X_i, and is thus equal to the product of the individual optimal agreement probabilities. The main result of this section is to prove this in the case that for each i, $\mathcal{U}_i$ is the set of trees of some fixed depth d_i.

THEOREM 3.1. *Let $f_1, \ldots, f_k$ and $D_1, \ldots, D_k, D$ be as above. Let $d_1, \ldots, d_k$ be nonnegative integers. Then*

$$q_D(f_1, \ldots, f_k; \mathcal{T}_{d_1}, \ldots, \mathcal{T}_{d_k}) = \prod_{i=1}^{k} q_{D_i}(f_i, \mathcal{T}_{d_i}).$$

Note that Theorem 1.1 is a special case of the above. Before giving the proof we present two examples to show that multiplicativity fails for some natural alternative choices of the classes $\mathcal{U}_1, \ldots, \mathcal{U}_k$.

Example 2. Theorem 3.1 fails if we replace the class $\mathcal{T}_{d_i}$ by the class $\mathcal{S}^i_{d_i}$ of trees that are restricted to query at most d_i variables from X_i along any path but can query any number of variables outside X_i. Consider the following trivial example. Let $k = 2$ and let $X_1 = \{x_1\}, X_2 = \{x_2\}$. The distribution D_1 assigns x_1 to 1 with probability $1/2$, and D_2 assigns x_2 to 1 with probability $1/2$, so $\tilde{\alpha}$ is uniformly distributed on B^2. The functions f_1 and f_2 are given by $f_1(x_1) = x_1, f_2(x_2) = x_2$. Now let $d_1 = d_2 = 0$. This means that we do not allow T_1 to look at any variables in X_1 and we do not allow T_2 to look at any variables in X_2. Clearly $q_{D_1}(f_1, \mathcal{S}^1_0) = q_{D_2}(f_2, \mathcal{S}^2_0) = 1/2$. However, we can achieve simultaneous agreement probability better than $1/4$. Let T_1 be the tree that queries x_2 and outputs $\tilde{\alpha}(x_2)$ and T_2 be the tree that queries x_1 and outputs $\tilde{\alpha}(x_1)$. Then, the probability that both T_1 and f_1 agree and T_2 and f_2 agree is just the probability that x_1 and x_2 are assigned the same value, which is $1/2$.

A somewhat more subtle example is the following.

Example 3. For a distribution D over B^X, let $\mathcal{T}^D_d$ be the class of trees whose expected depth with respect to D is d, i.e., $T \in \mathcal{T}^D_d$ if the average number of variables queried with respect to $\tilde{\alpha}$ chosen from D is at most d. Then the above theorem becomes false if we replace $\mathcal{T}_{d_i}$ with $\mathcal{T}^{D_i}_{d_i}$. To see this, let X be a set of four variables, and f be the product of the variables. Let U be the uniform distribution over assignments to X and let $d = 3$. First we show that the maximum agreement probability with f attained by a decision tree S of expected depth at most 3 is equal to $3/4$. Agreement probability $3/4$ is attained by the tree S that queries a particular variable x, and if it is -1, then it returns -1, and otherwise it queries the remaining three variables and returns their product. To see that this is the best possible, note that if T is any decision tree, then for each leaf l in T of depth less than 4, T will agree with f on exactly half of the inputs that reach l. Thus, if p_i is the probability that a random input $\tilde{\alpha}$ ends up at a leaf of depth i, then the agreement probability $q_D(f; T)$ can be bounded above by $p_4 + 1/2(1 - p_4)$; it suffices to show that $p_4 \leq 1/2$. Now p_1 either equals 0, $1/2$, or 1. If $p_1 > 0$ then $p_4 \leq 1/2$. If $p_1 = 0$, then the expected depth of the tree is at least $4p_4 + 2(1 - p_4) = 2 + 2p_4$, which means that $p_4 \leq 1/2$.

Now let X_1, f_1, D_1 and X_2, f_2, D_2 be copies of X, f, U on disjoint variable sets. We show that it is possible to choose decision trees T_1, T_2, each of expected depth at most 3, whose agreement probability exceeds $9/16 = (3/4)^2$. Let T_1 be the S described above and let x_1 denote the variable in X_1 probed first by T_1. Let T_2 be the following tree: first probe x_1 (in X_1). If it is -1, output -1. If it is $+1$, then probe all four variables in X_2 and output their product. The expected depth of this tree is 3, since half the paths have depth 1 and half the paths have depth 5. Now, let us consider the probability of the event A that both $T_1(\tilde{\alpha}) = f_1(\tilde{\alpha})$ and $T_2(\tilde{\alpha}) = T_2(\tilde{\alpha})$:

$$\text{Prob}_D[A] = \frac{1}{2}\text{Prob}_D[A \mid \tilde{\alpha}(x_1) = -1] + \frac{1}{2}\text{Prob}[A \mid \tilde{\alpha}(x_1) = 1].$$

The conditional probability of A given $\tilde{\alpha}(x_1) = -1$ is $1/4$ and the conditional probability of A given $\tilde{\alpha}(x_1) = 1$ is 1. Thus the probability of simultaneous agreement is $5/8$.

What happens in the above example is that the variable x_1 acts as a shared random coin that partially coordinates the two computations so that they are more likely to be simultaneously correct. On the face of it, it seems quite possible that a similar trick might also help for the class of trees covered by the theorem; but as we now show, no such trick can work in this case.

Proof of Theorem 3.1. Fix a sequence $T_1, \ldots, T_k$ of decision trees with T_i of depth at most d_i, and let $\tilde{\alpha} = (\tilde{\alpha}^1, \ldots, \tilde{\alpha}^k)$ be a random assignment to $X = X_1 \cup \cdots \cup X_k$ chosen according to the distribution D. For $I \subseteq [k] = \{1, \ldots, k\}$, let $C(I)$ denote the event $\bigwedge_{i \in I}(T_i(\alpha) = f_i(\alpha^i))$, i.e., the event that all of the trees indexed by I evaluate their respective functions correctly. We seek to prove that $\mathrm{Prob}[C([k])]$ is bounded above by $\prod_{i=1}^{k} q_{D_i}(f_i, \mathcal{T}_{d_i})$.

The proof is by induction on k, and for fixed k by induction on $d_1 + \cdots + d_k$. The result is trivial if $k = 1$.

So assume that $k \geq 2$. Consider first the case that $d_i = 0$ for some i. We may assume that $d_k = 0$. Thus, the kth computation must guess the value of $f_k(\tilde{\alpha}^k)$ without looking at any variables, so T_k consists of a single leaf labeled -1 or 1; i.e., $T_k(\alpha)$ is a constant $t \in B$. Now, by conditioning on the value of the vector $\tilde{\alpha}^k$, the probability, P^*, that $C([k])$ holds can be bounded above:

$$\begin{aligned}
P^* &\leq \sum_{\beta^k \in B^{X_k} \mid f_k(\beta^k) = t} \mathrm{Prob}[\tilde{\alpha}^k = \beta^k] \times \mathrm{Prob}[C([k-1]) \mid \tilde{\alpha}^k = \beta^k] \\
&\leq \max_{\beta^k \in B^{X_k}} \mathrm{Prob}[C([k-1]) \mid \tilde{\alpha}^k = \beta^k] \times \sum_{\beta^k \in B^{X_k} \mid f_k(\beta^k) = t} \mathrm{Prob}[\tilde{\alpha}^k = \beta^k] \\
&= \max_{\beta^k \in B^{X_k}} \mathrm{Prob}[C([k-1]) \mid \tilde{\alpha}^k = \beta^k] \times \mathrm{Prob}[T_k(\tilde{\alpha}^k) = f_k(\tilde{\alpha}^k)] \\
&\leq \max_{\beta^k \in B^{X_k}} \mathrm{Prob}[C([k-1]) \mid \tilde{\alpha}^k = \beta^k] \times q_{D_k}(f_k, \mathcal{T}_0).
\end{aligned}$$

Now let γ be the value of β^k that maximizes the probability in the last expression. For each i between 1 and $k-1$, define the tree U_i by contracting T_i using $\tilde{\alpha}^k = \gamma$. Then we may rewrite the last term as

$$\mathrm{Prob}[(U_1(\tilde{\alpha}) = f_1(\tilde{\alpha})) \wedge \cdots \wedge (U_{k-1}(\tilde{\alpha}) = f_{k-1}(\tilde{\alpha}))] \times q_{D_k}(f_k, \mathcal{T}_0).$$

Each tree U_i has depth at most d_i, and so we may bound the first factor by $q_D(f_1, \ldots, f_{k-1}; \mathcal{T}_{d_1}, \ldots, \mathcal{T}_{d_{k-1}})$, which is equal to $\prod_{i=1}^{k-1} q_{D_i}(f_i, \mathcal{T}_{d_i})$, by the induction hypothesis. Thus the desired result follows.

Now we assume that $d_i > 0$ for all i. Define a directed graph on $\{1, \ldots, k\}$ with an edge from i to j if the first variable probed by T_i is an input to f_j. Since this directed graph has out-degree one, it has a directed cycle, which may be a self-loop. Let $j \geq 1$ be the length of the cycle. Let us reindex the trees so that the vertices of the cycle in order are $1, \ldots, j$. Thus for each $i < j$, the first probe of T_i is a variable in X_{i+1} (which we will denote x_{i+1}) and the first probe of T_j is a variable, denoted x_1, in X_1. The case $j = 1$ does not have to be treated as a special case.

In the rest of the proof, we analyze the probability of simultaneous agreement of the trees with their corresponding functions by conditioning on the assignments to $x_1, \ldots, x_j$. This will enable us to apply the induction hypothesis and prove the theorem. For the case $j = 1$ this is easy; for $j > 1$ the underlying intuition is that it

is possible simultaneously to replace each of the trees T_i for $i \in [j]$, by trees T_i' of the same depth in which the first probe in T_i' is x_i, without decreasing the probability of simultaneous agreement.

For $b \in B$, let f_i^b denote the function obtained from f_i by fixing $x_i = b$. Also, let D_i^b be the distribution on the set $X_i - x_i$ obtained from D_i by conditioning on $x_i = b$.

Now, for $\mathbf{b} = (b_1, \ldots, b_j) \in B^j$, let $A(\mathbf{b})$ denote the event that $(\tilde{\alpha}(x_1) = b_1) \wedge \cdots \wedge (\tilde{\alpha}(x_j) = b_j)$. We can write the probability that all of the T_i compute correctly by conditioning on $\mathbf{b}$ as follows:

$$\sum_{\mathbf{b} \in B^{[j]}} \text{Prob}[A(\mathbf{b})]\text{Prob}[C([k]) \mid A(\mathbf{b})]. \tag{1}$$

We seek to upper-bound this expression by

$$\prod_{i=1}^{k} q_{D_i}(f_i, \mathcal{T}_{d_i}). \tag{2}$$

To do this we show the following.

Claim. For each $\mathbf{b} \in B^{[j]}$, the conditional probability of $C([k])$ given $A(\mathbf{b})$ is at most

$$\left(\prod_{i=1}^{j} q_{D_i^{b_i}}(f_i^{b_i}, \mathcal{T}_{d_i - 1})\right)\left(\prod_{i=j+1}^{k} q_{D_i}(f_i, \mathcal{T}_{d_i})\right).$$

Assuming the claim for the moment, we can then substitute into (1) to obtain the following bound on the probability that all of the trees are correct:

$$\left(\prod_{i=j+1}^{k} q_{D_i}(f_i, \mathcal{T}_{d_i})\right)\left(\sum_{\mathbf{b} \in B^j} \text{Prob}[A(\mathbf{b})] \prod_{i=1}^{j} q_{D_i^{b_i}}(f_i^{b_i}, \mathcal{T}_{d_i - 1})\right). \tag{3}$$

The sum can be rewritten as

$$\sum_{\mathbf{b} \in B^j} \prod_{i=1}^{j} \text{Prob}[\tilde{\alpha}(x_i) = b_i] q_{D_i^{b_i}}(f_i^{b_i}, \mathcal{T}_{d_i - 1}),$$

which is equal to

$$\prod_{i=1}^{j} \left(\text{Prob}[\tilde{\alpha}(x_i) = -1] q_{D_i^{-1}}(f_i^{-1}, \mathcal{T}_{d_i - 1}) \quad + \quad \text{Prob}[\tilde{\alpha}(x_i) = 1] q_{D_i^{1}}(f_i^{1}, \mathcal{T}_{d_i - 1})\right).$$

Now, the ith term in this product corresponds to the probability of correctly computing f_i if we first probe x_i and then, depending on the outcome, use the optimal depth $d_i - 1$ tree to evaluate the residual function. Thus, we can upper-bound this term by $q_{D_i}(f_i, \mathcal{T}_{d_i})$. But then (3) is upper-bounded by (2) as required.

So it suffices to prove the claim. Define $f_i^{A(\mathbf{b})}$ to be the function $f_i^{b_i}$ for $i \leq j$ and to be f_i otherwise. Similarly, the distribution $D_i^{A(\mathbf{b})}$ is equal to $D_i^{b_i}$ for $i \leq j$ and to D_i otherwise. Observe that by the mutual independence of $\tilde{\alpha}^1, \ldots, \tilde{\alpha}^k$, their joint distribution given $A(\mathbf{b})$ is the product distribution of $D_i^{A(\mathbf{b})}$ for i between 1 and k.

Let $T_i^{A(\mathbf{b})}$ be the tree obtained by contracting T_i under the assumption that $A(\mathbf{b})$ holds. Then the conditional probability given $A(\mathbf{b})$ that $T_i(\alpha) = f_i(\alpha^i)$ for all i, is equal to the probability (with respect to the product distribution on $D_i^{A(\mathbf{b})}$) that for all i, $T_i^{A(\mathbf{b})} = f_i^{A(\mathbf{b})}$. Now for each i the depth of $T_i^{A(\mathbf{b})}$ is at most $d_i - 1$ if $i \leq j$, and is at most d_i for $i > j$, so we may apply induction to say that the probability, with respect to the product distribution on $D_i^{A(\mathbf{b})}$, that $T_i^{A(\mathbf{b})} = f_i^{A(\mathbf{b})}$ for every i, is at most

$$\left(\prod_{i=1}^{j} q_{D_i^{A(\mathbf{b})}}(f_i^{A(\mathbf{b})}, \mathcal{T}_{d_i-1})\right)\left(\prod_{i=j+1}^{k} q_{D_i^{A(\mathbf{b})}}(f_i^{A(\mathbf{b})}, \mathcal{T}_{d_i})\right),$$

which is equal to the expression in the claim. This proves both the claim and the theorem. □

Remark 1. The proof of the theorem can be extended to a more general model of decision-tree computation. For this model, in the case of a single function we are given a function f on an arbitrary domain S, and want to compute $f(s)$ for an unknown input $s \in S$. We are further given a set Q of admissible queries, where each query $q \in Q$ is a partition of S into sets $(S_1^q, \ldots S_r^q)$. The response to query q is the index i such that $s \in S_i^q$. The nodes of a decision tree are labeled by queries, and the branches out of the node correspond to the answers to the query. For a set of functions f_i on disjoint domains S_i, the formulation of the product problem generalizes to this model. The statement and proof of the theorem now go through assuming that (1) each allowed query depends only on variables from one function and (2) the distributions D_i are independent.

4. Help bits. The help-bits model can be formulated with respect to any model of computation. Informally, we have a function g which we wish to compute using an algorithm from a given class of algorithms. Before we begin computing, we are allowed to ask a certain number of *arbitrary* binary questions about the input. Based on the answers to these questions we choose an algorithm and then do the computation on the input. The answers to the preliminary questions are referred to as "help bits." We say that g is computable by a class of algorithms using h help bits if there is such a procedure that uses at most h questions and correctly evaluates g on all inputs.

As was observed previously in specific computational contexts [Be87, CH], the problem of whether g can be computed with h help bits can be recast as a cover problem. Given g and a class of algorithms, we wish to choose a set of algorithms from the class that covers all inputs in the sense that for each input x, there is at least one algorithm in the chosen set that outputs $g(x)$. We refer to such a set of algorithms as a *cover* of g. Then g is computable by the class of algorithms using h help bits if and only if there is a cover of g of size 2^h. To see this, suppose that g is computable by the class using h help bits. Then each of the 2^h possible sets of answers to the questions specifies an algorithm in the class, and these algorithms together are a cover of g. Conversely, given a cover of size 2^h, index the algorithms in the cover by Boolean strings of length h, and for each input x choose a Boolean string b_x that indexes an algorithm $A(b_x)$ in the cover that correctly evaluates $g(x)$. Then we may take our help-bit queries to be, "What are the bits of b_x?"

Note that, provided that the class of algorithms includes all constant functions, any function g whose range has size at most 2^h is trivially computable using h help bits, so the problem is only interesting for functions whose range is "large enough."

In this paper we are interested in the special case of the help-bits problem where the function we are computing is a Cartesian product $\bar{f}$ of Boolean functions $f_1, \ldots, f_k$ over disjoint variable sets $X_1, \ldots, X_k$. The function $\bar{f}$ has variable set $X = X_1 \cup \cdots \cup X_k$, and its value on assigment $\alpha = (\alpha_1, \ldots, \alpha_k)$ is the k-tuple $f_1(\alpha^1), \ldots, f_k(\alpha^k)$. Given an unknown assignment α to X we want to evaluate $\bar{f}(\alpha) = (f_1(\alpha^1), \ldots, f_k(\alpha^k))$ by a decision forest. We are allowed to ask, "for free," an *arbitrary* set of h binary questions about the assignment α. The answer to these h questions is a vector $\mathbf{a} \in B^h$. For each such $\mathbf{a}$ we will have a decision forest $F^{\mathbf{a}} = (T_1^{\mathbf{a}}, \ldots, T_k^{\mathbf{a}})$, where we require that $F^{\mathbf{a}}(\alpha)$ agrees with $\bar{f}(\alpha)$ for every assignment α that is consistent with $\mathbf{a}$. Note that, unlike the product problem, we are now dealing with worst case complexity of exact computation.

An algorithm in this model is specified by l arbitrary Boolean functions $h_1, \ldots, h_l$ (the help-bit functions) on variable set X, together with 2^l decision forests, which are indexed by the possible outputs of $h_1, \ldots, h_l$. The depth of the algorithm is the maximum depth of any of the $2^l k$ decision trees in these forests. In general, the decision tree $T_i^{\mathbf{a}}$ that computes $f_i(\alpha^i)$ for α consistent with $\mathbf{a}$ is allowed to probe variables outside of X_i. This is conceivably useful, because together with the help bits, such probes could imply information about the variables in X_i. For instance, if one of the help-bit functions is $(f_i(\alpha^i) \times \alpha^j(x))$, where x is a variable in X_j, then by probing the variable x, we can deduce $f_i(\alpha^i)$. If $T_i^{\mathbf{a}}$ only probes variables in X_i we say that it is *pure*. If each of the $2^l k$ decision trees is pure, the algorithm is pure. We denote by $C_h(\bar{f})$ the minimum depth (complexity) of any algorithm that computes $\bar{f}$ using h help bits and by $C_h^{pure}(\bar{f})$ the minimum depth of a pure algorithm that computes $\bar{f}$.

Our main result applies to the case that, for some variable set X and Boolean function f over X, each of the X_i are copies of X and the functions f_i are copies of f, and in this case we denote the k-tuple $\bar{f}$ by $f^{[k]}$. It is not hard to generalize this and formulate similar results for the case that the f_i are arbitrary functions on arbitrary disjoint sets of variables. When convenient we state the lemmas below for this general case. The main result of this section (which is a slight refinement of Theorem 1.2), is the following theorem.

THEOREM 4.1. *For any Boolean function f on n variables and any positive integer k,*

$$C_{k-1}^{pure}(f^{[k]}) \geq C_{k-1}(f^{[k]}) \geq deg^s(f).$$

If k is sufficiently large, then

$$C_{k-1}^{pure}(f^{[k]}) = C_{k-1}(f^{[k]}) = deg^s(f).$$

Before proving the theorem, we mention an example. As noted in the introduction, if f is the majority function, then $deg^s(f) = 1$, and so for this function and any sufficiently large integer k, the theorem asserts that given $k-1$ help bits it is possible to compute $f^{[k]}$ with depth 1. M. Blum pointed out to us explicitly how to do this for the case $n = 3$. Enumerate the subsets of $\{1, \ldots, k\}$ having size at least $2k/3$. The number of these sets is 2^{ck} for some $c < 1$. Fix an encoding of these sets by ck bits. Now given k assignments to the variables of f imagine the assignments arranged in a $k \times 3$ array. In each row, at least two of the three entries agree with the majority value, so there is a column in which at least $2k/3$ of the entries agree with the function value on that row. For the help bits, we ask for the lowest index of such

a column (requiring two bits) and then for the set S of rows for which this column gives the function value (requiring ck bits). Armed with this information, the value of the function on the assignment indicated by row r is equal to the entry in that row and the designated column if $r \in S$, and is the negative of the entry otherwise. Thus the decision tree for evaluating f on the rth assignment need only probe that one variable.

To prove Theorem 4.1, it will be useful to "invert" our point of view and fix d and consider the minimum number of help bits required to compute a given function. Define $H_d(\bar{f})$ (resp. $H_d^{pure}(\bar{f})$) to be the minimum number of help bits needed to compute $\bar{f}$ using a depth-d algorithm (resp. pure depth-d algorithm).

We will prove the following theorem.

THEOREM 4.2. *Let f be a Boolean function on n variables.*

1. *If $d < deg^s(f)$, then for all k,*

$$H_d(f^{[k]}) = H_d^{pure}(f^{[k]}) = k.$$

2. *There exists a positive constant δ_f such that for $d = deg^s(f)$,*

$$H_d^{pure}(f^{[k]}) = (1 - \delta_f)k + o(k).$$

The first part of Theorem 4.2 easily implies the lower bound on $C_{k-1}(f^{[k]})$ (and hence on $C_{k-1}^{pure}(f^{[k]})$) in Theorem 4.1. Also, the second part of Theorem 4.2 implies that for sufficiently large k, $H_{deg^s(f)}^{pure}(f^{[k]}) \leq k - 1$, which implies $C_{k-1}^{pure}(f^{[k]}) \leq deg^s(f)$, which is the second part of Theorem 4.1.

Thus it suffices to prove Theorem 4.2. We can reinterpret $H_d(\bar{f})$ and $H_d^{pure}(\bar{f})$ in terms of the notion of cover given in the beginning of this section. Say that a set $\mathcal{F}$ of decision forests covers all assignments with respect to $\bar{f}$ if for each α there is an $F \in \mathcal{F}$ such that $F(\alpha) = \bar{f}(\alpha)$. Define $\tau_d(\bar{f})$ to be the minimum size of a cover that consists of forests of depth d. Then the remarks at the beginning of this section imply the following proposition.

PROPOSITION 4.1. *Let $\bar{f} = (f_1, \ldots, f_k)$ and d be a nonnegative integer. Then*

1. $H_d(\bar{f}) = \lceil \log \tau_d(\bar{f}) \rceil$,
2. $H_d^{pure}(\bar{f}) = \lceil \log \tau_d^{pure}(\bar{f}) \rceil$.

So we now concentrate on obtaining bounds on $\tau_d(f^{[k]})$ and $\tau_d^{pure}(f^{[k]})$. For this we need yet another definition. A *randomized decision tree over X* is a probability distribution Q on the set of decision trees over X. A randomized decision tree is said to *approximate f with probability p* if for each assignment α, if $\tilde{T}$ is chosen according to Q, the probability that $\tilde{T}(\alpha) = f(\alpha)$ is at least p. We define $p_d(f)$ to be the maximum p such that there is a distribution Q over the set of decision trees of depth at most d that approximates f with probability p. That this maximum exists follows by noting first that we may assume that Q assigns nonzero probability only to trees whose range is contained in the range of f. Let N be the number of such trees (which is finite). Then the set of all such distributions can be viewed as a subset of $\mathbf{R}^N$, and this subset is compact. For fixed f the probability that $\tilde{T}(\alpha) = f(\alpha)$ can be viewed as a continuous function of the "vector" Q. Thus we have a continuous function on a compact set, which must attain a maximum. It is easy to see that $p_d(f) \geq 1/2$ and that if d is equal to $C(f)$, the ordinary decision-tree complexity of f, then $p_d(f) = 1$. Intuitively, the quantity $p_d(f)$ represents the best agreement probability we can guarantee (with respect to worst case input) by choosing an appropriate randomized decision tree. A fundamental observation of Yao [Y2] (which was originally made in a slightly different

context) says that we can choose a probability distribution on inputs such that for any fixed decision tree (or randomized decision tree) the agreement probability with respect to that distribution does not exceed $p_d(f)$. This observation follows from the min-max theorem for two-person zero-sum games (matrix games), and variants of it hold in most nonuniform complexity models. To formalize this observation in our context, recall from the previous section that for a distribution D over assignments, a function f, and a class $\mathcal{T}$ of decision trees, we defined the agreement probability $q_D(f, \mathcal{T})$ to be the maximum over all trees $T \in \mathcal{T}$ of the probability that T and f agree on a random assignment chosen according to D.

LEMMA 4.1. *For any Boolean function f over X and integer $d \geq 0$, there exists a distribution $\hat{D}$ on assignments to X such that $q_{\hat{D}}(f, \mathcal{T}_d) = p_d(f)$.*

Proof. We need first to recall the min-max theorem for two-person zero-sum games. Let M be a $r \times s$ matrix with real entries. A stochastic vector is a nonnegative real-valued vector with entries summing to 1. If v is a vector, write $\max(v)$ for the maximum entry of v and $\min(v)$ for the minimum entry of v. In its simplest form the min-max theorem says that if v ranges over all stochastic vectors of length r then $\min(vM)$ attains a maximum V_1, and if w ranges over all stochastic vectors of length s then $\max(Mw)$ attains a minimum V_2, and $V_1 = V_2$.

To apply the min-max theorem in this case, let M be the matrix whose rows are indexed by the set $\mathcal{T}_d$ and whose columns are indexed by the set B^X of assignments, and whose entry in position (T, α) is 1 if $T(\alpha) = f(\alpha)$ and 0 otherwise. A stochastic vector v indexed by $\mathcal{T}_d$ corresponds to a randomized decision tree, and $\min(vM)$ represents the minimum over all assignments α of the probability that this randomized decision tree computes f correctly on all assignments. Taking the maximum of this over all such v, we get $V_1 = p_d(f)$. Similarly, a stochastic vector indexed by B^X corresponds to a probability distribution over all assignments, and thus V_2 is equal to the minimum over all such distributions D of $q_D(f, \mathcal{T}_d)$. Applying the min-max theorem we get that the distribution $\hat{D}$ that attains this minimum satisfies the conclusion of the lemma. □

Returning to the problem of bounding $\tau_d(f^{[k]})$, we now prove the following lemma.

LEMMA 4.2. *For any Boolean function f on n variables and $k, d \geq 0$, we have*

$$\frac{1}{p_d(f)^k} \leq \tau_d(f^{[k]}) \leq \tau_d^{pure}(f^{[k]}) \leq \left\lceil \frac{nk \ln 2}{p_d(f)^k} \right\rceil.$$

Proof. The middle inequality is trivial. For the last inequality, we use a standard probabilistic argument to show that there is a set of at most $\lceil \frac{nk \ln 2}{p_d(f)} \rceil$ pure forests of depth at most d that cover all of the assignments. Let Q be the distribution on decision trees over X of depth at most d that approximates f with probability $p_d(f)$. For $i \leq k$, let Q_i be the corresponding distribution over the set of decision trees over X. Then Q_i approximates f_i with probability $p_d(f)$. Consider the distribution $P = Q_1 \times \cdots \times Q_k$ over forests. Suppose we select t forests $\tilde{F}_1, \ldots, \tilde{F}_t$ according to P. For a given assignment α and $j \leq t$, the probability that $\tilde{F}_j$ covers α is at least $p_d(f)^k$. Thus the probability that none of the forests covers α is at most $(1 - p_d(f)^k)^t$, and the probability that there exists an assignment α that is covered by none of the forests is at most $2^{nk}(1 - p_d(f)^k)^t < 2^{nk} e^{-p_d(f)^k t}$. If $t = \lceil (nk \ln 2)/p_d(f)^k \rceil$, then this last expression is at most 1, so there is a positive probability that the forest covers all assignments, and so there must be a set of t forests of depth d that cover all assignments.

Now we turn to the lower bound on $\tau_d(f^{[k]})$. Let $\hat{D}$ be the distribution whose existence is asserted by Lemma 4.1. Suppose that $F_1, \ldots, F_t$ is a set of forests that cover all assignments α to X. Consider the distribution P over all assignments α which is the product $\hat{D}_1 \times \cdots \times \hat{D}_k$, where $\hat{D}_i$ is the copy of $\hat{D}$ on X_i. Then, by Theorem 3.1, for any forest F_i, the probability that it covers $\tilde{\alpha}$ is at most $p_d(f)^k$. Then the expected fraction of assignments covered by $F_1, \ldots, F_t$ is at most $tp_d(f)^k$. Since $F_1, \ldots, F_k$ covers all assignments, this expectation must be at least 1, so $t \geq 1/p_d(f)^k$. □

It is worth noting that the above lemma holds (with essentially the same proof) in the general setting described at the beginning of this section; we can replace $\mathcal{T}_d$ by any set of Boolean functions on variable set X and define analogues to $p_d(f)$ and $\tau_d(f)$.

As an immediate corollary of Lemma 4.2 and Proposition 4.1 we get the following bounds on the complexity of the help-bits problem.

COROLLARY 4.1. *For any Boolean function f on n variables and integers $k, l, d \geq 0$:*

$$k(-\log_2 p_d(f)) \leq H_d(f^{[k]}) \leq H_d^{pure}(f^{[k]}) \leq \lceil k(-\log_2 p_d(f)) + \log_2(nk) \rceil .$$

From this we have the following result.

COROLLARY 4.2. *For any Boolean function f on n variables and nonnegative integer d:*

1. *If $p_d(f) = 1/2$ then $H_d(f^{[k]}) = k$ for all k.*
2. *If $p_d(f) > 1/2$ then $H_d^{pure}(f^{[k]}) = k \log_2 \frac{1}{p_d(f)} + o(k)$.*

Next we need to connect the quantity $p_d(f)$ to the sign degree $deg^s(f)$. The following relationship has appeared elsewhere in slightly different form (see, e.g., [BS]).

PROPOSITION 4.2. *For any Boolean function f, $p_d(f) > 1/2$ if and only if $d \geq deg^s(f)$.*

Proof. Let $d \geq deg^s(f)$ and let the variable set X be $x_1, \ldots, x_n$. Then there is an n-variate polynomial $g(x_1, \ldots, x_n)$ of degree at most d such that $g(\alpha) > 0$ if and only if $f(\alpha) = 1$ for all assignments $\alpha = (\alpha_1, \ldots, \alpha_n)$. By adding a small constant to the polynomial we may assume that $g(\alpha)$ is never 0 and by an appropriate scaling we may assume without loss of generality that the sum of the absolute values of the coefficients of g is 1. Consider the following randomized decision tree: Choose a monomial of g uniformly at random, where the probability that a given monomial is chosen is the absolute value of its coefficient. Probe the variables of the monomial and output the product of the values. It is easily seen that for any assignment α, the probability of correctly evaluating $f(\alpha)$ minus the probability of incorrectly evaluating $f(\alpha)$ is equal to $|g(\alpha)|$, which is strictly positive (here we use that our domain is $\{-1, 1\}$). Thus for any α this algorithm correctly evaluates $f(\alpha)$ with probability exceeding 1/2.

Now suppose $p_d(f) > 1/2$. There must exist a randomized decision tree Q on depth-d trees that evaluates $f(\alpha)$ correctly with probability exceeding 1/2. It is well known and easy to see (by induction on d, looking at the two subtrees of the root) that if T is a decision tree of depth d on variables $\{x_1, \ldots, x_n\}$, then there is a polynomial g_T of degree d such that $g_T(\alpha) = T(\alpha)$ for all assignments α. Define the polynomial $g = -\frac{1}{2} + \sum_{T \in \mathcal{T}_d} Q(T) g_T$, where $Q(T)$ is the probability that T is selected under the distribution Q. Then $g(\alpha) = \text{Prob}_Q[\tilde{T}(\alpha) = 1] - 1/2$. By the choice of Q, this latter term is positive if and only if $f(\alpha) = 1$. □

Theorem 4.2 (and hence Theorem 4.1) now follows easily.

Proof of Theorem 4.2. By Proposition 4.2, $p_{deg^s(f)-1}(f) = 1/2$ and therefore, by Corollary 4.2, $H_{deg^s(f)-1}(f^{[k]}) = k$ for all k. Also by Proposition 4.2, $p_{deg^s(f)} > 1/2$, and so Corollary 4.2 implies that $H^{pure}_{deg^s(f)}(f^{[k]}) = \log_2 \frac{1}{p_{deg^s(f)}(f)} k + o(k)$. Taking $\delta_f = \log_2(2p_{deg^s(f)})$ yields the second part of Theorem 4.2. □

Remark 2. It is interesting to note that, for k large enough, it is possible to obtain an optimal algorithm in which all of the decision trees have a particularly simple form. The randomized algorithm in the proof of Proposition 4.2 uses only decision trees that correspond to computing monomials of g. Using this randomized algorithm in the proof of the upper bound of Lemma 4.2, the decision trees used in the help-bits algorithm are all of the same form.

Remark 3. In the proof of the lower bound in Lemma 4.2, we used Theorem 3.1 in order to deduce that for any forest F of depth at most d, the probability, with respect to a particular distribution P on assignments, that F is correct for all k functions is at most $p_d(f)^k$. In the special case $d = deg^s(f) - 1$, which is the relevant case for proving that $C_{k-1}(f^{[k]}) > deg^s(f) - 1$ in Theorem 4.1, there is an alternative argument which we now sketch. As noted above, for $d = deg^s(f) - 1$, we have $p_d(f) = 1/2$, and thus for $\tilde{\alpha}$ selected from $\hat{D}$ (the distribution of Lemma 4.1), any decision tree of depth d agrees with f with probability exactly $1/2$. In particular, this can be shown to imply that if we fix the values of any d variables, then either that partial assignment occurs with probability 0 under $\hat{D}$, or the value of f conditioned on this assignment is unbiased.

Now, define the random variable c_i to be 0 if $T_i(\tilde{\alpha}) = f_i(\tilde{\alpha})$ and 1 otherwise. We want to show that the probability that $c_i = 0$ for all i is at most $1/2^k$. In fact, the distribution on $(c_1, \ldots, c_k)$ is uniform on $\{0,1\}^k$. By the XOR lemma of [Vaz] (see also [CGHFRS]), a distribution over $\{0,1\}^k$ is uniform if for any subset J of $\{1, \ldots, k\}$, the random variable c_J defined to be the XOR of the c_i for $i \in J$ is unbiased. Let s_J be the probability that $c_J = 0$. The event $c_J = 0$ is the same as the event that $T_J(\tilde{\alpha})(= \prod_{i \in J} T_i(\tilde{\alpha}))$ is equal to $f_J(\tilde{\alpha})(= \prod_{i \in J} f_i(\tilde{\alpha}))$. Now by combining the decision trees $\{T_i | i \in J\}$ we get a single decision tree of depth at most $|J|d$ that computes T_J. We claim that such a decision tree must agree with f_J with probability exactly $1/2$, which is enough to finish the argument. We prove the claim by showing that for each leaf of the tree T_J that is reached with nonzero probability, $f_J(\tilde{\alpha})$ conditioned on $\tilde{\alpha}$ reaching the leaf is unbiased. For each such leaf of the tree, there is an $i \in J$ such that at most d variables of X_i appear on the path. Recall that the value of f_i is unbiased when conditioned on the values of these d variables. If we further condition the value of f_J by the values of all variables not in X_i, then f_i is still unbiased and therefore so is f_J.

Remark 4. One implication of Theorem 4.1 is that for every f, and k sufficiently large (depending on f), $C^{pure}_{k-1}(f^{[k]}) = C_{k-1}(f^{[k]})$, the best algorithm for computing $f^{[k]}$ using $k-1$ help bits uses pure trees. It is reasonable to speculate that this is the case for f for all k and l, and this is open. The first case of interest would be the case $k = 2$ and $l = 1$. In this case, it is not hard to show that $C^{pure}_1(f^{[2]}) = C(f)$, i.e., a pure algorithm computing $f^{[2]}$ with one help bit does no better then the ordinary decision tree complexity of f. (Very recent work of Beigel and Hirst [BH] extends this to general (not necessarily pure) algorithms.) To see this, note that the value of the help bit partitions the set of assignments of $X = X_1 \cup X_2$ into two groups A_1 and A_0. It is not hard to see that either the set of assignments on X_1 induced by A_1 is all of B^{X_1}, or the set of assignments on X_2 induced by A_0 must be all of B^{X_2}. Thus in the case that the help bit is 1, the decision tree used to evaluate (the copy of) f on the

variables X_1 must correctly evaluate f for all assignments to X_1 and thus has depth at least $C(f)$, and similarly in the case that the help bit is 0, the decision tree used to evaluate f on the variables X_2 must correctly evaluate f for all assignments to X_2 and thus has depth at least $C(f)$.

Acknowledgments. The authors have had many conversations with several people regarding this research. We would especially like to acknowledge the contributions of Richard Beigel, Nati Linial, Russell Impagliazzo, and Avi Wigderson. We also thank two anonymous referees for their extensive comments, which helped us to improve the exposition.

REFERENCES

[ABG] A. Amir, R. Beigel, and W. Gasarch, *Some connections between bounded query classes and nonuniform complexity*, in Proc. Structure in Complexity 5th Annual Conference, IEEE Computer Society, Los Alamitos, CA, 1990, pp. 232–243.

[Ant] M. Anthony, *On the Number of Boolean Functions of a Given Threshold Order*, Technical Report LSE-MPS-36, London School of Economics, 1992.

[Be87] R. Beigel, *A structural theorem that depends quantitatively on the complexity of SAT*, in Proc. Structure in Complexity 2nd Annual Conference, IEEE Computer Society, Los Alamitos, CA, 1987, pp. 28–32.

[Be93] R. Beigel, *The polynomial method in circuit complexity*, in Proc. Structure in Complexity 8th Annual Conference, IEEE Computer Society, Los Alamitos, CA, 1993, pp. 82–95.

[BH] R. Beigel and T. Hirst, *One help bit doesn't help*, in Proc. 30th ACM Symposium on Theory of Computing, Dallas, 1998, pp. 124–130.

[BS] J. Bruck and R. Smolensky, *Polynomial threshold functions, AC^0 functions and spectral norms*, SIAM J. Discrete Math., 21 (1992), pp. 33–42.

[Bsh] N. H. Bshouty, *On the extended direct sum conjecture*, in Proc. 21st ACM Symposium on Theory of Computing, Association of Computing Machinery, New York, 1989, pp. 177–185.

[Cai] J. Cai, *Lower bounds for constant depth circuits in the presence of help bits*, Inform. Process. Lett., 36 (1990), pp. 79–84.

[CH] J. Cai and L. Hemachandra, *Enumerative counting is hard*, Inform. and Comput., 82 (1989), pp. 34–44.

[CGHFRS] B. Chor, O. Goldreich, J. Håstad, J. Friedman, S. Rudich, and R. Smolensky, *The bit extraction problem of t-resilient functions*, in Proc. 26th Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1985, pp. 396–407.

[FKN] T. Feder, E. Kushilevitz, and M. Naor, *Amortized communication complexity*, in Proc. 32nd Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1991, pp. 239–248.

[GF] G. Galbiati and M. J. Fischer, *On the complexity of 2-output boolean networks*, Theoret. Comput. Sci., 16 (1981), pp. 177–185.

[IRW] R. Impagliazzo, R. Raz, and A. Wigderson, *A direct product theorem*, in Proc. 9th Conference on Structure in Complexity Theory, IEEE Computer Society, Los Alamitos, CA., 1994, pp. 88–96.

[JT] J. Ja'Ja' and J. Takche, *On the validity of the direct sum conjecture*, SIAM J. Comput., 15 (1986), pp. 1004–1020.

[KKN] M. Karchmer, E. Kushilevitz, and N. Nisan, *Fractional covers and communication complexity*, in Proc. 7th Conference on Structures in Complexity Theory, IEEE Computer Society, Los Alamitos, CA, 1992, pp. 262–274.

[KRW] M. Karchmer, R. Raz, and A. Wigderson, *On proving super-logarithmic depth lower bounds via the direct sum in communication complexity*, in Proc. 6th Conference on Structures in Complexity Theory, IEEE Computer Society, Los Alamitos, CA, 1991, pp. 299–304.

[Lin] N. Linial, *personal communication*, Jerusalem, 1993.

[Paul] W. J. Paul, *Realizing boolean functions on disjoint set of variables*, Theoret. Comput. Sci., 2 (1978), pp. 383–396.

[Raz] R. Raz, *A parallel repitition theorem*, in Proc. 27th annual ACM Symposium on Theory of Computing, Association of Computing Machinery, New York, 1995, pp. 447–456.

[Sa93] M. Saks, *Slicing the hypercube*, in Surveys in Combinatorics, Keith Walker, ed., London Math. Soc. Lecture Note Ser. 187, Cambridge University Press, Cambridge, UK, 1993, pp. 211–255.

[Ulig] D. Ulig, *On the synthesis of self-correcting schemes from functional elements with a small number of reliable components*, Math Notes Acad. Sci. USSR, 15 (1974), pp. 558–562.

[Vaz] U. Vazirani, *Randomness, Adversaries and Computation*, Ph.D. Thesis, Department of Computer Science and Electrical Engineering, U.C. Berkeley, Berkeley, CA, 1986.

[Y2] A. Yao, *Probabilistic computations: Towards a unified measure of complexity*, in Proc. 18th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 1051–1072

A TIME-SPACE TRADEOFF FOR UNDIRECTED GRAPH TRAVERSAL BY WALKING AUTOMATA*

PAUL BEAME†, ALLAN BORODIN‡, PRABHAKAR RAGHAVAN§, WALTER L. RUZZO†, AND MARTIN TOMPA†

Abstract. We prove a time-space tradeoff for traversing undirected graphs, using a structured model that is a nonjumping variant of Cook and Rackoff's "jumping automata for graphs."

Key words. graph connectivity, graph reachability, time-space tradeoff, walking automaton, jumping automaton, JAG

AMS subject classifications. 05C40, 68Q05, 68Q10, 68Q15, 68Q20, 68Q25

PII. S0097539793282947

1. The complexity of graph traversal.

Graph traversal is a fundamental problem in computing, since it is the natural abstraction of many search processes. In computational complexity theory, graph traversal (or, more precisely, *st*-connectivity) is a fundamental problem for an additional reason: understanding the complexity of directed versus undirected graph traversal seems to be the key to understanding the relationships among deterministic, probabilistic, and nondeterministic space-bounded algorithms. For instance, although directed graphs can be traversed nondeterministically in polynomial time and logarithmic space simultaneously, it is not widely believed that they can be traversed deterministically in polynomial time and small space simultaneously. (See Tompa [32] and Edmonds and Poon [22] for lower bounds and Barnes et al. [5] for an upper bound.) In contrast, *undirected* graphs can be traversed in polynomial time and logarithmic space *probabilistically* by using a random walk (Aleliunas et al. [2], Borodin et al. [15]); this implies similar resource bounds on (nonuniform) deterministic algorithms (Aleliunas et al. [2]). More recent work presents uniform deterministic polynomial time algorithms for the undirected case using sublinear space (Barnes and Ruzzo [8]), and even $O(\log^2 n)$ space (Nisan [28]), as well as a deterministic algorithm using $O(\log^{1.5} n)$ space, but more than polynomial time (Nisan, Szemerédi, and Wigderson [29]).

In this paper we concentrate on the undirected case. The simultaneous time and space requirements of the best-known algorithms for undirected graph traversal are as follows. Depth-first or breadth-first search can traverse any n vertex, m edge undirected graph in $O(m+n)$ time, but requires $\Omega(n)$ space. Alternatively, a random walk can traverse an undirected graph using only $O(\log n)$ space, but requires $\Theta(mn)$ expected time (Aleliunas et al. [2]). In fact, Feige [23], based on earlier work

*Received by the editors March 5, 1993; accepted for publication (in revised form) February 7, 1997; published electronically January 29, 1999. This material is based upon work supported in part by the Natural Sciences and Engineering Research Council of Canada, by the National Science Foundation under grants CCR-8703196, CCR-8858799, CCR-8907960, and CCR-9002891, and by IBM under Research Contract 16980043. A portion of this work was performed while the fourth author was visiting the University of Toronto, whose hospitality is gratefully acknowledged.

http://www.siam.org/journals/sicomp/28-3/28294.html

†Department of Computer Science and Engineering, University of Washington, Box 352350, Seattle, WA 98195 (beame@cs.washington.edu, ruzzo@cs.washington.edu, tompa@cs.washington.edu).

‡Department of Computer Science, University of Toronto, Toronto, ON, Canada M5S 1A4 (bor@cs.toronto.edu).

§IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120 (pragh@almaden.ibm.com).

of Broder et al. [18] and Barnes and Feige [7], has shown that there is a spectrum of compromises between time and space for this problem: any graph can be traversed in space S and expected time T, where $ST \leq mn(\log n)^{O(1)}$. This raises the intriguing prospect of proving that logarithmic space and linear time are not simultaneously achievable or, more generally, proving a time-space tradeoff that closely matches these upper bounds.

Although it would be desirable to show a tradeoff for a general model of computation such as a random access machine, obtaining such a tradeoff is beyond the reach of current techniques. Thus it is natural to consider a "structured" model (Borodin [14]), that is, one whose basic move is based on the adjacencies of the graph, as opposed to one whose basic move is based on the bits in the graph's encoding. An appropriate structured model for proving such a tradeoff is some variant of the JAG ("jumping automaton for graphs") of Cook and Rackoff [20]. Such an automaton has a set of states, and a limited supply of pebbles that it can move from vertex to adjacent vertex ("walk") or directly to a vertex containing another pebble ("jump"). The purpose of its pebbles is to mark certain vertices temporarily, so that they are recognizable when some other pebble reaches them. The pebbles represent vertex names that a structured algorithm might record in its workspace. Walking represents replacing a vertex name by some adjacent vertex found in the input. Jumping represents copying a previously recorded vertex name.

Rabin (see [20]), Savitch [31], Blum and Sakoda [13], Blum and Kozen [12], Hemmerling [24], and others have considered similar models; see Hemmerling's monograph for an extensive bibliography (going back over a century) emphasizing results for "labyrinths": graphs embedded in two- or three-dimensional Euclidean space.

The JAG is a structured model, but not a weak one. In particular, it is general enough to encompass in a natural way most known algorithms for graph traversal. For instance, a JAG can execute a depth-first or breadth-first search, provided it has one pebble for each vertex, by leaving a pebble on each visited vertex in order to avoid revisiting it, and keeping the stack or queue of pebble names in its state. Furthermore, as Savitch [31] shows, a JAG with the additional power to move a pebble from vertex i to vertex $i+1$ can simulate an arbitrary Turing machine on directed graphs. Even without this extra feature, we have shown [10] that JAGs are as powerful as Turing machines for the purposes of solving undirected graph problems (our main focus).

Cook and Rackoff define the time T used by a JAG to be the number of pebble moves, and the space to be $S = P \log_2 n + \log_2 Q$, where P is the number of pebbles and Q the number of states of the automaton. (Keeping track of the location of each pebble requires $\log_2 n$ bits of memory, and keeping track of the state requires $\log_2 Q$.) It is well known that *st*-connectivity for directed graphs can be solved by a deterministic Turing machine in $O(\log^2 n)$ space, by applying Savitch's theorem [30] to the obvious $O(\log n)$ space nondeterministic algorithm for the problem. Cook and Rackoff show that the same $O(\log^2 n)$ space upper bound holds for deterministic JAGs by direct construction of an $O(\log n)$ pebble, $n^{O(1)}$ state deterministic JAG for directed *st*-connectivity. More interestingly, they also prove a lower bound of $\Omega(\log^2 n / \log\log n)$ on the space required by JAGs solving this problem, nearly matching the upper bound. Standard techniques (Adleman [1], Aleliunas et al. [2]) extend this result to any randomized JAG whose time bound is at most exponential in its space bound. Berman and Simon [11] extend this space lower bound to probabilistic JAGs with even larger time bounds, namely, exponential in $\log^{O(1)} n$.

In this paper we use a variant of the JAG to study the tradeoff between time and space for the problem of *undirected* graph traversal. The JAG variant we consider is

more restricted than the model introduced by Cook and Rackoff, because the pebbles are not permitted to jump. This nonjumping model is closer to the one studied by Blum and Sakoda [13], Blum and Kozen [12], and Hemmerling [24]. We will distinguish this nonjumping variant by referring to it as a WAG: "walking automaton for graphs."

Several authors have considered traversal of undirected regular graphs by a WAG with an unlimited number of states but only the minimum number (one) of pebbles, a model better known as a *universal traversal sequence* (Aleliunas et al. [2], Alon, Azar, and Ravid [3], Bar-Noy et al. [4], Borodin, Ruzzo, and Tompa [16], Bridgland [17], Buss and Tompa [19], Istrail [25], Karloff, Paturi, and Simon [26], Tompa [33]). A result of Borodin, Ruzzo, and Tompa [16] shows that such an automaton requires $\Omega(m^2)$ time (on regular graphs with $3n/2 \leq m \leq n^2/6-n$). Thus, for the particularly weak version of logarithmic space corresponding to the case $P = 1$, a quadratic lower bound on time is known.

The known algorithms and the lower bounds for universal traversal sequences suggest that the true time-space product for undirected graph traversal is approximately quadratic, perhaps $\Theta(mn)$. The result of this paper is a lower bound that provides progress toward proving this conjecture. More specifically, we prove lower bounds on time that are nonlinear in m for a wide range of values of P. In particular, for any WAG M solving st-connectivity in logarithmic space, there is a family of regular graphs on which M requires time $m^{1+\Omega(1)}$. Near the other extreme, if M uses a number of pebbles that is sublinear in m, there is a family of regular graphs on which M requires time superlinear in m. Although these give the desired quadratic lower bound only at the extreme of linear time, they each at least establish that logarithmic space and linear time are not simultaneously achievable on the nonjumping model when $m = \omega(n)$. (They do not settle the question of simultaneous achievability of logarithmic space and linear time when $m = O(n)$ since the families of regular graphs mentioned above have degree $d = \omega(1)$ and hence $m = \omega(n)$; see sections 3 and 4.)

We prove upper and lower bounds for undirected graph problems on other variants of the JAG in a companion paper [10]. Following the preliminary appearance of these results, Edmonds [21] proved a much stronger result for traversing undirected graphs, and Barnes and Edmonds [6] and Edmonds and Poon [22] proved even more dramatic tradeoffs for traversing directed graphs.

2. Walking automata for graphs. The problem we will be considering is "undirected st-connectivity": given an undirected graph G and two distinguished vertices s and t, determine if there is a path from s to t.

Consider the set of all n-vertex, edge-labeled, undirected graphs $G = (V, E)$ with maximum degree d. For this definition, edges are labeled as follows. For every edge $\{u, v\} \in E$ there are two labels $\lambda_{u,v}, \lambda_{v,u} \in \{0, 1, \ldots, d-1\}$ with the property that, for every pair of distinct edges $\{u, v\}$ and $\{u, w\}$, $\lambda_{u,v} \neq \lambda_{u,w}$. It will sometimes be convenient to treat an undirected edge as a pair of directed *half-edges*, each labeled by a single label. For example, the half-edge directed from u to v is labeled $\lambda_{u,v}$.

Following Cook and Rackoff [20], a WAG is an automaton with Q states and P distinguishable pebbles, where both P and Q may depend on n and d. For the st-connectivity problem, two vertices s and t of its input graph are distinguished. The P pebbles are initially placed on s. Each move of the WAG depends on the current state, which pebbles coincide on vertices, which pebbles are on t, and the edge labels emanating from the pebbled vertices. Based on this information, the automaton changes state and selects some pebble p and some $i \in \{0, 1, \ldots, d-1\}$.

The selected i must be an edge label emanating from the vertex currently pebbled by p, and p is moved to the other endpoint of the edge with label i. (The decision to make t "visible" to the WAG but s "invisible" was made simply to render one-pebble WAGs on regular graphs equivalent to universal traversal sequences.) A WAG that determines st-connectivity is required to enter an accepting state if and only if there is a path from s to t. Note that WAGs are nonuniform models.

We have defined WAGs running on arbitrary graphs, but our lower bounds apply even to WAGs that are only required to operate correctly on regular graphs. The restriction to regular graphs, in addition to strengthening the results, provides comparability to the known results about universal traversal sequences. A technicality that must be considered in the case of regular graphs is that they do not exist for all choices of degree d and number of vertices n, as is seen from the following proposition.

PROPOSITION 1. *d-regular, n-vertex graphs exist if and only if dn is even and $d \leq n-1$.*

(See [16, Proposition 1], for example, for a proof.) To allow use of Ω-notation in expressing our lower bounds, however, the "time" used by a WAG must be defined for *all* sufficiently large n. To this end, we consider the time used by a WAG on d-regular, n-vertex graphs where dn is odd to be the same as its running time on d-regular, $(n+1)$-vertex graphs.

3. The tradeoff. In this section we prove time lower bounds for WAGs with P pebbles. The proof generalizes an unpublished construction of Szemerédi (communicated to us by Sipser) that proved an $\Omega(n \log n)$ lower bound on the length of universal traversal sequences for 3-regular graphs.

THEOREM 2. *Let P and d be fixed functions of n with dn even, $P \geq 1$, $d \geq 6$, and $d^2 + Pd = o(n)$. Let $m = dn/2$, $\epsilon = 1/(3\ln(6e))$, and*

$$d_0 = (2P/e)^{3P/(3P+2)}\, n^{1/(3P+2)}.$$

Let M be any (deterministic) WAG with P pebbles that determines st-connectivity for all d-regular, n-vertex graphs. Then M requires time

(a) $\Omega\left(m(\log n)\frac{d/P}{\log(d/P)}\right)$, *if $P \leq \epsilon \ln(n/d^2)$ and $6P \leq d \leq d_0$,*

(b) $\Omega\left(mP\left(\frac{n}{d^2}\right)^{\frac{1}{3P}}\right)$, *if $P \leq \epsilon \ln(n/d^2)$ and $d_0 < d$, and*

(c) $\Omega\left(m \min\left(d, \log \frac{n}{(d^2+Pd)}\right)\right)$, *otherwise.*

Before proving the theorem, we will make a few observations about it. Perhaps the most noteworthy is that these bounds are nonlinear whenever either $d = \omega(1)$ or $d \geq 6P$.

It is obvious that the regions (i.e., the sets of (P, d) pairs) where the three cases apply are pairwise disjoint. It is also true that all three regions are nonempty for all sufficiently large n, although we will not justify this statement.

Although they have very different forms, the three bounds meet "smoothly," except along the line segment $d = 6P$, $1 \leq P \leq \epsilon \ln(n/d^2)$. Specifically, we will show that where any pair of the three bounds meet along the curve $P = \epsilon \ln(n/d^2)$, $d \geq 6P$, both are $\Theta(m \log(n/d^2))$, and where bounds (a) and (b) meet along the curve $d = d_0$, $1 \leq P \leq \epsilon \ln(n/d^2)$, both are $\Theta(md_0)$.

All three bounds are increasing functions of d (recall $m = dn/2$). The ratio of the lower bounds to m is also an interesting quantity. Note that the ratio of bound (a) to m is an increasing function of d, while that of bound (b) is decreasing. Since they are equal (within constant factors) at $d = d_0$, the two could be combined into the single expression $\Omega(m \min((\log n)(d/P)/\log(d/P),\ P(n/d^2)^{1/(3P)}))$, as was done in bound (c).

It seems likely that the decrease in bound (b) is an artifact of the proof technique rather than an intrinsic reduction in the complexity of the problem, since intuitively higher degree would seem to make the search more difficult. On the other hand, higher degree reduces the graph's maximum possible diameter, which perhaps helps. It is known that the length of universal traversal sequences is not monotonic in d, although it may be monotonic up to some large threshold, perhaps $d = \lfloor n/2 \rfloor - 1$. (See Borodin, Ruzzo, and Tompa [16] for a discussion.) Similarly, the complexity of st-connectivity is not monotone in d, since regular graphs of degree $d > \lfloor n/2 \rfloor - 1$ are necessarily connected, but it is plausibly monotone for d up to cn, for some constant $0 < c < 1/2$.

Two special cases of the theorem are of particular interest. Namely, the following two corollaries show that logarithmic space implies time $m^{1+\Omega(1)}$ and that sublinear space implies superlinear time.

COROLLARY 3. *Let M be a WAG with P pebbles that determines st-connectivity for all regular n-vertex graphs. If $P = O(1)$, then there is a family of regular graphs on which M requires time $\Omega(m^{1+1/(3P+3)})$.*

Proof. Consider the family of regular graphs with degree $d = d_0 = \Theta(n^{1/(3P+2)})$. Theorem 2 applies, specifically case (a). This gives a time lower bound of $\Omega(md) = \Omega(m^{1+1/(3P+3)})$. □

For $P = 1$ the $\Omega(m^{7/6})$ bound given above is not as strong as the $\Omega(m^2)$ bound given by Borodin, Ruzzo, and Tompa [16] but is included for comparative purposes. Also, the $\Omega(m^2)$ lower bound for universal traversal sequences holds for degree up to $n/3 - 2$, so the decrease in the ratio of bound (b) to m noted above certainly *is* an artifact of our proof when $P = 1$.

COROLLARY 4. *Let M be a WAG with P pebbles that determines st-connectivity for all regular n-vertex graphs. If $P = o(n)$, then there is a family of regular graphs on which M requires time $\Omega(m \log(n/P)) = \omega(m)$.*

Proof. Suppose $P \geq n^{1/3}$. Consider the family of regular graphs with degree $d = \sqrt{n/P} = \omega(1)$. Then $d^2 + Pd \leq 2\sqrt{Pn} = o(n)$, so Theorem 2 applies, specifically case (c). This gives a lower bound of $\Omega(m \log d) = \Omega(m \log(n/P))$ on time. When $P < n^{1/3}$, a similar analysis suffices, choosing $d = \log n$. □

Corollary 4 is tight: time $O(m)$ is possible with $O(n)$ pebbles [10, Theorem 15]. Note also that, when $P = \Theta(n)$, the time is still $\Omega(m \log(n/P))$ [10, Theorem 3].

Various constants in the theorem can be improved by slight modification to the construction and/or its analysis, but in the interest of clarity we will not present these refinements.

Proof of Theorem 2. The idea underlying the proof is to build a graph with many copies of some fixed gadgets, each with many "entry points." Since M does not have enough pebbles to mark all the gadgets it has explored, it must spend time reexploring each gadget from different entry points, or it risks the possibility that one of them might never be fully explored. The crux of the argument is to choose the right gadgets and to interconnect them so that we can be sure this happens. We use an "adversary" argument to show this. We begin by giving an overview of the argument, followed

by more detailed descriptions of the gadgets and adversary strategy, and finally the analysis.

Overview. Imagine the adversary "growing" the graph as follows. At a general point in the construction, the graph consists of some gadgets that are fully specified except for the interconnections among their "entry point" vertices. The adversary simulates M on this partial graph until M attempts to move some pebble p out of an entry point using a label for which no edge is yet defined. Our main freedom in the construction is the choice of the gadget at the other endpoint of this interconnecting edge f. The adversary will pick it so that M will spend a nonnegligible number of steps τ "exploring" the gadget reached through f. The adversary can achieve this for most of the $\Omega(m)$ interconnecting edges, yielding an $\Omega(m\tau)$ lower bound on time. The parameter τ will vary depending on n, P, and d, giving the three lower bounds quoted in the statement of the theorem.

The interconnecting edge f is chosen as follows. Note that no single labeled gadget γ will suffice to keep p "busy" for τ steps. For example, M's very next move of p, say by label a, might be an exit from γ. On the other hand, if the adversary can learn that M's next move of p will be on label a, it can choose some gadget in which label a moves from an entry point *into* the gadget, rather than exiting from it. Similarly, if it can learn the next τ moves by p (and/or other pebbles following p across f), the adversary can choose a gadget in which this whole sequence of moves avoids exiting from the gadget. A key point is that M can sense only very limited facts about the gadget that p enters when it crosses f. Suppose p has just crossed f, arriving at a vertex v. M can sense (i) the degree of v, (ii) whether v is the target vertex t, and (iii) whether there are other pebbles on v. Thus, in general M has several possible next moves for p, based on which of these conditions hold. The adversary avoids having to consider these alternative futures by assuming, respectively, (1) that the graph is d-regular, (2) that M does not reach t (within $\Omega(m\tau)$ steps), and (3) that f connects to a gadget that contains no other pebbles when p enters it, and that remains free of other pebbles (except perhaps ones that follow p across f) for τ moves. Given these assumptions, the adversary will be able to deterministically simulate the next several moves by M so that it can decide which labeled gadget can host those moves without allowing a pebble to exit. Of course, the adversary must also ensure that assumptions (1)–(3) are ultimately justified. Building a d-regular graph requires some care but is not too difficult. Assumption (2) will follow easily if each connecting edge accounts for τ moves. Assumption (3) is slightly trickier; we will return to it below.

We view the overall adversary strategy as a two-phase process. A *local* phase determines the internal ("local") structure required of a gadget hosting the next several moves of p so that no pebble will exit this gadget until at least τ moves have been charged to it, starting after p's entry. The basic idea is to use a "lazy, greedy" definition: lazy in that the adversary will not define a labeled half-edge in the gadget until just before M needs to move a pebble across it, and greedy in that when such a half-edge is defined, it will be defined to stay within the gadget. Of course, this cannot continue indefinitely, but it will be possible for at least the first τ moves within the gadget. Thus, pebble motion across half-edges exiting the gadget is deferred for at least this long.

The adversary's simulation of M is now "rolled back" to the point at which p crossed f. The *global* phase of the adversary's strategy is to choose a gadget already present in the graph and to connect f to it. Recall that our goal is to reuse each gadget many times, so that the total time spent in it asymptotically exceeds its number of

edges. (Occasionally, when all entry points of suitable gadgets have been used, a new copy of the needed gadget will be added. This process terminates when the number of vertices in the graph approaches n.) The gadget chosen for f must match the gadget determined by the local phase, must have an unused entry point to which to connect f, and (before f was connected to it) must have remained free of pebbles from the time when p crossed f until τ moves were charged to it. The "pebble-free" condition ensures assumption (3) above. Such a condition is necessary since, if it were violated, M might encounter "unexpected" pebbles in the chosen gadget, i.e., pebbles not encountered during the simulation in the local phase. This could cause M to deviate from the sequence of moves predicted by the local phase, and so possibly allow p or one of the pebbles that followed it across f to exit from the gadget in fewer than τ moves.

A point we slighted above is that the "τ steps" under discussion are not necessarily consecutive and are not necessarily all made by p or by pebbles that followed p across f. For example, p's moves after crossing f might be interleaved with moves by some other pebble p' after crossing another undefined edge f' and/or many previously defined connecting edges. In general, the adversary keeps track of these many interleaved activities by charging pebble moves to connecting edges, with the "local phase" for an undefined connecting edge f being the interval between its charge reaching 1 (at the first crossing of f by some pebble) and its charge reaching τ.

The final issue to address is that we want to avoid adding a new copy of a gadget until all entry points of most existing copies have been used. Specifically, we will have at most a fixed number ($P(\tau+2)+d$, to be precise) of "open" copies of each gadget at any time. As noted above, many steps may occur between the first and τth steps charged to f. During this interval, other pebbles might touch *all* open copies of the gadget needed for f, leaving no *pebble-free* open gadget to which to connect f. Our solution to this problem is found in the adversary's method of charging pebble moves to edges. Moves in f's gadget are always charged to f. In addition, certain moves touching other gadgets are charged to f also. With this scheme, we can bound both the number of moves that occur in f's gadget and the number of other gadgets that are touched by pebbles during f's local phase. Thus, no gadget is expected to absorb too many moves, and there will be at least one suitable pebble-free open copy of the needed gadget when f accumulates charge τ.

The construction will "waste" (i.e., not fully utilize the connecting edges of) up to $P(\tau+2)+d$ copies of each gadget. The main constraint that limits τ is that it must be small enough that this waste is small, i.e., $P(\tau+2)+d$ times the number of distinct types of labeled gadgets times the number of connecting edges per gadget is small compared to the total number of connecting edges.

We will now present the construction in more detail. We actually define a sequence of graphs $G_{i,j},\ 0 \le i \le \mu,\ 0 \le j$, representing successive phases of the construction. Like τ, the parameter μ varies slightly depending on n, p, and d, but will be $\Theta(m)$ in each case. (The maximum value of j is unimportant, but turns out to be about $P\tau$.) Each graph consists of the following:

1. A set of gadgets, each with the same size S and number L of entry vertices, and a fully defined internal structure and labeling. Each vertex that is not an entry vertex has degree d. There is a fixed $d' \ge 1$ such that each entry vertex has d' edges to neighbors in the same gadget, and up to $d-d'$ *connecting edges* joining it to the entry vertices of other gadgets or protogadgets (see below). We will show that $d-d' \ge d/2$ and that $L/S > 1/3$, ensuring at termination that the number of connecting edges is $\Theta(m)$.

2. A set of labeled *committed* connecting edges joining entry vertices of gadgets. $G_{i,j}$ will have exactly i committed connecting edges.

3. A set of up to P partially labeled *uncommitted* connecting edges, each joining an entry vertex u of some gadget to an entry vertex v of a protogadget (see below). The uncommitted half-edge from u to v is labeled, but the half-edge from v to u is unlabeled.

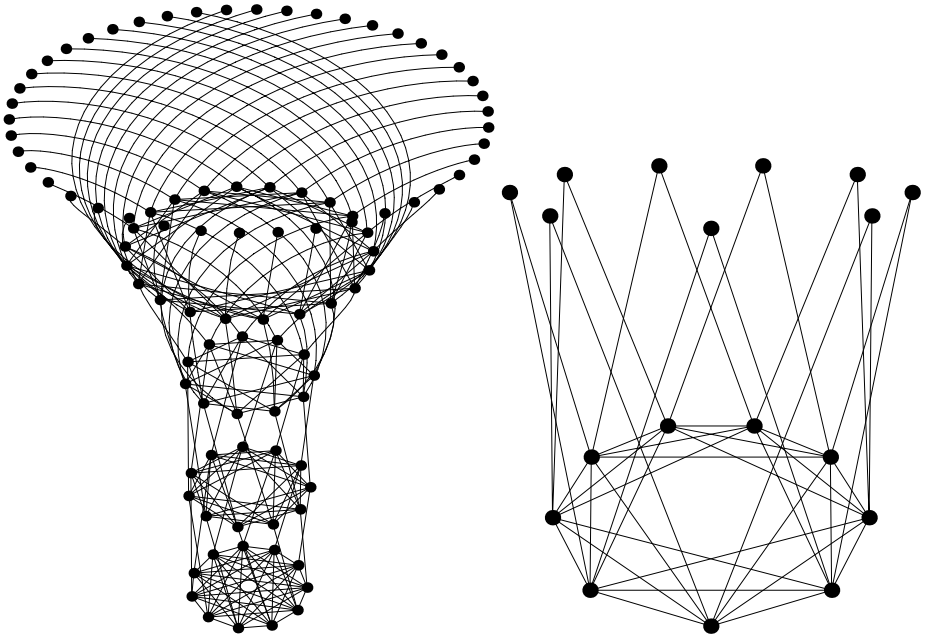
4. A set of up to P partially defined *protogadgets*. Like a gadget, a protogadget has S vertices, including L entry vertices, but unlike the gadgets, the internal structure of a protogadget is in general only partially defined; its vertices may have degree less than d, and its half-edges may not be labeled. In particular, only one entry vertex v of each protogadget will be incident to a connecting edge, say the uncommitted connecting edge $\{u, v\}$, and, as indicated above, the half-edge from v to u will be unlabeled. The protogadgets are the tools used in the local phases of the adversary's strategy.

In outline, the adversary's strategy is as follows. The initial graph $G_{0,0}$ consists of one arbitrarily chosen gadget. The start vertex s is an arbitrary vertex in this gadget. For any $G_{i,j}$, the *initial configuration* of M on $G_{i,j}$ consists of M in its start state and all P pebbles on $G_{i,j}$'s copy of s. Associate with each connecting edge of the graph $G_{i,j}$ an integer *charge*, initially zero. The adversary will charge each pebble motion to at most one connecting edge, according to a rule to be given later. It will simulate M starting from M's initial configuration on $G_{i,j}$ until one of the following two things happens. (It simulates M as if all vertices in $G_{i,j}$ were of degree d, even though some are of smaller degree.)

1. Suppose M attempts to move a pebble from a vertex u across a half-edge labeled a, where no such labeled half-edge exists. If u is an entry vertex in some gadget, add a new uncommitted half-edge from u labeled a to the entry point v of a new protogadget. More precisely, we define $G_{i,j+1}$ to be $G_{i,j}$ plus that half-edge and protogadget. If u is in some protogadget, choose some other vertex v in the *same* protogadget (according to a rule to be given later) and add a half-edge from u to v labeled a. More precisely, we define $G_{i,j+1}$ to be $G_{i,j}$ plus that half-edge (plus a few others, as we will see). The choice of v is not arbitrary; one point we must establish is that there will always be a suitable vertex v when needed. The thrust of this step in the adversary strategy is to keep pebbles "trapped" in protogadgets for as long as possible. This portion of the adversary's strategy is the "local" strategy introduced above, so-called because of its focus on the structure *within* a gadget.

2. Suppose an uncommitted edge f in $G_{i,j}$ accumulates a charge of τ. In this case, we will convert f into a committed edge. More precisely, we will form $G_{i+1,0}$ from $G_{i,j}$ by choosing an existing gadget "similar" to f's protogadget and committing f to enter the chosen gadget. (This is described more fully below.) Again, f cannot be committed arbitrarily; a second point that we must establish is that an appropriate gadget (usually) exists when needed, and that M's behaviors on $G_{i,j}$ and $G_{i+1,0}$ are similar. The thrust of this step is that the size of $G_{i+1,0}$ is growing slowly with i, since we are (usually) able to reuse existing gadgets, but the time M spends in $G_{i+1,0}$ is rising rapidly with i, since a lower bound on the total running time of M is τ times the number of committed edges (i, which is ultimately $\mu = \Theta(m)$). This portion of the adversary's strategy is the "global" strategy, so-called because of its focus on the interconnections among gadgets.

The adversary continues the simulation on $G_{i,j+1}$ or $G_{i+1,0}$ as appropriate, and repeats this process until $G_{\mu,0}$ is constructed.



(a)
$d = 10,\ k = 4,\ q = 2,\ r = 1$
For clarity, only half of the forward edges are shown.

(b)
$d = 8,\ k = 1,\ q = 2,\ r = 1$

FIG. 1. *Examples of the funnel gadgets.*

Gadgets. Before describing the adversary strategy in more detail, we will describe the gadgets and protogadgets. The gadgets are called "funnels." An example is shown in Fig. 1a. The entry vertices are those on the "rim" of the funnel. Intuitively, the adversary will try to "trap" pebbles in a funnel for a while by assigning edge labels so that the moves taken by pebbles in the gadget in the near future (i.e., the next τ moves in the gadget) either stay on the same layer or drop to the next deeper layer. The "cone" portion of the funnel (near the top of Fig. 1a) allows many entry vertices to share vertices in the narrower portion near the bottom of Fig. 1a. An example of a two-layer funnel is shown in Fig. 1b.

Four interrelated parameters k, q, g, and r, which in turn depend on n, P, and d, characterize the gadgets. All four are positive integers. Each gadget has $k+1$ *layers,* numbered 0 through k. Layer $l,\ 0 \le l \le k$, has

$$n_l = (d+1) \cdot \max(1, 2^{\lceil \log_2 k \rceil - l})$$

vertices, designated v_i^l, $0 \le i \le n_l - 1$. The entry vertices are those on layer 0. Hence, the number of entry vertices is

$$L = (d+1) \cdot 2^{\lceil \log_2 k \rceil},$$

and the total number of vertices per gadget is

$$S = (d+1)(2^{\lceil \log_2 k \rceil + 1} - 1 + k - \lceil \log_2 k \rceil).$$

Note that

$$\frac{L}{S} = \frac{(d+1)\cdot 2^{\lceil \log_2 k\rceil}}{(d+1)(2^{\lceil \log_2 k\rceil+1} - 1 + k - \lceil \log_2 k\rceil)} > \frac{2^{\lceil \log_2 k\rceil}}{2^{\lceil \log_2 k\rceil+1} + k} \geq \frac{1}{3}, \tag{1}$$

as promised, and that

$$S \leq (1+1/d)\, d\, (2\cdot 2^{\lceil \log_2 k\rceil} + k) \leq (7/6)\, d\, (5k) < 6dk. \tag{2}$$

The parameter q is an even integer, $2 \leq q$. The d edge labels $\{0, 1, \ldots, d-1\}$ are partitioned into $g = \lfloor d/q \rfloor$ "full" blocks, each of size q, plus perhaps one "partial" block of size $d \bmod q$ in case q does not evenly divide d. The same fixed partition is used for all gadgets and is arbitrary, except that for each $a \in \{0, 1, \ldots, d-1\}$, we place both a and its mate in the same block, where the *mate* of label a is $d-1-a$. Note that if d is odd, then $(d-1)/2$ is its own mate and will be in the partial block.

The remaining gadget parameter r is an integer satisfying $1 \leq r \leq g/3$. Note that the existence of such an r implies that $g \geq 3$, and hence

$$q \leq d/3. \tag{3}$$

Intuitively, r denotes an upper bound on the number of pebbles that we attempt to trap in a given gadget.

The edges within a gadget always connect vertices on the same or adjacent layers. A half-edge is called a "forward" half-edge if it goes from layer l to layer $l+1$, "backward" if it goes to layer $l-1$, and "cross" if it goes to layer l. For each layer l and each block B of labels, there is a $t \in \{\text{forward}, \text{backward}, \text{cross}\}$ such that all half-edges with labels in B leaving vertices on layer l will be of type t. Thus it is natural to refer to the labels and the blocks of labels at a layer as forward, backward, or cross, as well as the half-edges. For $i \in \mathbb{N}$, $a \in \{0, 1, \ldots, d-1\}$, and $0 \leq l \leq k$, define

$$\chi(i,a,l) = \begin{cases} (i+a+1) \bmod n_l & \text{if } a < (d-1)/2, \\ (i+n_l/2) \bmod n_l & \text{if } a = (d-1)/2, \\ (i-(d-1-a)-1) \bmod n_l & \text{if } a > (d-1)/2. \end{cases}$$

If $a \in \{0, 1, \ldots, d-1\}$ is a forward label at layer l, then for $0 \leq i \leq n_l - 1$, a will label the half-edge from vertex v_i^l to vertex $v_{\chi(i,a,l+1)}^{l+1}$. Similarly, if a is a cross label it will go to $v_{\chi(i,a,l)}^{l}$. Notice that a cross edge labeled a will be labeled by a's mate in the reverse direction. No parallel edges arise since $n_l \geq d+1$. As an example, if all edges are cross edges (a case that does not arise in our constructions) and if $n_l = d+1$, then layer l would be a $(d+1)$-clique. As another example, whenever label 0 is a cross label at layer l, the half-edges labeled 0 will form a Hamiltonian cycle through the layer l vertices, and those edges will be labeled $d-1$ (0's mate) in the other direction. Note that the backward labels are not constrained by χ.

The set of gadgets is defined as follows. For $0 \leq l \leq k$ let

$$b_l = \begin{cases} g - r & \text{if } l = 0, \\ r(n_{l-1}/n_l) & \text{otherwise,} \end{cases}$$

$$f_l = \begin{cases} 0 & \text{if } l = k, \\ r & \text{otherwise.} \end{cases}$$

See Table 1. If q does not evenly divide d, then the labels in the partial block will be cross labels at each layer $0 \leq l \leq k$. For each layer $0 \leq l \leq k$, choose f_l of

TABLE 1
Number of edge blocks of each type per layer.

Layer	0	1	$\cdots$	$\lceil \log_2 k \rceil$	$\lceil \log_2 k \rceil + 1$	$\cdots$	$k-1$	k
f_l	r	r	$\cdots$	r	r	$\cdots$	r	0
b_l	$g-r$	$2r$	$\cdots$	$2r$	r	$\cdots$	r	r
Cross	0	$g-3r$	$\cdots$	$g-3r$	$g-2r$	$\cdots$	$g-2r$	$g-r$

the remaining g blocks as forward labels and b_l as backward labels (connecting edge labels, if $l = 0$). All blocks not selected above will be cross labels. Note that the rules in the previous paragraph define the forward and cross half-edges, given their labels, but not the backward half-edges. The chosen backward labels are assigned to these half-edges in an arbitrary but fixed way. Note that there are just enough backward labels: each of the n_{l-1} vertices on level $0 \le l-1 < k$ has exactly qr forward labels, with destinations evenly distributed over the n_l vertices on layer l, so each vertex on layer l is incident to exactly $qr(n_{l-1}/nl) = q \cdot b_l$ edges from layer $l-1$.

For layer 0, the b_0 blocks selected above will label connecting edges. Thus, each entry vertex will be adjacent to exactly $d' = rq + (d \bmod q)$ other vertices in the same gadget, and to $d - d'$ connecting edges. Note, since $r \le g/3$ and $q \le d/3$ (from inequality (3)), that

$$d - d' = gq - rq \ge (2/3)gq = (2/3)\lfloor d/q \rfloor q \ge (2/3)((3/4)(d/q))q = d/2, \tag{4}$$

as claimed earlier. Also note that at most $3r$ blocks are chosen as forward and backward at each layer, and that this is always possible since $g \ge 3r$.

The number of distinct gadget types created by this process is

$$\begin{aligned}
&\binom{g}{r}^{k} \binom{g-r}{2r}^{\lceil \log_2 k \rceil} \binom{g-r}{r}^{k-\lceil \log_2 k \rceil - 1} \binom{g}{r}^{1} \\
&\qquad \le \binom{g}{r}^{2k-\lceil \log_2 k \rceil} \binom{g}{2r}^{\lceil \log_2 k \rceil} \\
&\qquad \le \left(\frac{eg}{r}\right)^{r(2k-\lceil \log_2 k \rceil)} \left(\frac{eg}{2r}\right)^{2r\lceil \log_2 k \rceil} \\
&\qquad \le \left(\frac{eg}{r}\right)^{r(2k+\lceil \log_2 k \rceil)}. \qquad (5)
\end{aligned}$$

Figure 1b fully shows a gadget with $d = 8,\ k = 1,\ q = 2,\ g = 4$, and $r = 1$, with forward edges labeled 0 and 7 from layer 0 and backward edges labeled 3 and 4 from layer 1. Figure 1a shows a gadget with $d = 10,\ k = 4,\ q = 2,\ g = 5$, and $r = 1$, with forward edges labeled 0 from layers 0 through 3. In the interest of clarity, the forward edges labeled 9 (0's mate) are not shown in the figure.

Protogadgets and local strategy. The protogadgets are built incrementally by the adversary. Initially, each consists of S vertices, denoted as in the gadgets, together with the cross edges defined by the partial block of labels (if any) at each level. As discussed previously, the adversary proceeds by simulating M from its initial configuration on $G_{i,j}$. Suppose during the tth step of this simulation that M attempts to move some pebble p along the half-edge labeled a from some vertex u but no such half-edge exists. As sketched earlier, if u is an entry vertex of some gadget, we create a new protogadget into which p will move. If u is a vertex v_i^l in some protogadget

π, the adversary decides whether to make the block of labels containing a all forward half-edges or all cross half-edges (see below). The graph $G_{i,j+1}$ is then defined to be the same as $G_{i,j}$, except that at layer l in π, a's block of half-edges is added. The adversary restarts the simulation of M, starting from M's initial configuration on $G_{i,j+1}$. It should be clear that during the first $t-1$ steps of the simulation, M will behave on $G_{i,j+1}$ exactly as it did on $G_{i,j}$, since $G_{i,j}$ is a subgraph of $G_{i,j+1}$. The tth step, of course, was impossible in $G_{i,j}$, but is possible in $G_{i,j+1}$. Note that p can exit from π only at an entry vertex but is no nearer to such a vertex in $G_{i,j+1}$ after the tth step than before. Thus we can view M as running on a dynamically growing graph, one being built by the adversary so as to trap pebbles in protogadgets for some number of moves. We will adopt this view when no confusion will arise and let $G_{i,*}$ denote the last $G_{i,j}$ built before $G_{i+1,0}$.

Let z be the number of free blocks at level l, i.e., blocks whose half-edges have not yet been defined. The adversary chooses a's block to label cross edges provided $z > b_l + f_l$ and forward edges provided $b_l < z \le b_l + f_l$. If $z \le b_l$, the adversary *fails* (but see Claim 1 below).

Let

$$\tau = \begin{cases} (k - \lceil \log_2 k \rceil)\, \lfloor g/3 \rfloor & \text{if } P \le r, \\ r & \text{if } P > r. \end{cases}$$

Note that $(k - \lceil \log_2 k \rceil)\, \lfloor g/3 \rfloor \ge r$ since $k \ge 1$ and $g/3 \ge r$, so in either case we have

$$(k - \lceil \log_2 k \rceil)\, \lfloor g/3 \rfloor \ge \tau. \tag{6}$$

We prove three claims about the protogadgets. We will see later that the global strategy prevents M from making more than τ moves in any protogadget, so Claim 1 below shows that the adversary will never fail.

CLAIM 1. *The adversary will never fail, provided M makes at most τ moves in any protogadget.*

Proof. First, clearly at most $\min(P, \tau)$ pebbles can enter a protogadget in τ steps, and for the particular definition of τ chosen above, $\min(P, \tau) \le r$. Now, suppose the claim is false. Suppose the adversary first fails during an attempted move at level l in some protogadget π. Then at least $g - b_l$ moves were previously made by pebbles at layer l. As noted, at most r pebbles can enter π in τ moves. It cannot be the case that $l < k$, since for all such layers $g - b_l \ge r = f_l$, so during the last r of the $g - b_l$ moves, all r pebbles moved past layer l, leaving none to cause failure there. Thus, the failure occurred in layer k. For a pebble to reach layer k, it must be that the maximum number of cross edges, plus at least one forward edge, have been previously defined at each layer less than k. Thus, the number of moves completed in this protogadget prior to failure is at least

$$\begin{aligned} &(g - b_k) + \sum_{l=0}^{k-1} (g - b_l - f_l + 1) \\ &= (g - r) + \lceil \log_2 k \rceil\, (g - 3r + 1) + (k - \lceil \log_2 k \rceil - 1)(g - 2r + 1) \\ &\ge (k - \lceil \log_2 k \rceil)(g - 2r + 1) \\ &> (k - \lceil \log_2 k \rceil)\, \lfloor g/3 \rfloor \\ &\ge \tau. \end{aligned}$$

The second inequality uses the assertion that $r \le g/3$, and the third uses inequality (6). ∎

CLAIM 2. *Each protogadget is a subgraph of some gadget.*

Proof. The adversary chooses at most f_l forward blocks and at most $g-(f_l+b_l)$ cross blocks at each layer. Thus there are enough unchosen blocks to select a total of exactly f_l forward and b_l backward blocks, which precisely defines a gadget. □

CLAIM 3. *All entry points of a protogadget are equivalent in the sense that if M makes at most τ moves in a protogadget entered through vertex v_h^0, then the resulting configuration will be exactly the same as if it had entered through vertex v_0^0, except that positions of all pebbles in it on layer l will be shifted by $h \bmod n_l$ for $0 \le l \le k$.*

Proof. Intuitively, this reflects the rotational symmetry of the funnel. To make this precise, we claim that for any $h \in \mathbb{N}$ and any protogadget π, the mapping $\phi_h(v_i^l) = v_{i'}^l$, where $i' = (i+h) \bmod n_l$, is an automorphism on π, i.e., a surjection on the vertices of π preserving labeled half-edges. Consider a forward edge labeled a at level l in π, say (v_i^l, v_j^{l+1}), where $j = \chi(i,a,l+1)$. Note that for each fixed a and l, there is a constant c (depending on a and l but independent of i) such that $\chi(i,a,l+1) = (i+c) \bmod n_{l+1}$. Now $\phi_h(v_i^l) = v_{i'}^l$, $\phi_h(v_j^{l+1}) = v_{j'}^{l+1}$, with $i' = (i+h) \bmod n_l$, and $j' = (j+h) \bmod n_{l+1}$, so since n_{l+1} divides n_l we have

$$\begin{aligned}
\chi(i',a,l+1) &= (i'+c) \bmod n_{l+1} \\
&= (((i+h) \bmod n_l) + c) \bmod n_{l+1} \\
&= (i+h+c) \bmod n_{l+1} \\
&= (((i+c) \bmod n_{l+1}) + h) \bmod n_{l+1} \\
&= (j+h) \bmod n_{l+1} \\
&= j'.
\end{aligned}$$

A similar argument applies to cross edges. □

The analog of Claim 3 also holds for gadgets, provided the τ moves use only forward and/or cross edges. The same may not be true if backward edge labels are used.

Global strategy. We have now described the gadgets and protogadgets and the adversary's strategy for building them. We turn to the remaining part of its strategy: charging and committing edges. Recall that the adversary associates a charge with each connecting edge, in which it counts moves in $G_{i,*}$. In addition, it associates with each connecting edge a second integer, called a *birthdate*, recording the time at which a pebble first crosses the edge.

The construction of $G_{i+1,0}$ from $G_{i,*}$ proceeds as follows. The adversary begins with M in its *initial* configuration in the current graph $G_{i,*}$. The adversary simulates successive moves of M on $G_{i,*}$ until some uncommitted connecting edge accumulates charge τ, where edge charges are determined by the following rules. During a move, suppose M moves pebble p along

1. an edge internal to a gadget or protogadget. Let f be the connecting edge most recently crossed by p. If f has charge less than τ, then charge the move to f; otherwise there is no charge.

2. a connecting edge f (committed or not). Charge the move to the oldest (i.e., least birthdate) connecting edge having charge less than τ. If this is the first step in which a pebble has crossed edge f in either direction, define the birthdate of f to be the current time.

As sketched in the overview, the second charging rule ensures that when an uncommitted connecting edge f, even one whose associated pebbles have moved infrequently, accumulates charge τ, only a few of the gadgets of the appropriate type can have been touched by pebbles since the birth of f.

When some uncommitted connecting edge $f = \{u, v\}$ with label $\lambda_{u,v} = a$ accumulates charge τ we stop the simulation and construct from $G_{i,*}$ a new graph $G_{i+1,0}$ defined as follows. Let π_v be the protogadget entered through f, with v in π_v. Note that by the charging rules above, each move in π_v has been charged to f, so there have been at most τ such moves. Thus by Claim 1 the adversary did not fail while building π_v. By Claim 2, the protogadget π_v is a subgraph of some gadget γ_v. We say an entry vertex of a gadget is *open* if it has degree less than d. If possible, choose an entry vertex x of a gadget in $G_{i,*}$ such that

(7)
- x is open,
- x's gadget is of the same type as γ_v,
- x and u are not adjacent, and
- x's gadget has remained pebble free since the birthdate of the uncommitted edge f.

$G_{i+1,0}$ is identical to $G_{i,*}$, except that the protogadget π_v is removed and the uncommitted edge $f = \{u, v\}$ is replaced by the committed edge $\{u, x\}$ with labels $\lambda_{u,x} = a$ and $\lambda_{x,u} = b$, where b is any label not already present on an outgoing half-edge at x. If there is no such x, or if using the only such x would result in $G_{i+1,0}$ having neither uncommitted edges nor open entry vertices, we instead add one additional gadget of type γ_v, choose as x any of the new gadget's entry vertices, then proceed as described above. The latter contingency avoids premature termination of the construction. The requirement that x and u be nonadjacent avoids construction of parallel edges.

The behavior of M on $G_{i+1,0}$ is similar to its behavior on $G_{i,*}$. Suppose in $G_{i,*}$ that the uncommitted edge f was first crossed during the simulation of the bth move of M (i.e., has birthdate b) and accumulates charge τ during move b'. When M is simulated on $G_{i+1,0}$, it will behave exactly as on $G_{i,j}$ for the first $b-1$ moves, since the portion of $G_{i+1,0}$ visited during that period is exactly the same as the portion visited in $G_{i,*}$. In particular, the charges and birthdates attached to edges will be the same. (Thus, one can view the adversary as rolling back the simulation to step b, committing f, and resuming.) Between steps b and b' those pebbles that crossed edge f in $G_{i,*}$ will be in x's gadget γ_x in $G_{i+1,0}$ instead of in the protogadget π_v entered through f as they were in $G_{i,*}$, but since γ_x contains π_v as a subgraph, their motions in $G_{i+1,0}$ will exactly reflect their motions in $G_{i,*}$. Note that by Claim 3 this is true regardless of which entry vertex x of γ_x was chosen. It is crucial that the chosen gadget γ_x was pebble free between steps b and b', so there is no possibility that these pebbles will meet pebbles in γ_x in $G_{i+1,0}$ that they did not meet in π_v in $G_{i,*}$. Again, the charges and birthdates attached to edges will be the same in $G_{i+1,0}$ as in $G_{i,*}$ through step b'. In particular, each of the $i+1$ committed edges in $G_{i+1,0}$ will have a charge of τ, and hence M will run for at least $(i+1)\tau$ steps on $G_{i+1,0}$.

Final Construction. After $G_{i+1,0}$ is built, we restart the simulation from the beginning on $G_{i+1,0}$ to build $G_{i+2,0}$, etc. Continue this process until $G_{\mu,0}$ is constructed. Finally, from $G_{\mu,0}$ we build a pair of similar graphs G and G', one connected and the other not, on which M will have identical behavior. In particular, if M runs for fewer than $\mu \cdot \tau$ steps, then M cannot be correct on both. The connected graph G is built by

1. committing all uncommitted edges, as described above;
2. joining the remaining open entry vertices with some number, Δ, of extra vertices so as to make G have n vertices and be d-regular; and
3. designating one of these extra vertices as t.

One way to accomplish the second step is the following. First, pick any two nonadjacent open vertices and connect them. Repeat this as often as possible. Let u be the number of "missing" half-edges, i.e., the total over all open vertices of d minus their degrees, and let i be the number of remaining open vertices. Since the pairing process could not be applied to reduce i further, it must be the case that the i open entry vertices form a clique. Recalling that each entry vertex is incident to at most $d - d'$ connecting edges, the number u of missing half-edges can be at most

$$i((d - d') - (i - 1)) \leq i(d - i) \leq d^2/4,$$

since $d' \geq 1$ and since $i(d-i)$ is maximized when $i = d/2$. Thus $u \leq d^2/4$. Furthermore, u will necessarily be even, since each entry vertex starts with $d-d'$ missing half edges; since from (4) $d - d'$ is a multiple of q, hence even; and since each committed edge replaces a pair of missing half-edges. Notice that this implies that $d(n - \Delta)$ is even, since the gadgets together contain $n - \Delta$ vertices and $d(n - \Delta) - u$ half-edges, which naturally occur in pairs. Complete the construction by adding a Δ-vertex, d-regular graph that contains a $u/2$-matching, removing the edges of this matching, and connecting each of the u missing half-edges to a distinct endpoint of the matching. Such a regular graph exists by Proposition 1, since dn, $d(n - \Delta)$, and hence $d\Delta$ are even; since, as shown below, $d < \Delta$ and $u \leq d^2/4 < \Delta$; and since the proof of Proposition 1 given in Borodin et al. [16] constructs a regular graph that is Hamiltonian and hence has a $u/2$-matching. (That construction is similar to the construction of cross edges in one layer of our gadgets, where the 0-labels form a Hamiltonian cycle.)

The nonconnected graph G' is built similarly, except that $d + 1$ of the Δ extra vertices, including t, are connected in a clique and hence are disconnected from the rest of the graph.

By an argument similar to the one above, M's behavior on both G and G' is essentially the same as on $G_{\mu,0}$. In particular, the edge charges will be the same, so M will run for at least $\mu \cdot \tau$ steps without reaching any of the Δ extra vertices, including t. One point to be shown in the analysis below is that $\Delta \geq d+1+\max(d+1, d^2/4) = d^2/4+d+1$, i.e., large enough to allow completion of the construction of G and G' as described above. Since $d \leq \sqrt{n} - 2$ (in fact, $d^2 + Pd = o(n)$), it suffices that $\Delta \geq n/4$.

Analysis. All that remains to show our $\Omega(m\tau)$ lower bound is to give values for the various parameters so as to satisfy the constraints listed above (and to maximize τ). For convenience, we summarize the relevant parameters and constraints here.

C1. Number of committed connecting edges: $\mu = \Omega(m)$.
C2. Number of vertices added in the final step of the construction: $\Delta \geq n/4$.
C3. Number of layers per gadget: $k \geq 1$.
C4. Size of full blocks in the label partition: $q \geq 2$, even.
C5. Number of full label blocks: $g = \lfloor d/q \rfloor$.
C6. Upper bound on the number of pebbles entering a protogadget: $1 \leq r \leq g/3$.
C7. Time per committed edge: τ; if $P \leq r$ then $\tau = (k - \lceil \log_2 k \rceil) \lfloor g/3 \rfloor$; else $\tau = r$.

To satisfy constraint C1, choose

$$\mu = \lfloor dLn/(8S) \rfloor . \tag{8}$$

Since we have seen in inequality (1) that $L/S = \Omega(1)$, we have $\mu = \Omega(m)$ as claimed above.

We now turn to constraint C2. We say a gadget is *closed* if each of its L entry vertices is connected to the maximum number $d-d'$ of committed half-edges; otherwise

the gadget is *open*. $G_{\mu,0}$ has exactly μ committed edges, or 2μ committed half-edges. From (4), each closed gadget contributes $(d-d')L \geq dL/2$ committed half-edges, so by (8) there can be no more than $2\mu/(dL/2) \leq n/(2S)$ closed gadgets in $G_{\mu,0}$, each of size S, and so closed gadgets contribute no more than $n/2$ vertices to G. Thus, to ensure constraint C2, i.e., that Δ is at least $n/4$, it suffices to ensure that the following additional constraint holds:

C8. Number of vertices in open gadgets: must be at most $n/4$.

When building $G_{i+1,0}$ from $G_{i,*}$, the adversary replaces a protogadget π by a copy of a fixed gadget γ. There might be many copies of the gadget γ with which π can be replaced. A key claim in establishing constraint C8 is that there are never more than $P(\tau+2)+d$ open copies of such a gadget.

CLAIM 4. *When $G_{i+1,0}$ is defined, if there are $P(\tau+2)+d$ open copies of the gadget γ_v, then at least one of them will have an entry vertex x satisfying the conditions* (7), *so a new (open) gadget will* not *be introduced into $G_{i+1,0}$.*

Proof. We show an upper bound on the number of open gadgets that are disqualified from containing x. It is easy to see that at most $d-d' \leq d-1$ entry vertices are adjacent to u in $G_{i,*}$. A more subtle problem is to bound the number of gadgets that can be touched by pebbles between the birth of π's uncommitted connecting edge f and the time at which f has accumulated charge τ. At most P gadgets contain pebbles at the time of f's birth. At most $P-1$ edges older than f can have charge less than τ, because, by Claim 1, for each such edge f' there is at least one pebble that does not leave its gadget or protogadget until f' has accumulated charge τ. Each gadget touched by some pebble after the birth of f necessitates the crossing of some connecting edge. Thus after at most $(P-1)\tau$ such crossings, f will be the oldest uncommitted edge, and after at most τ more crossings, f will have charge τ. Thus, at most $P(\tau+1)$ gadgets can be touched by pebbles during the relevant interval. Finally, at all times, at most P open gadgets are incident to uncommitted half-edges, hence at most P lack open entry vertices. Thus, the number of vertices x not disqualified is at least $P(\tau+2)+d-(d-1)-P(\tau+1)-P=1$, which establishes the claim. ∎

Inequality (5) bounds the number of distinct gadget types, Claim 4 bounds the number of open copies of each, and inequality (2) bounds the size of each copy. Thus, the total number of vertices in open gadgets is at most

$$\left(\frac{eg}{r}\right)^{r(2k+\lceil \log_2 k\rceil)} (P(\tau+2)+d)\, 6dk. \tag{9}$$

We divide the remainder of the analysis into two cases. The second applies when P is small. The first applies to either small or large P but gives a weaker bound than the second for small P.

Case 1. Let $\delta = 3\epsilon/2 = 1/(2\ln(6e))$, let

$$\beta = 72\left(d^2 + Pd\ln\frac{n}{d^2+Pd}\right),$$

and suppose n, P, and d are such that $d^2+Pd \leq n/e$ and $\beta \leq n/e^{6/\delta}$, both of which are true for all sufficiently large n, since $d^2+Pd = o(n)$. Then we claim that the following parameter values satisfy constraints C3–C8:

$$
\begin{aligned}
k &= 1, \\
q &= 2\left\lceil \frac{d-5}{\delta \ln(n/\beta)} \right\rceil, \\
g &= \lfloor d/q \rfloor, \\
r &= \lfloor g/3 \rfloor, \\
\tau &= r.
\end{aligned}
$$

Note that constraints C3 and C5 are immediately satisfied, as is constraint C7 since $k = 1$. It is also immediate that q is even and is positive, since $d \geq 6$, $\delta > 0$, and $n/\beta > 1$; hence constraint C4 is satisfied.

For constraint C6, it is immediate that $r \leq g/3$. To show $r \geq 1$ it suffices to show $q \leq d/3$:

$$
q = 2\left\lceil \frac{d-5}{\delta \ln(n/\beta)} \right\rceil \leq 2\left\lceil \frac{d-5}{6} \right\rceil = 2\left\lfloor \frac{d}{6} \right\rfloor \leq \frac{d}{3}.
$$

To satisfy constraint C8 above, we first note (making frequent use of the inequalities $x/2 \leq \lfloor x \rfloor$ and $\lceil x \rceil \leq 2x$, valid for all $x \geq 1$) that

$$
\begin{aligned}
g/r = g/\lfloor g/3 \rfloor &\leq g/(g/6) = 6, \\
r = \lfloor g/3 \rfloor &\leq g/3 = \lfloor d/q \rfloor /3 \leq d/(3q) \\
&= \frac{d}{6\left\lceil \frac{d-5}{\delta \ln(n/\beta)} \right\rceil} \leq \frac{1}{6}\frac{d}{d-5}\delta \ln(n/\beta) \leq \delta \ln(n/\beta), \\
r + 2 &\leq 3r, \\
d^2 + Pd &\leq d^2 + Pd \ln \frac{n}{d^2 + Pd} < \beta, \text{ and} \\
\delta &= 1/(2\ln(6e)) < 1.
\end{aligned}
$$

Returning to constraint C8, we must show that (9) is at most $n/4$:

$$
\begin{aligned}
\left(\frac{eg}{r}\right)^{r(2k+\lceil \log_2 k \rceil)} &(P(\tau+2)+d)\,6dk \\
&= 6\left(\frac{eg}{r}\right)^{2r}(d^2 + Pd(r+2)) \\
&\leq 18(6e)^{2r}(d^2 + Pdr) \\
&\leq 18(6e)^{2\delta \ln(n/\beta)}(d^2 + \delta Pd \ln(n/\beta)) \\
&= 18(n/\beta)(d^2 + \delta Pd \ln(n/\beta)) \\
&= \frac{18n(d^2 + \delta Pd \ln(n/\beta)}{72\left(d^2 + Pd \ln \frac{n}{d^2+Pd}\right)} \\
&< n/4,
\end{aligned}
$$

as desired.

To complete the analysis of Case 1, we show that τ is large enough to imply the bound in the statement of the theorem:

$$
\begin{aligned}
\tau = r = \lfloor g/3 \rfloor &\geq g/6 = \lfloor d/q \rfloor /6 \geq d/(12q) \\
&= \frac{d}{24\left\lceil \frac{d-5}{\delta \ln(n/\beta)} \right\rceil}.
\end{aligned}
$$

The latter quantity equals $d/24$, if $d \le \delta \ln(n/\beta) + 5$. If $d > \delta \ln(n/\beta) + 5$, then

$$\begin{aligned}
\frac{d}{24\left\lceil \frac{d-5}{\delta \ln(n/\beta)} \right\rceil} &\ge \frac{d}{48\left(\frac{d-5}{\delta \ln(n/\beta)}\right)} \\
&= \frac{1}{48}\frac{d}{d-5}\delta \ln(n/\beta) \\
&\ge \frac{\delta \ln(n/\beta)}{48} \\
&= \frac{\delta}{48} \ln \frac{n}{72\left(d^2 + Pd \ln \frac{n}{d^2+Pd}\right)} \\
&\ge \frac{\delta}{48} \ln \frac{n}{72\,(d^2 + Pd) \ln \frac{n}{d^2+Pd}} \\
&= \Omega\left(\ln \frac{n}{d^2 + Pd}\right).
\end{aligned}$$

The penultimate inequality holds since, by assumption, $\ln(n/(d^2+Pd)) \ge 1$. The final lower bound follows since $\ln(x/(72 \ln x)) = \Omega(\ln x)$. Thus, as claimed in the statement of the theorem, $\tau = \Omega(\min(d, \ln(n/(d^2 + Pd))))$.

Case 2. Recall $\epsilon = 1/(3\ln(6e))$ and suppose $6P \le d \le \sqrt{n}/6^9$ and $1 \le P \le \epsilon \ln(n/d^2)$. (Note that $d \le \sqrt{n}/6^9$ must be true for all sufficiently large n, since $d^2+Pd = o(n)$.) Then we claim that the following parameter values satisfy constraints C3–C8:

$$\begin{aligned}
\hat{q} &= \frac{ed}{P}\left(\frac{d^2}{n}\right)^{1/(3P)}, \\
q &= 2\left\lceil \hat{q}/2 \right\rceil, \\
g &= \lfloor d/q \rfloor, \\
r &= P, \\
k &= \left\lfloor \frac{\ln(n/d^2)}{3P \ln(ed/(qP))} \right\rfloor, \\
\tau &= (k - \lceil \log_2 k \rceil)\, \lfloor g/3 \rfloor = \Theta(gk).
\end{aligned}$$

Note that constraint C5 is immediately satisfied, as is constraint C7 since $r \ge P$. Since $\hat{q}$ is positive, it is also immediate that q is even and is positive, hence constraint C4 is satisfied.

Note for future use that

$$(n/d^2)^{1/(3P)} \ge (n/d^2)^{1/(3\epsilon \ln(n/d^2))} = e^{1/(3\epsilon)} = 6e. \tag{10}$$

For constraint C3, note that $q \ge \hat{q}$. Thus,

$$k = \left\lfloor \frac{\ln((n/d^2)^{1/(3P)})}{\ln(ed/(qP))} \right\rfloor \ge \left\lfloor \frac{\ln((n/d^2)^{1/(3P)})}{\ln(ed/(\hat{q}P))} \right\rfloor = \left\lfloor \frac{\ln((n/d^2)^{1/(3P)})}{\ln((n/d^2)^{1/(3P)})} \right\rfloor = 1.$$

Thus $k \ge 1$. Using a similar analysis, we note for future use that $k = 1$ whenever $q > 2$. This holds since $q > 2$ implies $\hat{q}/2 > 1$, which implies $q \le 2\hat{q}$. Thus,

$$k = \left\lfloor \frac{\ln((n/d^2)^{1/(3P)})}{\ln(ed/(qP))} \right\rfloor \le \left\lfloor \frac{\ln((n/d^2)^{1/(3P)})}{\ln(ed/(2\hat{q}P))} \right\rfloor = \left\lfloor \frac{\ln((n/d^2)^{1/(3P)})}{\ln((n/d^2)^{1/(3P)}/2)} \right\rfloor = 1. \tag{11}$$

The last equality follows from the fact that $1 < (\ln x)/(\ln(x/2)) < 2$ whenever $x > 4$, and from (10).

For constraint C6, it is immediate that $r = P \geq 1$. To show $r \leq g/3$ it suffices to show $3qP \leq d$. If $q = 2$, this holds, since by assumption $6P \leq d$. If $q > 2$, then $\hat{q}/2 > 1$, so

$$3qP = 6\lceil \hat{q}/2 \rceil P \leq 6\hat{q}P = 6(ed/P)(d^2/n)^{1/(3P)}P \leq 6ed/(6e) = d. \tag{12}$$

The last inequality follows from (10).

For constraint C8, we first note for integer $k \geq 1$ that

$$2k + \lceil \log_2 k \rceil \leq 8k/3.$$

(The bound is tight at $k = 3$.) Also,

$$\tau + 2 = (k - \lceil \log_2 k \rceil)\lfloor g/3 \rfloor + 2 \leq gk,$$

since $g \geq 3$. Using (9), we bound the number of vertices in open gadgets as follows.

$$\begin{aligned}
&\left(\frac{eg}{r}\right)^{r(2k+\lceil \log_2 k \rceil)} (P(\tau+2)+d)\, 6dk \\
&\qquad \leq 6\left(\frac{eg}{P}\right)^{8Pk/3} dk(Pgk + d) \\
&\qquad \leq 6\left(\frac{ed}{qP}\right)^{(8/3)P\left\lfloor \frac{\ln(n/d^2)}{3P\ln(ed/(qP))}\right\rfloor} dk(Pdk/q + d) \\
&\qquad \leq 6\left(\frac{n}{d^2}\right)^{8/9} d^2(Pk^2/q + k) \\
&\qquad \leq 6n^{8/9}d^{2/9}(Pk^2/q + k).
\end{aligned}$$

We break the rest of the derivation of constraint C8 into two subcases based on d. Note that, since $3qP \leq d$ from (12),

$$k = \left\lfloor \frac{\ln(n/d^2)}{3P\ln(ed/(qP))} \right\rfloor \leq \frac{\ln n}{3P\ln(3e)} \leq \ln n.$$

When $d < n^{1/4}$, since k and P are both $O(\log n)$, we have

$$6n^{8/9}d^{2/9}(Pk^2/q + k) = O(n^{8/9}(n^{1/4})^{2/9}\log^3 n) = O(n^{17/18}\log^3 n) = o(n).$$

When $d \geq n^{1/4}$, we will show that $q > 2P$ and $k = 1$, so we have

$$6n^{8/9}d^{2/9}(Pk^2/q + k) \leq 6n^{8/9}(n^{1/2}/6^9)^{2/9}(1/2 + 1) = n/4.$$

We show that $q > 2P$ as follows.

$$\frac{q}{P} \geq \frac{\hat{q}}{P} = \frac{ed}{P^2}\left(\frac{d^2}{n}\right)^{1/(3P)} \geq \frac{n^{1/4}}{P^2}\left(\frac{n^{2/4}}{n}\right)^{1/3} = \frac{n^{1/12}}{P^2} = \omega(1).$$

(Recall $P = O(\log n)$.) Since $q > 2P$ and $P \geq 1$, we have $q > 2$, so $k = 1$ by (11).

To complete the analysis of Case 2, we show that $\tau = (k - \lceil \log_2 k \rceil)\lfloor g/3 \rfloor$ is large enough to imply the bound in the statement of the theorem. Again we split the

analysis into two subcases based on d. We have $q > 2$ if and only if $\hat{q} > 2$, which holds exactly when

$$d > d_0 = (2P/e)^{3P/(3P+2)}\, n^{1/(3P+2)}.$$

In this case we have $k = 1$ by (11) and

$$\begin{aligned}\tau = \lfloor g/3 \rfloor &\geq g/6 \geq d/(12q) \geq d/(24\hat{q}) = \frac{d}{24\left(\frac{ed}{P}\left(\frac{d^2}{n}\right)^{1/(3P)}\right)} \\ &= \frac{P}{24e}\left(\frac{n}{d^2}\right)^{1/(3P)} \qquad (13) \\ &= \Omega\left(P\left(\frac{n}{d^2}\right)^{1/(3P)}\right),\end{aligned}$$

as claimed. We remark that, by (10), when P is maximal, (13) is $P/4 = \Theta(\log(n/d^2))$, so the transition to the bound given in Case 1 is "smooth."

In the second subcase we have $d \leq d_0$. First, note that

$$d_0 = (2P/e)^{3P/(3P+2)}\, n^{1/(3P+2)} \leq P n^{1/(3P+2)} \leq P n^{1/5} = o(n^{1/4}). \qquad (14)$$

Second, since $d \leq d_0$, we have $q = 2$ and $k \geq 1$. Also, note that $(k - \lceil \log_2 k \rceil)/k \geq 1/3$, (attaining the minimum at $k = 3$) and that $g \geq 3$. Hence $\tau = \Omega(gk)$ and

$$\begin{aligned} gk &= \left\lfloor \frac{d}{q} \right\rfloor \left\lfloor \frac{\ln(n/d^2)}{3P \ln(ed/(qP))} \right\rfloor \\ &\geq \frac{d \ln(n/d^2)}{24P \ln(ed/(2P))} \\ &= \frac{d \ln(n/o(n^{1/2}))}{24P \ln(ed/(2P))} \\ &= \Omega\left(\frac{d/P}{\ln(d/P)} \ln n\right), \end{aligned}$$

as claimed. We remark that when P is maximal and $d \geq 6P$, the estimate in (14) can be refined, allowing one to show $d = \Theta(P) = \Theta(\log n)$. Thus, τ again matches the bound in Case 1 (up to constant factors).

Finally, when $d = d_0$ we have $\hat{q}$ exactly equal to 2; similarly, when $d = d_0$, the expression of which k is the floor is precisely 1. Furthermore, both expressions vary slowly with d, so both are $\Theta(1)$ when d is near d_0. Thus, again $\tau = (k - \lceil \log_2 k \rceil)\, \lfloor \lfloor d/q \rfloor /3 \rfloor$ is "smooth" as d crosses d_0, the threshold between the lower bounds quoted in (a) and (b) in the statement of the theorem, and in fact both lower bounds are $\Theta(md_0)$ for d near d_0.

This completes the proof. □

It is interesting to note why the proof would fail if M were allowed to jump pebbles. In the local phase, the adversary was able to pick an existing gadget in which p must invest τ steps. In the presence of jumping, this fails, since p can always jump out of the new gadget. As a particular foil to the proof above, imagine an automaton that stations one pebble p on an entry vertex of some gadget, and successively moves a second pebble q to each neighbor, jumping q back to p to find the next neighbor. In time $\Theta(d)$, this has touched all $\Theta(d)$ connecting edges incident to that entry vertex, which was impossible in the construction above.

4. Open problem. The obvious important problem is to strengthen and generalize these lower bounds. Following an earlier version of this paper [9], Edmonds [21] proved a much stronger time-space tradeoff on general JAGs: for every $z \geq 2$, a JAG with at most $\frac{1}{28z}\frac{\log n}{\log\log n}$ pebbles and at most $2^{\log^z n}$ states requires time $n \cdot 2^{\Omega((\log n)/(\log\log n))}$ to traverse 3-regular graphs. The ultimate goal might be to prove that $ST = \Omega(mn)$ for JAGs or even for general models of computation.

Acknowledgment. We thank Michael Sipser for showing us the construction generalized in section 3.

REFERENCES

[1] L. M. Adleman, *Two theorems on random polynomial time*, in 19th Annual IEEE Symposium on Foundations of Computer Science, Ann Arbor, MI, 1978, pp. 75–83.

[2] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. W. Rackoff, *Random walks, universal traversal sequences, and the complexity of maze problems*, in 20th Annual IEEE Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1979, pp. 218–223.

[3] N. Alon, Y. Azar, and Y. Ravid, *Universal sequences for complete graphs*, Discrete Appl. Math., 27 (1990), pp. 25–28.

[4] A. Bar-Noy, A. Borodin, M. Karchmer, N. Linial, and M. Werman, *Bounds on universal sequences*, SIAM J. Comput., 18 (1989), pp. 268–277.

[5] G. Barnes, J. F. Buss, W. L. Ruzzo, and B. Schieber, *A sublinear space, polynomial time algorithm for directed s-t connectivity*, SIAM J. Comput., 27 (1998), pp. 1273–1282.

[6] G. Barnes and J. A. Edmonds, *Time-space lower bounds for directed s-t connectivity on JAG models*, in Proceedings 34th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 228–237.

[7] G. Barnes and U. Feige, *Short random walks on graphs*, SIAM J. Disc. Math., 9 (1996), pp. 19–28.

[8] G. Barnes and W. L. Ruzzo, *Undirected s-t connectivity in polynomial time and sublinear space*, Comput. Complexity, 6 (1996/1997), pp. 1–28.

[9] P. W. Beame, A. Borodin, P. Raghavan, W. L. Ruzzo, and M. Tompa, *Time-space tradeoffs for undirected graph traversal*, in Proc. 31st Annual IEEE Symposium on Foundations of Computer Science, St. Louis, MO, 1990, pp. 429–438.

[10] P. W. Beame, A. Borodin, P. Raghavan, W. L. Ruzzo, and M. Tompa, *Time-space tradeoffs for undirected graph traversal by graph automata*, Inform. and Comput., 130 (1996), pp. 101–129.

[11] P. Berman and J. Simon, *Lower bounds on graph threading by probabilistic machines*, in 24th Annual IEEE Symposium on Foundations of Computer Science, Tucson, AZ, 1983, pp. 304–311.

[12] M. Blum and D. C. Kozen, *On the power of the compass (or, why mazes are easier to search than graphs)*, in 19th Annual IEEE Symposium on Foundations of Computer Science, Ann Arbor, MI, 1978, pp. 132–142.

[13] M. Blum and W. J. Sakoda, *On the capability of finite automata in* 2 *and* 3 *dimensional space*, in 18th Annual IEEE Symposium on Foundations of Computer Science, Providence, RI, 1977, pp. 147–161.

[14] A. Borodin, *Structured vs. general models in computational complexity*, L'Enseignement Mathématique, XXVIII (1982), pp. 171–190. Also in [27, pp. 47–65].

[15] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa, *Two applications of inductive counting for complementation problems*, SIAM J. Comput., 18 (1989), pp. 559–578. See also 18 (1989), p. 1283.

[16] A. Borodin, W. L. Ruzzo, and M. Tompa, *Lower bounds on the length of universal traversal sequences*, J. Comput. System Sci., 45 (1992), pp. 180–203.

[17] M. F. Bridgland, *Universal traversal sequences for paths and cycles*, J. Algorithms, 8 (1987), pp. 395–404.

[18] A. Z. Broder, A. R. Karlin, P. Raghavan, and E. Upfal, *Trading space for time in undirected s-t connectivity*, SIAM J. Comput., 23 (1994), pp. 324–334.

[19] J. Buss and M. Tompa, *Lower bounds on universal traversal sequences based on chains of length five*, Inform. Comput., 120 (1995), pp. 326–329.

[20] S. A. Cook and C. W. Rackoff, *Space lower bounds for maze threadability on restricted machines*, SIAM J. Comput., 9 (1980), pp. 636–652.

[21] J. A. Edmonds, *Time-space trade-offs for undirected ST-connectivity on a JAG*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 718–727.

[22] J. A. Edmonds and C. K. Poon, *A nearly optimal time-space lower bound for directed st-connectivity on the NNJAG model*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 147–156.

[23] U. Feige, *A spectrum of time-space trade-offs for undirected s-t connectivity*, J. Comput. System Sci., 54 (1997), pp. 305–316.

[24] A. Hemmerling, *Labyrinth Problems: Labyrinth-Searching Abilities of Automata*, Teubner-Texte Math. 114, Teubner, Leipzig, 1989.

[25] S. Istrail, *Polynomial universal traversing sequences for cycles are constructible*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, 1988, pp. 491–503.

[26] H. J. Karloff, R. Paturi, and J. Simon, *Universal traversal sequences of length $n^{O(\log n)}$ for cliques*, Inform. Proc. Lett., 28 (1988), pp. 241–243.

[27] *Logic and Algorithmic*, An International Symposium Held in Honor of Ernst Specker, Zürich, Feb. 5–11, 1980, Monographie No. 30 de L'Enseignement Mathématique, Université de Genève, Switzerland, 1982.

[28] N. Nisan, $RL \subseteq SC$, Comput. Complexity, 4 (1994), pp. 1–11.

[29] N. Nisan, E. Szemerédi, and A. Wigderson, *Undirected connectivity in $O(\log^{1.5} n)$ space*, in Proceedings 33rd Annual IEEE Symposium on Foundations of Computer Science, Pittsburgh, PA, 1992, pp. 24–29.

[30] W. J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.

[31] W. J. Savitch, *Maze recognizing automata and nondeterministic tape complexity*, J. Comput. System Sci., 7 (1973), pp. 389–403.

[32] M. Tompa, *Two familiar transitive closure algorithms which admit no polynomial time, sublinear space implementations*, SIAM J. Comput., 11 (1982), pp. 130–137.

[33] M. Tompa, *Lower bounds on universal traversal sequences for cycles and other low degree graphs*, SIAM J. Comput., 21 (1992), pp. 1153–1160.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 1073–1085

ON THE APPROXIMABILITY OF NUMERICAL TAXONOMY (FITTING DISTANCES BY TREE METRICS)*

RICHA AGARWALA†, VINEET BAFNA‡, MARTIN FARACH§, MIKE PATERSON¶, AND MIKKEL THORUP‖

Abstract. We consider the problem of fitting an $n \times n$ distance matrix D by a tree metric T. Let ε be the distance to the closest tree metric under the L_∞ norm; that is, $\varepsilon = \min_T\{\| T-D \| \infty\}$. First we present an $O(n^2)$ algorithm for finding a tree metric T such that $\| T-D \| \infty \leq 3\varepsilon$. Second we show that it is $\mathcal{NP}$-hard to find a tree metric T such that $\| T-D \| \infty < \frac{9}{8}\varepsilon$. This paper presents the first algorithm for this problem with a performance guarantee.

Key words. approximation algorithm, tree metric, taxonomy

AMS subject classifications. 62P10, 68Q25, 92B10, 92-08

PII. S0097539795296334

1. Introduction. One of the most common methods for clustering numeric data involves fitting the data to a *tree metric*, which is defined by a weighted tree spanning the points of the metric, the distance between two points being the sum of the weights of the edges of the path between them. Not surprisingly, this problem, the so-called numerical taxonomy problem, has received a great deal of attention (see [2, 7, 8] for extensive surveys) with work dating as far back as the beginning of the century [1]. Fitting distances by trees is an important problem in many areas. For example, in statistics, the problem of clustering data into hierarchies is exactly the tree fitting problem. In "historical sciences" such as paleontology, historical linguistics, and evolutionary biology, tree metrics represent the branching processes which lead to some observed distribution of data. Thus, the numerical taxonomy problem has been, and continues to be, the subject of intense research.

In particular, consider the case of evolutionary biology. By comparing the DNA sequences of pairs of species, biologists get an estimate of the evolutionary time which has elapsed since the species separated by a speciation event. A table of pairwise distances is thus constructed. The problem is then to reconstruct the underlying evolutionary tree. Dozens of heuristics for this problem appear in the literature every year (see, e.g., [8]).

The numerical taxonomy problem is usually cast in the following terms. Let S

*Received by the editors December 12, 1995; accepted for publication (in revised form) April 8, 1997; published electronically January 29, 1999. A preliminary version of this paper appeared in Symposium on Discrete Mathematics '96.

http://www.siam.org/journals/sicomp/28-3/29633.html

†DIMACS, Rutgers University, Piscataway, NJ 08855 (richa@helix.nih.gov). This research was supported by Special Year National Science Foundation grant BIR-9412594. Current address: National Human Genome Research Institute/National Institutes of Health, Bethesda, MD 20892.

‡DIMACS, Rutgers University, Piscataway, NJ 08855 (bafna@dimacs.rutgers.edu). This research was supported by Special Year National Science Foundation grant BIR-9412594. Current address: Bioinformatics, UW2230, SmithKline Beecham, 709 Swedeland Road, King of Prussia, PA 19406.

§Department of Computer Science, Rutgers University, Piscataway, NJ 08855 (farach@cs.rutgers.edu). This research was supported by NSF Career Development Award CCR-9501942.

¶Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom (msp@dcs.warwick.ac.uk). This research was supported in part by the ESPRIT LTR project 20244—ALCOM–IT.

‖Department of Computer Science, University of Copenhagen, Universitetsparken 1, 2100 Copenhagen Ø, Denmark (mthorup@diku.dk). This work was done while visiting DIMACS.

be the set of species under consideration.

The numerical taxonomy problem.
Input: $D : S^2 \to \Re_{\geq 0}$, a distance matrix.
Output: A *tree metric* T which spans S and *fits* D.

This definition leaves two points unanswered: first, what kind of tree metric, and, second, what does it mean for a metric to fit D? Typically we are talking about any tree metric, but sometimes we want to restrict ourselves to *ultrametrics* defined by rooted trees where the distance to the root is the same for all points in S. In order to distinguish specific types of tree metrics, such as ultrametrics, from the general case, we will refer to unrestricted tree metrics as *additive* metrics. There may be no tree metric coinciding exactly with D, so by "fitting" we mean approximating D under norms such as L_1, L_2, or L_∞. That is, for $k = 1, 2, \ldots, \infty$, we want to find a tree metric T minimizing $\| T - D \|_k$ ($\| T - D \|_k$ is formally defined in Definition 2.6).

History. The numerical taxonomy problem for additive metric fitting under L_k norms was explicitly stated in its current form in 1967 [4]. Since then it has collected an extensive literature. In 1977 [10], it was shown that if there is a tree metric T coinciding exactly with D, it is unique and constructible in linear, i.e., $O(|S|^2)$, time. Unfortunately there is typically no tree metric coinciding exactly with D, and in 1987 [5], it was shown that for L_1 and L_2, the numerical taxonomy problem is $\mathcal{NP}$-hard, both in the additive and in the ultrametric cases. Additional complexity results appear in [9].

The only positive fitting result is from 1995 [6] and shows that under the L_∞ norm an optimal ultrametric is polynomially computable, in fact, in linear time. However, while ultrametrics have interesting special case applications, the fundamental problem in the area of numerical taxonomy is that of fitting D by general tree metrics. Unfortunately no provably good algorithms existed for fitting distances by additive metrics, and in [6] the numerical taxonomy problem for general tree metrics under the L_∞ norm was posed as a major open problem.

Our results. We consider the numerical taxonomy problem for additive metrics under the L_∞ norm as suggested in [6]. Let ε be the distance to the closest additive metric under the L_∞ norm, that is, $\varepsilon = \min_T\{\| T - D \|_\infty\}$. First we present an $O(n^2)$ algorithm for finding an additive metric T such that $\| T - D \|_\infty \leq 3\varepsilon$. We complement this result not only by finding that an L_∞-optimal solution is $\mathcal{NP}$-hard, but we also rule out arbitrarily close approximations by showing that it is $\mathcal{NP}$-hard to find an additive metric T such that $\| T - D \|_\infty < \frac{9}{8}\varepsilon$.

Our algorithm is achieved by transforming the general tree metric problem to that of ultrametrics with a loss of a factor of 3 on the approximation ratio. Since the ultrametric problem is optimally solvable, our first result follows. We also generalize our transformation from the general tree metric to ultrametrics under any L_k norm with the same loss of a factor of 3.

The paper is organized as follows. After some preliminary definitions in section 2, we give our 3-approximation algorithm in section 3. We show in section 4 that our analysis is tight and that some natural "improved" heuristics do not help in the worst case. In section 5, we give our $\mathcal{NP}$-completeness and nonapproximability proofs. Finally, in section 6, we generalize our reduction from L_∞ to L_k norms with finite k.

2. Preliminaries. We present some basic definitions.

DEFINITION 2.1. *A* metric *on a set* $S = \{1, \ldots, n\}$ *is a function* $D : S^2 \to \Re_{\geq 0}$ *such that*

- $D[x, y] = 0 \iff x = y$,

- $D[x,y] = D[y,x]$,
- $D[x,y] \le D[x,z] + D[z,y]$ (*the triangle inequality*).

Likewise, $D : S^2 \to \Re_{\ge 0}$ is a *quasimetric* if it satisfies the first two conditions. For (quasi) metrics A and B, $A+B$ is the usual matrix addition, i.e., $(A+B)[i,j] = A[i,j] + B[i,j]$.

DEFINITION 2.2. *A (quasi) metric D is* (quasi) additive *if, for all points a,b,c,d,*

$$D[a,b] + D[c,d] \le \max\{D[a,c] + D[b,d], D[a,d] + D[b,c]\}.$$

This inequality is known as the 4-*point condition.*

THEOREM 2.3 (see [3]). *A metric is additive if and only if it is a tree metric.*

DEFINITION 2.4. *A metric D is an* ultrametric *if, for all points a,b,c,*

$$D[a,b] \le \max\{D[a,c], D[b,c]\}.$$

As noted above, an ultrametric is a type of tree metric.

DEFINITION 2.5. *A quasimetric D on n objects is a* centroid *quasimetric if $\exists l_1, \ldots, l_n$ such that $\forall i \ne j$, $D[i,j] = l_i + l_j$.*

A centroid quasimetric D is a *centroid metric* if $l_i \ge 0$ for all i. A centroid metric is a type of tree metric since it can be realized by a weighted tree with a star topology and edge weights l_i.

The k-norms are formally defined as follows.

DEFINITION 2.6. *For $n \times n$ real-valued matrices M and $k \ge 1$, define the k-norm, sometimes denoted L_k, by*

$$\| M \|_k = \left(\sum_{i<j} | M[i,j] |^k \right)^{\frac{1}{k}},$$

$$\| M \|_\infty = \max_{i<j} \{ | M[i,j] | \}.$$

3. Upper bound. Let $m_a = \max_i\{D[a,i]\}$. Let C^a be the centroid metric with $l_i = m_a - D[a,i]$, i.e., $C^a[i,j] = l_i + l_j = 2m_a - D[a,i] - D[a,j]$.

LEMMA 3.1 (see [2, Theorem 3.2]). *For any point a, D is quasiadditive if and only if $D + C^a$ is an ultrametric.*

LEMMA 3.2 (see [2, Corollary 3.3]). *Given an additive metric A and a centroid quasi-metric Q, $A+Q$ is additive if and only if $A+Q$ satisfies the triangle inequality.*

Let D be a distance matrix and let $\mathcal{X}$ be the set of all additive metrics. We define $\mathcal{A}(D)$ to be (one of) the additive metrics such that

$$\| D - \mathcal{A}(D) \|_\infty = \min_{A \in \mathcal{X}} \| D - A \|_\infty.$$

For point a, we say a metric M is *a-restricted* if $\forall i, M[a,i] = D[a,i]$. Let $\mathcal{X}^a$ be the set of a-restricted additive metrics. We define $\mathcal{A}^a(D)$ to be (one of) the a-restricted additive metrics such that $\| D - \mathcal{A}^a(D) \|_\infty = \min_{A \in \mathcal{X}^a} \| D - A \|_\infty$. In other words, $\mathcal{A}^a(D)$ is an optimal a-restricted additive metric for D. We will sometimes refer to such a metric as *a-optimal*. Similarly, we define $\mathcal{U}(D)$ to be an optimal ultrametric for D. Note that the functions $\mathcal{A}(), \mathcal{A}^a()$, and $\mathcal{U}()$ need not be uniquely valued. In

the following, we will let the output be an arbitrary optimal metric, unless otherwise noted. Recall that $\mathcal{U}()$ is computable in $O(n^2)$ time [6].

Lemma 3.1 suggests that we may be able to approximate the closest additive metric to D by approximating the closest ultrametric to $D + C^a$, i.e., by computing $\mathcal{U}(D + C^a) - C^a$ for some point a. Lemma 3.2 tells us that we need to guarantee the triangle inequality for the final metric to show that it is additive. Thus we need to modify our heuristic. Specifically, for any point a, we will show that $\| D - \mathcal{A}^a(D) \|_\infty \leq 3\| D - \mathcal{A}(D) \|_\infty$, and we will give a modification $\mathcal{U}^a()$ of $\mathcal{U}()$ such that $\mathcal{A}^a(D) = \mathcal{U}^a(D + C^a) - C^a$. We will use the following result implicit in [6].

THEOREM 3.3. *Consider two $n \times n$ distance matrices $L, M : S^2 \to \Re_{>0}$ and a real value h such that $L[i,j] \leq M[i,j] \leq h$ for all i, j. There is an $O(n^2)$ algorithm to compute an ultrametric U, if it exists, such that for all i, j, $L[i,j] \leq U[i,j] \leq h$, and $\| M - U \|_\infty$ is minimized.*

Proof. Our proof uses the construction of Theorem 5 in [6]. First we show how, given a distance matrix $A : S^2 \to \Re_{>0}$, we can construct in time $O(n^2)$ an ultrametric U, such that $U \leq A$ (i.e., $\forall i, j : U[i,j] \leq A[i,j]$) and such that for any ultrametric $U' \leq A$, $U' \leq U$.

Let T be a minimum spanning tree over the graph defined by A. The ultrametric U is now defined as follows. Let $e = (i,j)$ be the maximum weight edge of T, and let T_1 and T_2 be the subtrees of T obtained by deleting (i,j). Then U has root at height $A[i,j]/2$ and the subtrees of the root are the ultrametric trees U_1 and U_2 recursively defined on T_1 and T_2. Clearly, $U \leq A$.

CLAIM 3.3A. *For any ultrametric U', if $U' \leq A$ then $U' \leq U$.*

Proof. Let S_1 and S_2 be the partition of S defined by T_1 and T_2. By induction, for $k = 1, 2$, $U_k \geq U'|S_k^2$.

Let U_1' and U_2' be the two subtrees of U, and let S_1' and S_2' be the corresponding partitioning of S. Set $w = A[i,j]$ and $w' = \min_{(i,j) \in S_1' \times S_2'} A[i,j]$. Since w is the maximum weight in the minimum spanning tree T, $w' \leq w$. However, it is required that $U' \leq A$, so the height of the root of U' is $w'/2$, that is, the maximal distance in U' is w'. Thus for all $(i,j) \in S_1 \times S_2$, $U[i,j] = w \geq w' \geq U'[i,j]$. ∎

Consider an ultrametric U' as described in Theorem 3.3; i.e. for all i, j, $L[i,j] \leq U'[i,j] \leq h$, and $\varepsilon = \| M - U' \|_\infty$ is minimized. Set

$$\varepsilon^+ = \max_{i,j \in S}, (M[i,j] - U'[i,j]) \leq \varepsilon.$$

Suppose that we knew ε^+. Define A^{ε^+} such that $A^{\varepsilon^+}[i,j] = \min\{M[i,j] + \varepsilon^+, h\}$, and construct T^{ε^+} and $U^{\varepsilon^+} \leq A^{\varepsilon^+}$ as described above. Since $U' \leq A^{\varepsilon^+}$, $U^{\varepsilon^+}[i,j] \geq U'[i,j]$, so $L[i,j] \leq U^{\varepsilon^+}[i,j]$ and $\| M - U^{\varepsilon^+} \|_\infty \leq \| M - U' \|_\infty$.

Now observe that if T is a minimum spanning tree for M then T is also a minimum spanning tree for A^{ε^+}. Thus it follows that the topology of an optimal ultrametric U may be the same as the one we would construct from T and M. Given that T defines the right topology, we can construct the optimal ultrametric as follows.

Let $e = (i,j)$ be the maximum M-weight edge of T, and let T_1 and T_2 be the subtrees of T obtained by deleting (i,j). Let S_1 and S_2 be the partition of S defined by T_1 and T_2. Set

$$\mu = \frac{\max_{(i,j) \in S^2} M[i,j] + \min_{(i,j) \in S_1 \times S_2} M[i,j]}{2}.$$

Then U has root at height $\min\{h, \mu\}/2$ and the subtrees of the root are the ultrametric trees U_1 and U_2 recursively defined on T_1 and T_2. ∎

3.1. The L_∞ approximation. The *stem* of a leaf is the edge incident on it.

LEMMA 3.4. *For all points a,* $\| D - \mathcal{A}^a(D) \|_\infty \leq 3\| D - \mathcal{A}(D) \|_\infty$.

Proof. For all i, j, let $\varepsilon[i,j] = \mathcal{A}(D)[i,j] - D[i,j]$, and $\varepsilon = \max_{i,j}\{|\varepsilon[i,j]|\}$. Derive an a-restricted tree $T^{/a}$ from $\mathcal{A}(D)$ as follows. We will move all i either toward or away from a until each i is distance $D[a,i]$ from a. If $\mathcal{A}(D)[a,i] - D[a,i]$ is negative, we simply increase the length of its stem. Otherwise, i must be moved closer to a. Consider the (weighted) path from i to a. Let p be the point on this path which is distance $D[a,i]$ from a. We simply move i to this location. In either case, no point i is moved more than $|\varepsilon[a,i]|$, and so $|\mathcal{A}(D)[i,j] - T^{/a}[i,j]| \leq |\varepsilon[a,i]| + |\varepsilon[a,j]|$. Now, $T^{/a}$ is additive by construction, and for all i, $T^{/a}[a,i] = D[a,i]$. Further, for all i, j,

$$\begin{aligned} |D[i,j] - T^{/a}[i,j]| &\leq |\mathcal{A}(D)[i,j] - T^{/a}[i,j]| + |D[i,j] - \mathcal{A}(D)[i,j]| \\ &\leq (|\varepsilon[a,i]| + |\varepsilon[a,j]|) + |\varepsilon[i,j]| \\ &\leq 3\varepsilon. \end{aligned}$$

Finally, by the optimality of $\mathcal{A}^a(D)$,

$$\| D - \mathcal{A}^a(D) \|_\infty \leq \| D - T^{/a} \|_\infty \leq 3\varepsilon. \qquad \square$$

LEMMA 3.5. *For any point a,* $\mathcal{A}^a(D)$ *can be computed in polynomial time.*

Proof. We say an ultrametric U is a-restricted (with respect to D) if it satisfies the following constraints:

$$2m_a \geq U[i,j] \geq 2\max\{l_i, l_j\} \text{ for all } i, j, \tag{1}$$

$$U[a,i] = 2m_a \text{ for all } i \neq a. \tag{2}$$

For any distance matrix M, define $\mathcal{U}^a(M)$ to be an a-restricted ultrametric which minimizes $\| M - \mathcal{U}^a(M) \|_\infty$. Note that for all i, j, $\mathcal{U}^a(M)[i,j] \leq 2m_a$. We can therefore apply Theorem 3.3, and so $\| M - \mathcal{U}^a(M) \|_\infty$ can be computed in $O(n^2)$ time.

Let $T = \mathcal{U}^a(D + C^a) - C^a$. We now show that $T = \mathcal{A}^a(D)$.

CLAIM 3.5A. *T is an a-restricted additive metric.*

Proof. Let $D^a = D + C^a$. Constraint (2) implies that T is a-restricted, since $T[a,i] = \mathcal{U}^a(D + C^a)[a,i] - C^a[a,i] = 2m_a - (2m_a - D[a,i]) = D[a,i]$. By Lemma 3.2, we only need to show that T satisfies the triangle inequality, i.e.,

$$\begin{aligned} & T[i,j] \leq T[i,k] + T[k,j], \text{ for all distinct } i,j,k \\ \Leftrightarrow\ & \mathcal{U}^a(D^a)[i,j] - C^a[i,j] \leq \mathcal{U}^a(D^a)[i,k] - C^a[i,k] + \mathcal{U}^a(D^a)[k,j] - C^a[k,j] \\ \Leftrightarrow\ & \mathcal{U}^a(D^a)[i,j] \leq \mathcal{U}^a(D^a)[i,k] + \mathcal{U}^a(D^a)[k,j] - 2l_k \\ \Leftrightarrow\ & \mathcal{U}^a(D^a)[i,j] \leq \max\{\mathcal{U}^a(D^a)[i,k], \mathcal{U}^a(D^a)[k,j]\} \\ & \qquad + \min\{\mathcal{U}^a(D^a)[i,k], \mathcal{U}^a(D^a)[k,j]\} - 2l_k. \end{aligned}$$

Now, since $\mathcal{U}^a(D^a)$ is an ultrametric,

$$\mathcal{U}^a(D^a)[i,j] \leq \max\{\mathcal{U}^a(D^a)[i,k], \mathcal{U}^a(D^a)[k,j]\}.$$

Also, $\min\{\mathcal{U}^a(D^a)[i,k], \mathcal{U}^a(D^a)[k,j]\} \geq 2l_k$ by constraint (1). Hence, the claim is proved. $\square$

CLAIM 3.5B. *$\mathcal{A}^a(D) + C^a$ is an a-restricted ultrametric.*

Proof. From Lemma 3.1, $\mathcal{A}^a(D) + C^a$ is an ultrametric. To show that constraint (2) is satisfied, define $T' = \mathcal{A}^a(D) + C^a$ and note that

$$T'[a,i] = \mathcal{A}^a(D)[a,i] + C^a[a,i] = D[a,i] + l_i + l_a = 2m_a.$$

For constraint (1), we use the fact that $\mathcal{A}^a(D)$ is a metric, and therefore, for all $i, j \neq a$,

$$\begin{aligned} & \mathcal{A}^a(D)[a,j] \leq \mathcal{A}^a(D)[i,j] + \mathcal{A}^a(D)[a,i] \\ \Rightarrow \quad & T'[a,j] - C^a[a,j] \leq T'[i,j] - C^a[i,j] + T'[a,i] - C^a[a,i] \\ \Rightarrow \quad & T'[a,j] \leq T'[j,i] + T'[a,i] - 2l_i \\ \Rightarrow \quad & 2m_a \leq T'[j,i] + 2m_a - 2l_i \\ \Rightarrow \quad & T'[j,i] \geq 2l_i. \end{aligned}$$

By symmetry, $T'[j,i] \geq 2l_j$. Also,

$$\begin{aligned} T'[i,j] &= \mathcal{A}^a(D)[i,j] + l_i + l_j \\ &\leq \mathcal{A}^a(D)[a,i] + \mathcal{A}^a(D)[a,j] + l_i + l_j \\ &= 2m_a. \end{aligned}$$

Therefore, constraint (1) is also satisfied and Claim 3.5B is proved. □

Finally,

$$\begin{aligned} \| T - D \|_\infty &\geq \| \mathcal{A}^a(D) - D \|_\infty \text{ (by Claim 3.5A)} \\ &= \| (\mathcal{A}^a(D) + C^a) - (D + C^a) \|_\infty \\ &\geq \| \mathcal{U}^a(D + C^a) - (D + C^a) \|_\infty \text{ (by Claim 3.5B)} \\ &= \| T - D \|_\infty \text{ (by construction).} \end{aligned}$$

Therefore, $\| T - D \|_\infty = \| \mathcal{A}^a(D) - D \|_\infty$. This proves the lemma. □

Lemmas 3.4 and 3.5 imply the following theorem.

THEOREM 3.6. *Given an $n \times n$ distance matrix D, we can find a tree metric T in $O(n^2)$ time such that*

$$\| T - D \|_\infty \leq 3 \| \mathcal{A}(D) - D \|_\infty.$$

4. Tightness of analysis. In this section we show that the constant in Lemma 3.4 is tight, and that for some distance matrices it is not improved by trying different values of c.

THEOREM 4.1. *There is an $n \times n$ distance matrix D such that, for all points c,*

$$\frac{\| D - \mathcal{A}^c(D) \|_\infty}{\| D - \mathcal{A}(D) \|_\infty} = 3.$$

Proof. Consider the following distance matrix D for the points $q_0, \ldots, q_8$:

$$\begin{aligned} D[q_i, q_j] &= d - \varepsilon \quad \text{if } i = (j+1) \bmod 9 \text{ or } i = (j-1) \bmod 9 \\ &= 0 \qquad \text{if } i = j \bmod 3 \\ &= d + \varepsilon \quad \text{otherwise.} \end{aligned}$$

Note that for each $c = q_i$, there exists $a_1 = q_{(i+1) mod 9}$, $a_2 = q_{(i-1) mod 9}$, $b_1 = q_{(i+4) mod 9}$, and $b_2 = q_{(i-4) mod 9}$ such that

$$\begin{array}{ccccccc} D[c,a_1] & = & D[a_2,c] & = & D[b_2,b_1] & = & d-\varepsilon, \\ D[b_1,c] & = & D[c,b_2] & = & D[a_1,a_2] & = & d+\varepsilon, \text{ and} \\ D[a_1,b_1] & = & D[a_2,b_2] & = & 0. & & \end{array}$$

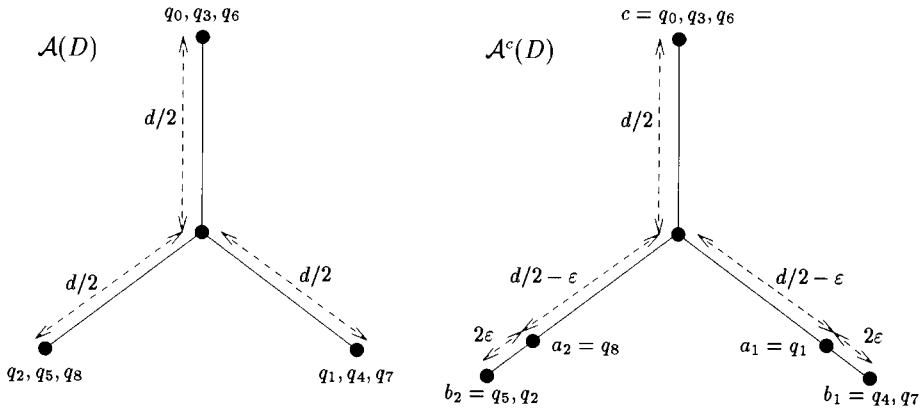


FIG. 4.1. *Trees approximating D.*

If we take d to be much larger than ε, it is easy to see that any reasonable approximation by a tree metric T uses a tree with a central vertex m from which three edges lead to subtrees containing c, $\{a_1, b_1\}$, and $\{a_2, b_2\}$ respectively.

Hence,

$$T[b_1, c] - T[c, a_1] + T[a_1, a_2] - T[a_2, c] + T[c, b_2] - T[b_2, b_1] = 0,$$

whereas

$$D[b_1, c] - D[c, a_1] + D[a_1, a_2] - D[a_2, c] + D[c, b_2] - D[b_2, b_1] = 6\varepsilon.$$

Therefore, any such approximation T satisfies $\| D - T \|_\infty \geq \varepsilon$.

For a c-restricted approximation T (where $T[u, c] = D[u, c]$ for all c), we find that

$$T[a_1, a_2] - D[a_1, a_2] - T[b_2, b_1] + D[b_2, b_1] = 6\varepsilon,$$

and so $\| D - T \|_\infty \geq 3\varepsilon$.

Figure 4.1 shows optimal solutions which establish that $\| D - \mathcal{A}(D) \|_\infty = \varepsilon$ and that $\| D - \mathcal{A}^c(D) \|_\infty = 3\varepsilon$. □

Some rather involved examples show that there are c-optimal trees for which changing the edge lengths cannot bring the error down below $3\varepsilon - o(1)$. Thus there is no significant worst-case advantage to the obvious heuristic of changing the edge lengths optimally using linear programming.

5. Lower bound. In this section, we show that the problem of finding a tree T such that $\| T - D \|_\infty < \frac{9}{8}\varepsilon$ is $\mathcal{NP}$-hard. First, we show that a decision version of the numerical taxonomy problem is $\mathcal{NP}$-complete.

The numerical taxonomy problem.
Input: A distance matrix $D : S^2 \to \Re_{\geq 0}$, and a threshold $\Delta \in \Re_{\geq 0}$.
Question: Is there a *tree metric* T which spans S and for which $\| T - D \|_\infty \leq \Delta$?

THEOREM 5.1. *The numerical taxonomy problem is $\mathcal{NP}$-complete.*

Proof. That the problem is in $\mathcal{NP}$ is immediate. We show $\mathcal{NP}$-completeness by reduction from 3SAT. For an instance of 3SAT with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_k$, we will construct a distance matrix D such that the 3SAT expression is

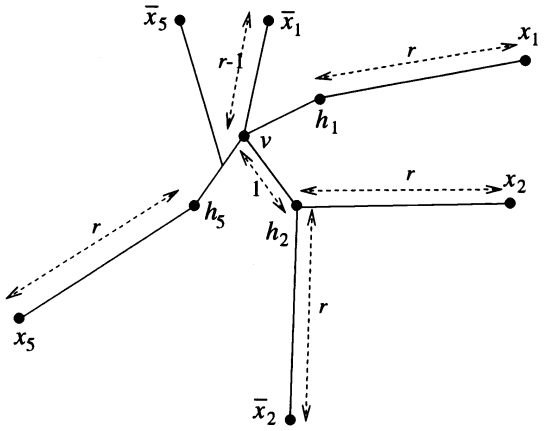


FIG. 5.1. *Portion of sample layout.*

satisfiable if and only if $\| D - \mathcal{A}(D) \|_\infty \leq \Delta = 2$. Let integer r represent some sufficiently large distance (such as 10). We construct a distance matrix D to approximate path lengths on a tree with leaves $x_i, \bar{x}_i, h_i$ for $1 \leq i \leq n$, and c_j, c'_j, c''_j for $1 \leq j \leq k$, and v.

To simplify the description of the construction we first present it in the form of a set of inequalities on the distances between the vertices of a tree T, which are expressed later in the required form. For example, we shall write "$T[x_i, \bar{x}_i] \geq 2r$" at first, and realize this constraint eventually by letting $D[x_i, \bar{x}_i] = 2r + \Delta$. We classify the inequalities as follows.

A. *Literal pairs.*

$$T[x_i, \bar{x}_i] \geq 2r, \quad T[x_i, h_i] \leq r, \quad T[\bar{x}_i, h_i] \leq r \quad \text{for all } i.$$

These inequalities force h_i to be the midpoint of the path between x_i and $\bar{x}_i$ for all i.

B. *Star-like tree.*

(1) $T[v, x_i] \leq r + 1, \quad T[v, \bar{x}_i] \leq r + 1 \quad \text{for all } i,$

(2) $T[h_i, h_j] \geq 2, \quad T[h_i, x_j] \geq r, \quad T[h_i, \bar{x}_j] \geq r \quad \text{for all } i, j \ (i \neq j).$

The inequalities B(1), together with those in A, imply $T[v, h_i] \leq 1$ for all i, and we can then use the first inequality of B(2) to deduce that $T[v, h_i] = 1$ for all i.

The vertex v must be at the center of a star with each h_i at distance 1 from it along separate edges. From each h_i, at least one of the two paths of length r to x_i and $\bar{x}_i$ proceeds away from v. An impression of a general feasible configuration is presented in Figure 5.1.

The essential feature of such configurations, which we shall use in our reduction, is that for each i, *at least one* of x_i and $\bar{x}_i$ is at distance $r + 1$ from v. The final inequalities will represent the satisfaction of clauses by literals. A satisfying literal will correspond to a vertex $\tilde{x}_i \in \{x_i, \bar{x}_i\}$ such that $T[v, \tilde{x}_i] = r - 1$. Clearly, x_i and $\bar{x}_i$ cannot both be satisfying literals.

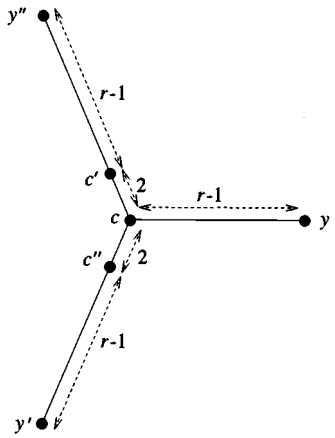


FIG. 5.2. *Layout of clause vertices.*

Now, we present the third set of inequalities that deal with the "clause" vertices c_j, c'_j, c''_j. Specifically, we will show that a clause is satisfied if and only if at least one of its literals is at a distance less than $r+1$ from v.

C. *Clause satisfaction.* For each clause $C_j = (y_j, y'_j, y''_j)$ where y_j, y'_j, y''_j are literals, we have three vertices c_j, c'_j, c''_j and the following inequalities (where we drop the subscript j for clarity):

$$T[c, y'] \leq r+1,\ T[c, y''] \leq r+1,$$

$$T[c', y''] \leq r+1,\ T[c', y] \leq r+1,$$

$$T[c'', y] \leq r+1,\ T[c'', y'] \leq r+1,$$

$$T[c, c'] \geq 2,\ T[c', c''] \geq 2,\ T[c'', c] \geq 2.$$

If $T[v, y_j]$, $T[v, y'_j]$, and $T[v, y''_j]$ are all $r+1$, then the first inequalities in C force each of c_j, c'_j, c''_j to coincide with v, contravening the last three inequalities. However, if at least one of these literals is at a distance $r-1$ of v then a configuration of the form illustrated in Figure 5.2 is feasible.

We claim that the complete set of inequalities is satisfiable if and only if the corresponding 3SAT formula is satisfiable. In one direction, suppose that there is a satisfying truth assignment to the logical variables. For each variable, lay out the corresponding tree vertices so that the vertex corresponding to the true literal is at distance $r-1$ from v (the "false" literal will be at distance $r+1$ from v). Each clause has a satisfying literal; therefore, for each j, at least one of y_j, y'_j, y''_j is at distance $r-1$ from v in the tree, thus allowing a legal placement of c_j, c'_j, c''_j. On the other hand, if there is a tree layout satisfying all the inequalities then at least one of y_j, y'_j, y''_j must be within distance $r-1$ of v for each j. Since at most one of x_i and $\bar{x}_i$ can be within $r-1$ of v, the layout yields a (partial) assignment which satisfies the logical formula.

To complete the proof, we construct a distance matrix D such that (1) if for some tree metric T, $\| T - D \|_\infty \leq \Delta$, then T satisfies all the inequalities from A, B, and

C, and (2) for the tree layout T described above, corresponding to a satisfiable 3SAT expression, we have $\| T - D \|_\infty \leq \Delta$.

Concerning (1), for all vertices a, b, and all $z \in \Re_{\geq 0}$, if an inequality is of the form $T[a,b] \geq z$, let $D[a,b] = z + \Delta$. Correspondingly, if the inequality is of the form $T[a,b] \leq z$, let $D[a,b] = z - \Delta$. Concerning (2), for $\tilde{x}_i \in \{x_i, \bar{x}_i\}$, $\tilde{x}_j \in \{x_j, \bar{x}_j\}$, $i \neq j$, in our intended configurations we have $2r - 2 \leq T[\tilde{x}_i, \tilde{x}_j] \leq 2r + 2$, with either extreme possible. Therefore, we take $D[\tilde{x}_i, \tilde{x}_j] = 2r$. Since $\Delta = 2$, this covers both extremes. Similarly, for each clause C we take $D[c, y] = D[c', y'] = D[c'', y''] = r + 1$. Suitable values for the remaining entries of D are easy to find. This completes the proof of Theorem 5.1. □

Next, we strengthen Theorem 5.1 to show a hardness-of-approximation result.

THEOREM 5.2. *Given a 3SAT instance S, a distance matrix D can be computed in polynomial time such that*

1. *if S is satisfiable, then $\| D - \mathcal{A}(D) \|_\infty \leq 2$;*
2. *if S is not satisfiable, then $\| D - \mathcal{A}(D) \|_\infty \geq 2 + \frac{1}{4}$.*

Proof. We extend the construction of Theorem 5.1 by relaxing some of the inequalities by a fixed amount δ and omitting others. The matrix D is the same as before.

A. *Literal pairs.*

$$T[x_i, \bar{x}_i] \geq 2r - \delta, \quad T[x_i, h_i] \leq r + \delta, \quad T[\bar{x}_i, h_i] \leq r + \delta \quad \text{for all } i.$$

B. *Star-like tree.*

$$T[v, x_i] \leq r + 1 + \delta, \quad T[v, \bar{x}_i] \leq r + 1 + \delta, \quad T[v, h_i] \leq 1 + \delta \quad \text{for all } i,$$

$$T[h_i, h_j] \geq 2 - \delta \quad \text{for all } i, j \ (i \neq j).$$

C. *Clause satisfaction.* For each clause $C = (y, y', y'')$ where y, y', y'' are literals, we have three vertices c, c', c'' and the following inequalities:

$$T[c, y'] \leq r + 1 + \delta,\ T[c, y''] \leq r + 1 + \delta,$$

$$T[c', y''] \leq r + 1 + \delta,\ T[c', y] \leq r + 1 + \delta,$$

$$T[c'', y] \leq r + 1 + \delta,\ T[c'', y'] \leq r + 1 + \delta,$$

$$T[c, c'] \geq 2 - \delta,\ T[c', c''] \geq 2 - \delta,\ T[c, c''] \geq 2 - \delta.$$

Note that the inequalities are a relaxation of the inequalities in the construction of Theorem 5.1. It follows that if S is satisfiable, then there is a tree T that satisfies these inequalities for all nonnegative δ. Consequently, if S is satisfiable, then $\| D - \mathcal{A}(D) \|_\infty \leq 2$.

In the remaining part, we consider an arbitrary tree T which satisfies inequalities A, B, and C. Our aim will be to show that if S is not satisfiable then $\delta \geq 1/4$, and so $\| D - T \|_\infty \geq 2 + 1/4$.

For any three distinct tree vertices u, v, w, let $\texttt{meet}(u, v, w)$ denote the intersection point of the paths between them. We interpret x_i as false if and only if

$T[h_i, \texttt{meet}(v, h_i, x_i)] \leq T[h_i, \texttt{meet}(v, h_i, \bar{x}_i)]$. Without loss of generality, we may *restrict our attention to a tree for which our interpretation sets all x_i to be false.*

For any variable x_i, let $\hat{h}_i$ denote $\texttt{meet}(h_i, x_i, \bar{x}_i)$ and $\hat{v}_i$ denote $\texttt{meet}(v, h_i, x_i)$. Note that x_i being false implies that $T[h_i, \hat{v}_i] \leq T[h_i, \hat{h}_i]$.

CLAIM 5.2A. *For all i,* (1) $T[h_i, \hat{v}_i] \leq T[h_i, \hat{h}_i] \leq 3\delta/2$, *and* (2) $T[x_i, \hat{h}_i] - T[h_i, \hat{h}_i] \geq r - 2\delta$.

Proof. For (1), $2T[h_i, \hat{h}_i] = T[x_i, h_i] + T[\bar{x}_i, h_i] - T[x_i, \bar{x}_i] \leq 2(r+\delta) - (2r-\delta) = 3\delta$. For (2), $T[x_i, \hat{h}_i] - T[h_i, \hat{h}_i] = T[x_i, \bar{x}_i] - T[\bar{x}_i, h_i] \geq 2r - \delta - (r + \delta) = r - 2\delta$. □

For each $j \neq i$, set $h_i^j = \texttt{meet}(h_j, h_i, x_i)$.

CLAIM 5.2B. *For all $\delta < \frac{2}{7}$ and for all $j \neq i$, $T[h_i, h_i^j] < T[h_i, \hat{v}_i]$.*

Proof. Suppose $T[h_i, h_i^j] \geq T[h_i, \hat{v}_i]$. Then there are simple paths from h_i to $\hat{v}_i$ to h_j and from v to $\hat{v}_i$ to h_j. Therefore,

$$\begin{aligned} 0 &= T[h_i, \hat{v}_i] + T[\hat{v}_i, h_j] - T[h_i, h_j] \\ &\leq T[h_i, \hat{v}_i] + T[v, h_j] - T[h_i, h_j] \\ &\leq 3\delta/2 + (1 + \delta) - (2 - \delta). \end{aligned}$$

Hence $\delta \geq \frac{2}{7}$. □

CLAIM 5.2C. *For all $\delta < \frac{2}{7}$ and for all $i \neq j$, $T[x_i, x_j] \geq 2r + 2 - 5\delta$.*

Proof. By Claims 5.2B and 5.2A(1), $T[h_i, h_i^j] \leq 3\delta/2$ and $T[h_j, h_j^i] \leq 3\delta/2$. Since $T[h_i, h_j] \geq 2 - \delta$ and $\delta \leq 1/2$, we may conclude that we have a simple path from h_i to h_i^j to h_j^i to h_j, and a simple path from x_i to $\hat{h}_i$ to h_i^j to h_j^i to $\hat{h}_j$ to x_j. Note, however, that $\hat{h}_i$ and h_i^j may coincide, and similarly for h_j^i and $\hat{h}_j$. In conclusion,

$$\begin{aligned} T[x_i, x_j] &= T[x_i, \hat{h}_i] + T[\hat{h}_i, h_i^j] + T[h_i^j, h_j^i] + T[h_j^i, \hat{h}_j] + T[\hat{h}_j, x_j] \\ &\geq T[x_i, \hat{h}_i] + T[h_i^j, h_j^i] + T[\hat{h}_j, x] \\ &= T[x_i, \hat{h}_i] + T[h_i, h_j] - T[h_i, \hat{h}_i] - T[h_j, \hat{h}_j] + T[\hat{h}_j, x_j] \\ &\geq 2(r - 2\delta) + 2 - \delta = 2r + 2 - 5\delta. \end{aligned}$$

For the last inequality, we used Claim 5.2A(2). □

Finally, we show that if S is not satisfiable then $\delta \geq 1/4$. If $\delta \geq 2/7$ then this is trivially true, so we may assume that the conclusions of Claims 5.2B and 5.2C apply.

Let vertices x, x', x'' in T correspond to the three false literals of a clause. Let $p = \texttt{meet}(x, x', x'')$. Without loss of generality, assume $T[x, p] \geq T[x', p] \geq T[x'', p]$. Let d be at the middle of the path from x to x''. By Claim 5.2C, $T[x, d] = T[x'', d] \geq r + 1 - 5\delta/2$. Hence the bounds of $r + 1 + \delta$ on $T[x, c']$ and $T[x'', c']$ from the inequalities in C imply that $T[d, c'] \leq 7\delta/2$.

Now d is situated on the path from x to p, and $T[p, x'] \geq T[p, x'']$, implying $T[d, x'] \geq T[d, x''] \geq r + 1 - 5\delta/2$. Hence, as above, the bounds of $r + 1 + \delta$ on $T[x, c'']$ and $T[x'', c'']$ imply that $T[d, c''] \leq 7\delta/2$. Consequently $T[c', c''] \leq T[c', d] + T[d, c''] \leq 7\delta$. However, from C we also have the inequality $T[c', c''] \geq 2 - \delta$. Thus $7\delta \geq 2 - \delta$ and so $\delta \geq 1/4$.

Since T was arbitrary, we have shown that if S is not satisfiable then there is no tree T such that $\| D - T \|_\infty < 2 + 1/4$, i.e., $\| D - \mathcal{A}(D) \|_\infty \geq 2 + 1/4$. □

Theorem 5.2 immediately implies a hardness-of-approximation result for the numerical taxonomy problem.

COROLLARY 5.3. *It is an $\mathcal{NP}$-hard problem, given a distance matrix D, to find*

an additive metric T such that

$$\frac{\| D - T \|_\infty}{\| D - \mathcal{A}(D) \|_\infty} < \frac{9}{8}.$$

Proof. For any such T, if $\| D - T \|_\infty \geq 2 + 1/4$ then $\| D - \mathcal{A}(D) \|_\infty > 2$ and S is unsatisfiable, and if otherwise then $\| D - \mathcal{A}(D) \|_\infty \leq \| D - T \|_\infty < 2 + 1/4$ and S is satisfiable. □

6. Generalization to other norms. First, we show that Lemma 3.4 can be generalized to other norms.

THEOREM 6.1. *Let D be a distance matrix and T be a tree such that $\| D - T \|_p \leq \varepsilon$. Then there exists a point a and an a-restricted tree $T^{/a}$ such that $\| D - T^{/a} \|_p \leq 3\varepsilon$.*

Proof. For any point a, the construction of Lemma 3.4 returns an a-restricted tree $T^{/a}$ such that

$$|T^{/a}[i,j] - D[i,j]| \leq |\varepsilon[i,j]| + |\varepsilon[a,i]| + |\varepsilon[a,j]| \text{ for all } i,j. \tag{3}$$

Also, by the convexity of the function $|x|^p$ for real x, we have

$$\sum_{i=1}^{k} \frac{|x_i|^p}{k} \geq \left| \frac{\sum_{i=1}^{k} x_i}{k} \right|^p. \tag{4}$$

We continue the proof by an averaging argument. Clearly,

$$\min_a \{(\| T^{/a} - D \|_p)^p\} \leq \frac{\sum_{a=1}^{n} (\| T^{/a} - D \|_p)^p}{n}.$$

We use inequalities (3) and (4) to bound the sum

$$\begin{aligned}
\sum_{a=1}^{n} (\| T^{/a} - D \|_p)^p &= \sum_{a=1}^{n} \sum_{i=1, i\neq a}^{n} \sum_{j=1, j\neq a}^{n} |\varepsilon[i,j] - \varepsilon[a,i] - \varepsilon[a,j]|^p \\
&\leq 3^{p-1} \sum_{a=1}^{n} \sum_{i=1, i\neq a}^{n} \sum_{j=1, j\neq a}^{n} |\varepsilon[i,j]|^p + |\varepsilon[a,i]|^p + |\varepsilon[a,j]|^p \\
&= 3^p n (\| T - D \|_p)^p.
\end{aligned}$$

The theorem follows. □

As in the case of L_∞, we can show that if T is an a-optimal tree for D under L_k, then $T + C^a$ is an optimal a-restricted ultrametric for $D + C^a$ under the same norm. We define the *Additive$_k$* problem as, given a matrix D, output an additive metric A minimizing $\| D - A \|_k$. Similarly, the *Ultrametric$_k$* problem is, given a matrix D, output an ultrametric U minimizing $\| D - U \|_k$.

We conclude with Theorem 6.2.

THEOREM 6.2. *If $A(D)$ is an algorithm which achieves an α-approximation for the a-restricted Ultrametric$_k$ problem and runs in time $T(n)$ on an $n \times n$ matrix, then there is an algorithm $F(D)$ which achieves a 3α-approximation for the Additive$_k$ problem and runs in $O(nT(n))$ time.*

REFERENCES

[1] R. Baire, *Leçons sur les Fonctions Discontinues*, Gauthier Villars, Paris, 1905.

[2] J-P. Barthélemy and A. Guénoche, *Trees and Proximity Representations*:, Wiley, New York, 1991.

[3] P. Buneman, *The recovery of trees from measures of dissimilarity*, in Mathematics in the Archaeological and Historical Sciences, F. Hodson, D. Kendall, and P. Tautu, eds., Edinburgh University Press, Edinburgh, 1971, pp. 387–395.

[4] L. Cavalli-Sforza and A. Edwards, *Phylogenetic analysis models and estimation procedures*, Amer. J. Human Genetics, 19 (1967), pp. 233–257.

[5] W. H. E. Day, *Computational complexity of inferring phylogenies from dissimilarity matrices*, Bull. Math. Biol., 49 (1987), pp. 461–467.

[6] M. Farach, S. Kannan, and T. Warnow, *A robust model for finding optimal evolutionary trees*, Algorithmica, 13 (1995), pp. 155–179.

[7] P. H. A. Sneath and R. R. Sokal, *Numerical Taxonomy*, W. H. Freeman, San Francisco, CA, 1973.

[8] D. L. Swofford and G. J. Olsen, *Phylogeny reconstruction*, in Molecular Systematics, D. M. Hillis and C. Moritz, eds., Sinauer Associates Inc., Sunderland, MA, 1990, pp. 411–501.

[9] H. T. Wareham, *On the Complexity of Inferring Evolutionary Trees*, Technical Report 9301, Memorial University of Newfoundland, 1993.

[10] M. S. Waterman, T. F. Smith, M. Singh, and W. A. Beyer, *Additive evolutionary trees*, J. Theor. Biol., 64 (1977), pp. 199–213.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 1086–1111

COMPETITIVE ON-LINE ALGORITHMS FOR DISTRIBUTED DATA MANAGEMENT*

CARSTEN LUND†, NICK REINGOLD†, JEFFERY WESTBROOK†‡, AND DICKY YAN§

Abstract. Competitive on-line algorithms for data management in a network of processors are studied in this paper. A data object such as a file or a page of virtual memory is to be read and updated by various processors in the network. The goal is to minimize the communication costs incurred in serving a sequence of such requests. Distributed data management on important classes of networks—trees and bus-based networks—are studied. Optimal algorithms with constant competitive ratios and matching lower bounds are obtained. Our algorithms use different interesting techniques, such as work functions [Chrobak and Larmore, *Proc. DIMACS Workshop on On-Line Algorithms,* AMS, 1991, pp. 11–64] and "factoring."

Key words. on-line algorithms, competitive analysis, memory management, data management

AMS subject classifications. 68Q20, 68Q25

PII. S0097539795287824

1. Introduction.

The management of data in a distributed network is an important and much studied problem in management science, engineering, computer systems, and theory [3, 11]. Dowdy and Foster [11] give a comprehensive survey of research in this area, listing 18 different models and many papers. A data object, F, such as a file or a page of virtual memory, is to be read and updated by a network of processors. Each processor may store a copy of F in its local memory, so as to reduce the time required to read the data object. All copies must be kept consistent, however, so having multiple copies increases the time required to write to the object. As read and write requests occur at the processors, an on-line algorithm has to decide whether to replicate, move, or discard copies of F after serving each request while trying to minimize the total cost incurred in processing the requests. The on-line algorithm has no knowledge of future requests, and no assumptions are made about the pattern of requests. We apply competitive analysis [6] to such an algorithm.

Let σ denote a sequence of read and write requests. A deterministic on-line algorithm A is said to be *c-competitive* if for all σ, $C_A(\sigma) \leq c \cdot OPT(\sigma) + B$ holds, where $C_A(\sigma)$ and $OPT(\sigma)$ are the costs incurred by A and the optimal off-line solution, respectively, and c and B are functions which are independent of σ but which may depend upon the input network and file size. If A is a randomized algorithm, we replace $C_A(\sigma)$ by its *expected* cost and consider two types of adversaries: The *oblivious* adversary chooses σ in advance, and the more powerful *adaptive* on-line adversary builds σ on-line, choosing each request with knowledge of the random moves made

*Received by the editors June 16, 1995; accepted for publication (in revised form) May 19, 1997; published electronically February 19, 1999. A preliminary version of this paper appeared as "On-Line distributed data management," in *Proc. 2nd Annual European Symposium on Algorithms*, ESA '94, Lecture Notes in Computer Science, Utrecht, The Netherlands, 1994, Springer-Verlag, New York, pp. 202–214.

http://www.siam.org/journals/sicomp/28-3/28782.html

†AT&T Labs—Research, 180 Park Avenue, Florham Park, NJ 07932 (lund@research.att.com, reingold@research.att.edu, westbrook@research.att.com).

‡The work of this author was performed while at Yale University. This research was partially supported by NSF grant CCR-9009753.

§Department of Operations Research, AT&T Labs, Room 3J-314, 101 Crawfords Corner Road, Holmdel, NJ 07733-3030 (yan@att.com). The work of this author was performed while at Yale University. The research of this author was partially supported by fellowships from Yale University.

by A on the previous requests. The oblivious adversary is charged the optimal off-line cost, while the adaptive on-line adversary has to serve σ and be charged on-line. (See Ben-David et al. [6] for a full discussion of different types of adversaries.) An algorithm is *strongly* competitive if it achieves the best possible competitive ratio.

In this paper, we focus on two important classes of networks: trees and the uniform network. A tree is a connected acyclic graph on n nodes and $(n-1)$ edges; the uniform network is a complete graph on n nodes with unit edge weights. We obtain strongly competitive deterministic and randomized on-line algorithms for these classes.

Our algorithms use different interesting techniques, such as offset functions and "factoring." Competitive on-line algorithms based on offset functions have been found for the 3-server [9] and the migration problems [10]. An advantage of these algorithms is that they do not need to record the entire history of requests and the actions of the on-line algorithm since decisions are based on the current offset values which can be updated easily. Factoring is first observed in [7] and used in [10, 17]. The idea is to break down an on-line problem on a tree into single edge problems. Thus strongly competitive strategies for a single edge are generalized to a tree. Our algorithms are strongly competitive for specific applications and networks and also illustrate these two useful techniques. Our randomized algorithm for file allocation is *barely random* [20]; i.e., it uses a bounded number of random bits, independent of the number of requests. A random choice is made only at the initialization of the algorithm, after which it runs deterministically.

1.1. Problem description. We study three variants of distributed data management: *replication* [1, 7, 17], *migration* [7, 10, 22], and *file allocation* (FAP) [2, 5]. They can be described under the same framework. We are given an undirected graph $G = (V, E)$ with nonnegative edge weights and $|V| = n$, where each node represents a processor. Let F represent a data file or a page of memory to be stored in the processors. At any time, let $R \subseteq V$, the residence set, represent the set of nodes that contain a copy of F. We always require that $R \neq \emptyset$. Initially, only a single node v contains a copy of F and $R = \{v\}$.

A sequence of read and write requests occur at the processors. A *read at processor* p requests an examination of the contents of some data location in F; a *write at processor* p requests a change to the contents of some location in F. The location identifies a single word or record in F. A read can be satisfied by sending a message to any processor holding a copy of F; that processor then returns the information stored in the requested location. A write is satisfied by sending an update message to each processor holding a copy F, telling it how to modify the desired location. After a request is served, the on-line server can decide how to reallocate the multiple copies of F.

Let $D \in \mathcal{Z}^+$ be an integer constant, $D \geq 1$, which represents the number of records in F.[1] The costs for serving the requests and redistributing the files are as follows:

Service Cost. Suppose a request occurs at a node v. If it is a read request, it is served at a cost equal to the shortest path distance from v to a nearest node in R; if it is a write request, it is served at a cost equal to the size of the minimum Steiner tree[2] that contains all the nodes in $R \cup \{v\}$.

[1] $\mathcal{R}$, $\mathcal{Z}^+$, and $\mathcal{Z}_0^+$ represent the sets of reals, positive integers, and nonnegative integers, respectively.

[2] See section 2 for a definition.

Movement Cost. The algorithm can replicate a copy of F to a node v at a cost D times the shortest path distance between v and the nearest node with a copy of F; it can discard a copy of F at no cost.

A file reallocation consists of a sequence of zero or more replications and discards of copies of F. The replications and discards can be done in any order as long as the residence set has a size of at least 1. The movement cost incurred during a reallocation is equal to the total sum of all replication costs.

The replication and migration problems are special cases of file allocation. For migration, we require $|R| = 1$. For replication, all the requests are reads, and it can be assumed that all replicated copies of F are not discarded. The (off-line) optimization problem is to specify R after each new request is served so that the total cost incurred is minimized. For on-line replication, we consider only competitive algorithms that have $B = 0$ in the inequality above; otherwise a trivial 0-competitive algorithm exists [7].

Following previous papers on allocation and related problems, we adopt a "lookahead-0" model. In this model, once a request is revealed, the on-line algorithm must immediately pay the service cost before making any changes to the residence set. One may contrast lookahead-0 with a lookahead-1 model, in which the algorithm may change the residence set before paying the service cost. We discuss the lookahead issue further below, together with some implementation issues.

1.2. Previous and related results. Black and Sleator [7] were the first to use competitive analysis to study any of these problems, giving strongly 3-competitive deterministic algorithms for file migration on trees and uniform networks and strongly 2-competitive deterministic algorithms for replication on trees and uniform networks.

Replication. Imase and Waxman [14] showed that a greedy algorithm for building Steiner trees on-line is $\Theta(\log n)$-competitive, where n is the number of nodes, and that this ratio is optimal within constant factors for general networks. This algorithm is the basis of a solution for on-line replication in general networks. Koga [17] gave randomized algorithms that are 2-competitive and 4-competitive against an adaptive on-line adversary on trees and circles, respectively. He also obtained a randomized algorithm with a competitive ratio that depends only on D and approaches $(1+1/\sqrt{2})$ as D grows large, against an oblivious adversary on trees.

Migration. Westbrook [22] obtained a randomized algorithm for uniform networks with a competitive ratio that depends only on D and approaches $((5+\sqrt{17})/4)$ as D grows large, against an oblivious adversary. For general networks, Westbrook [22] obtained a strongly 3-competitive randomized algorithm against an adaptive on-line adversary. He also obtained an algorithm against an oblivious adversary with a competitive ratio that depends only on D and approaches $(1+\phi)$-competitive as D grows large, where $\phi \approx 1.62$ is the golden ratio. Chrobak et al. [10] studied migration on various classes of metric spaces, including trees, hypercubes, meshes, real vector spaces, and general products of trees. They gave strongly $(2+1/2D)$-competitive randomized algorithms for these spaces, $(2+1/2D)$-competitive deterministic algorithms for some of these spaces, and a general lower bound for deterministic algorithms of $(85/27)$. Recently, Bartal, Charikar, and Indyk [4] obtained a 4.086-competitive deterministic algorithm.

File Allocation. For general networks, Awerbuch, Bartel, and Fiat [2] and Bartal, Fiat, and Rabani [5] give $O(\log n)$-competitive deterministic and randomized algorithms against an adaptive on-line adversary, respectively. Westbrook and Yan [23] show that Bartal, Fiat, and Rabani's algorithm is $O(\log d(G))$-competitive on an un-

TABLE 1.1
The state of the art: Trees and uniform networks. Note $e_D = (1 + 1/D)^D$.

		Replication	Migration	File allocation
Deterministic	uniform	2 [7]	3 [7]	3 [5]
	tree	2 [7]	3 [7]	3*
Randomized	uniform	$e_D/(e_D-1)$*	$2+1/(2D)$*	?
	tree	$e_D/(e_D-1)$*	$2+1/(2D)$ [10]	$2+1/D$*

*This paper

weighted graph with diameter *d(G)*, and there exists a $O(\log^2 d(G))$-competitive deterministic algorithm. Bartal, Fiat, and Rabani also find a $(3+O(1/D))$-competitive deterministic algorithm on a tree and strongly 3-competitive randomized algorithms against an adaptive on-line adversary on a tree and uniform network. Since replication is a special case of file allocation, these upper bounds are also valid for replication when the additive constant B is zero.

1.3. New results. This paper contributes the following results:

- For on-line file allocation on a tree, we give a strongly 3-competitive deterministic algorithm and a $(2+1/D)$-competitive randomized algorithm against an oblivious adversary and show that this is optimal even if G is an edge.
- For uniform networks, we show that the off-line file allocation problem can be solved in polynomial time. We give a strongly $(2 + 1/(2D))$-competitive randomized on-line algorithm for migration against an oblivious adversary on the uniform network.
- For the replication problem, we show that the off-line problem is NP-hard; this implies that the file allocation problem is also NP-hard. We obtain randomized algorithms that are $(e_D/(e_D - 1))$-competitive against an oblivious adversary on a tree and a uniform network; this is optimal even if G is an edge. (Albers and Koga [1] have independently obtained the same results for on-line replication using a different method.)
- We show that no randomized algorithm for replication on a single edge can be better than 2-competitive against an adaptive on-line adversary. Thus Koga's [17] algorithm for replication on a tree is strongly competitive.

Table 1.1 summarizes the competitive ratios of the best known deterministic and randomized algorithms against an oblivious adversary for replication, migration, and file allocation on trees and uniform networks. They are all optimal.

1.4. Lookahead and implementation issues. As stated above, we adopt the lookahead-0 model that has been used in all previous work on allocation and its variants. Studies of some other on-line problems, however, have used a lookahead-1 model, and in this subsection we comment briefly on the distinction.

In a lookahead-1 model of allocation, some request sequences could be served by an on-line algorithm at a lower cost than would be possible in the lookahead-0 model. For example, if a write request occurs, a lookahead-1 algorithm can drop all but one copies of F before servicing the request, thereby reducing the service cost. The lookahead-0 model is more appropriate for file allocation, however, because the service cost models both the message cost of satisfying a request, which includes the cost of transmitting an answer back to a read request or passing an update on to all copies, and the message cost of the control messages that must be transmitted in order for the algorithm to learn of new requests and to implement its replication and

drop decisions. Specifically, we assume that a new replication will not occur unless at least one member of the replication set has been told of a new request, and a processor will not discard a copy unless it has been told of a new write request.

We claim that for large values of D the optimal competitive ratio in a lookahead-1 model is not materially different from the optimal competitive ratio in a lookahead-0 model. In particular, if there is a c-competitive algorithm using lookahead-1, there is a $(c + 2/D)$-competitive algorithm using lookahead-0. The lookahead-0 algorithm simulates the lookahead-1 algorithm by keeping the same residence set. When the lookahead-1 algorithm saves service cost on a read, the amount saved can be no more than the distance it replicates files just prior to satisfying the request. Similarly, when the lookahead-1 algorithm saves service cost on a write, the amount saved can be no more than the weight of a minimum Steiner tree which connects the dropped copies to an undropped copy. But at some point in the past, at least one of the dropped copies must have been replicated over each edge in that Steiner tree. Hence for each unit of distance saved on reads by the lookahead-1 algorithm, one file was moved one unit of distance. The same holds for writes. The total cost saved by the lookahead-1 algorithm is $\frac{2}{D}$ times the total movement cost. Both algorithms incur the same movement cost, however.

One may ask whether our service cost is too optimistic: Could our algorithms actually be implemented using only the control messages accounted for in the service cost? Although we do not directly address this issue, our algorithms are essentially distributed in nature and can be implemented with only constant message overhead in the special case of uniform and tree networks.

2. Preliminaries. We use the technique of work functions and offset functions introduced by Chrobak and Larmore [9]. Let S be a set of states, one for each legal residence set. Thus S is isomorphic to $2^V \setminus \{\emptyset\}$. Let *R(s)* denote the residence set corresponding to state $s \in S$. We say the *file system is in state s* if the current residence set is $R(s), s \in S$. Let $Y = \{v^r, v^w | v \in V\}$ be the set of possible requests, where v^r and v^w represent read and write requests at node v, respectively. A request sequence $\sigma = (\sigma_1, \ldots, \sigma_p)$ is revealed to the on-line algorithm, with each $\sigma_i \in Y$. Suppose the network is in state s when σ_i arrives. The algorithm will be charged a service cost of $ser(s, \sigma_i)$, where $ser(s, \sigma_i) : S \times Y \longrightarrow \mathcal{R}$ is as described in section 1.1. After serving σ_i, the algorithm can move to a different state t at a cost *tran(s,t)*, where $tran : S \times S \longrightarrow \mathcal{R}$ is the minimum cost of moving between the two residence sets.

The work function $W_i(s)$ is the minimum cost of serving requests 1 to i, terminating in state s. Given σ, a minimum cost solution can be found by a dynamic programming algorithm with the following functional equation:

$$\forall\ s \in S,\ i \in \mathcal{Z}^+,\quad W_i(s) = \min_{t \in S}\{W_{i-1}(t) + ser(t, \sigma_i) + tran(t, s)\},$$

with suitable initializations. Let $opt_i = \min_{s \in S} W_i(s), i \geq 1$, be the optimal cost of serving the first i requests. We call $\omega_i(s) = W_i(s) - opt_i$ the *offset function* value at state s after request i has been revealed. Define $\Delta opt_i = opt_i - opt_{i-1}$; it is the increase in the optimal off-line cost due to σ_i.

Our on-line algorithms make decisions based on the current offset values, $\omega_i(s), s \in S$. Note that to compute the $\omega_i(s)$'s and Δopt_i's, it suffices to know only the $\omega_{i-1}(s)$'s. Since $OPT(\sigma) = \sum_{i=1}^{|\sigma|} \Delta opt_i$, to show that an algorithm A is c-competitive, we need only show that for each reachable combination of offset function, request, and file

system state, the inequality $\Delta C_A + \Delta\Phi \leq c \cdot \Delta opt_i$ holds, where ΔC_A is the cost incurred by A and $\Delta\Phi$ is the change in some defined potential function. If the total change in Φ is always bounded or nonnegative, summing up the above inequality over σ, we have $C_A(\sigma) \leq c \cdot OPT(\sigma) + B$, where B is some bounded value.

The Steiner tree problem. We shall refer to a network design problem called the Steiner tree problem (STP) [24], which can be stated as follows. An instance of STP is given by a weighted undirected graph $G = (V, E)$, a weight function on the edges $w : E \longrightarrow \mathcal{Z}_0^+$, a subset $Z \subseteq V$ of *regular nodes* or *terminals*, and a constant $B' \in \mathcal{Z}^+$. The decision problem is to ask if there exists a Steiner tree in G that includes all nodes in Z and has a total edge weight of no more than B'. STP is NP-complete even when G is restricted to bipartite graphs with unit edge weights or to planar graphs [12, 16]. Surveys on STP can be found in [13, 24]. On a tree network, the union of paths between all pairs of terminals gives the optimal Steiner tree.

3. Deterministic algorithms for FAP on a tree. We begin by introducing some concepts that will be used in building both deterministic and randomized algorithms for file allocation on trees.

We say a residence set is *connected* if it induces a connected subgraph in G. On a tree, if the residence set is always connected, each node without a copy of F can easily keep track of R, and hence the nearest copy of F, by using a pointer. In fact, when G is a tree we can limit our attention to algorithms that maintain a connected R at all times.

THEOREM 3.1. *On a tree, there exists an optimal algorithm that always maintains a connected residence set; i.e., given any (on-line or off-line) algorithm A, there exists an algorithm A' that maintains a connected R and $C_{A'}(\sigma) \leq C_A(\sigma)$ for all σ. If A is on-line, so is A'.*

Proof. Let $R(A)$ and (R') be the residence sets maintained by A and A', respectively. We simulate A on σ and let A' be such that at any time, $R')$ is the minimum connected set that satisfies $R(A) \subseteq R(A')$. Given $R(A)$ on a tree, $R(A')$ is defined and unique.

Since $R(A) \subseteq R(A')$, the reading cost incurred by A' cannot be greater than that by A. The same holds true for the writing cost issued at any node v, since $R(A') \cup \{v\}$ spans the unique minimum length Steiner tree for $R(A) \cup \{v\}$. So A' does not incur a greater read or write cost than A.

Algorithm A' does not need to carry out any replication unless A does, and only to nodes that are not already in $R(A')$. To maintain $R(A) \subseteq R(A')$, A' should leave a copy of F along any replication path; this can be done without incurring any extra cost. As $R(A) \subseteq R(A')$, A' never needs to traverse a replication path longer than that traversed by A for the same replication. Hence, A' cannot incur a greater replication cost. Since a reallocation is a sequence of replications and discards of F, A' maintains a connected set at all times and does not incur a greater cost than A in the reallocation. □

Henceforth we shall consider only algorithms that maintain a connected residence set R at all times. When we say that an algorithm replicates to node v, we shall mean it leaves a copy of F, at all nodes along the shortest path from the residence set to v. In a tree network we can make some additional simplifying assumptions. Suppose an algorithm A decides to move to residence set R' from set R. This reallocation involves some sequence of replications and drops.

LEMMA 3.2. *All replications can be performed before all drops without increasing the total cost of the reallocation.*

Proof. Dropping a copy can only increase the cost of following replications. □

Henceforth we assume that all algorithms comply with Lemma 3.2.

LEMMA 3.3. *Let $S = R' \setminus R$ be the nodes that gain a copy of F. Then F can be replicated to the nodes of S in any order at total cost $D \cdot |T(R') \setminus T(R)|$, where $T(R)$ is the subtree induced by node set R.*

Proof. A copy of F must be sent across each edge in $T(R') \setminus T(R)$ at least once. But in any order of replication, a copy cannot be sent across an edge more than once, because then both endpoints contain a copy of F. □

Henceforth we assume that all algorithms comply with Lemma 3.3.

A useful tool in handling on-line optimization on trees is *factoring* [7, 10]. It makes use of the fact that any sequence of requests σ and any tree algorithm can be "factored" into $(n-1)$ individual algorithms, one for each edge. The total cost in the tree algorithm is equal to the sum of the costs in each individual edge game. For edge (a, b) we construct an instance of two-processor file allocation as follows. The removal of edge (a, b) divides T into two subtrees T_a and T_b, containing a and b, respectively. A read or write request from a node in T_a is replaced by the same kind of request from a, and a request from a node in T_b is replaced by the same request from b. Let A be an algorithm with residence set $R(A)$. Algorithm A induces an algorithm on edge (a, b) as follows: if $R(A)$ falls entirely in T_a or T_b, then the edge algorithm is in state a or b, respectively; otherwise, the edge algorithm is in state ab. When the edge algorithm changes state, it does so in the minimum cost way (i.e., at most one replication). This factoring approach is used in our algorithms for file allocation on a tree. For the rest of this paper, given an edge (a, b), we use T_a and T_b to represent the subtrees described above, s to denote the state the edge is in, and let the offset functions triplet be $\omega_i = (\omega_i(a), \omega_i(b), \omega_i(ab))$, where $\omega_i(s)$ is the offset function value of state s after σ_i has arrived.

LEMMA 3.4. *For algorithm A and request sequence σ, let $A_{(a,b)}$ be the algorithm induced on edge (a, b) and σ_{ab} be the request sequence induced on edge (a, b). Then*

$$C_A(\sigma) = \sum_{(a,b)\in E} C_{A_{(a,b)}}(\sigma_{ab}).$$

Proof. We show that the cost incurred by any event contributes the same amount to both sides of the equation.

For a write request at a node v, $C_A(\sigma)$ increases by the weight of the unique Steiner tree, T', containing nodes in $R(A) \cup \{v\}$. In the induced problem of any edge e on T', the residence set and the request node are on opposite sides of e, and a write cost equal to e's weight is incurred. For other edges, v and the residence set lie on the same side of e, and no cost is incurred in their induced problems. So both sides of the equation increase by the same amount.

For a read request at a node v, the same argument as in the write case can be used, replacing T' by the unique path from v to the nearest node with a copy of F. Both sides of the equation increase by the same amount.

Suppose A moves from a residence set of R to R', and consider the sequence of replications and discards that make up the reallocation process. We show by induction on the length of this sequence that the movement cost to A is exactly equal to the sum of movement costs in the induced edge problems. Suppose that the first action in the sequence is to replicate F to node v. The cost to A is D times the sum of the lengths of the edges on the shortest path from R to v. Since R is connected, the edges on this path are exactly the edges that must replicate in their induced problems. Thus both

TABLE 3.1
Transition and service costs.

$tran(t,s)$		s		
		a	b	ab
t	a	0	D	D
	b	D	0	D
	ab	0	0	0

$ser(t,\sigma_i)$		σ_i			
		a^r	a^w	b^r	b^w
t	a	0	0	1	1
	b	1	1	0	0
	ab	0	1	0	1

TABLE 3.2
Changes in offsets.

Case 1: $k \geq 1$.

σ_{i+1}	$\omega_{i+1}(a)$	$\omega_{i+1}(b)$	$\omega_{i+1}(ab)$	Δopt_{i+1}
a^r	0	$\min(k+1,l)$	l	0
a^w	0	$\min(k+1,D)$	$\min(l+1,D)$	0
b^r	0	$k-1$	$l-1$	1
b^w	0	$k-1$	l	1

Case 2: $k = 0$.

σ_{i+1}	$\omega_{i+1}(a)$	$\omega_{i+1}(b)$	$\omega_{i+1}(ab)$	Δopt_{i+1}
a^r	0	$\min(1,l)$	l	0
a^w	0	1	$\min(l+1,D)$	0
b^r	$\min(1,l)$	0	l	0
b^w	1	0	$\min(l+1,D)$	0

sides of the equation increase by the same amount. If the first action is a discard, then no costs are incurred by A or any of the induced edge algorithms. □

LEMMA 3.5. *Let $OPT(\sigma_{ab})$ be the cost incurred by an optimal edge algorithm for (a,b) on sequence σ_{ab}. Then $\sum_{(a,b)\in E} OPT(\sigma_{ab}) \leq OPT(\sigma)$.*

Proof. The lemma follows by letting A in Lemma 3.4 be the optimal off-line algorithm for FAP on a tree and noting $C_{A_{(a,b)}}(\sigma_{ab}) \geq OPT(\sigma_{ab})$ for any A and edge (a,b). □

It follows from Lemmas 3.4 and 3.5 that if A is an on-line algorithm such that on any σ, and for each edge (a,b), $C_{A_{(a,b)}}(\sigma_{ab}) \leq c \cdot OPT(\sigma_{ab})$ holds, then A is c-competitive.

To construct a deterministic algorithm for the tree, we first construct a suitable optimal algorithm for a single edge. We then design the tree algorithm so that it induces this optimal edge algorithm in each edge, thereby guaranteeing competitiveness.

3.1. An optimal deterministic edge algorithm. Let $G = (a,b)$ be an edge and let $S = \{a, b, ab\}$ be the set of states the file system can be in—only node a has a copy, only node b has a copy, and both a and b have a copy, respectively. We can assume G is of unit length; otherwise the offsets and cost functions can be scaled to obtain the same results. We write the offset functions as a triplet $\omega_i = (\omega_i(a), \omega_i(b), \omega_i(ab))$ and similarly for the work functions. Suppose the starting state is a. Then $W_0 = (0, D, D)$. The *ser* and *tran* functions are given in Table 3.1. By the definition of the offset functions, and since it is free to discard a copy of F, we always have $\omega_i(ab) \geq \omega_i(a), \omega_i(b)$, and at least one of $\omega_i(a)$ and $\omega_i(b)$ is zero. Without loss of generality, we assume a starting offset function vector of $\omega_i = (0, k, l)$, $0 \leq k \leq l \leq D$, after σ_i has arrived. Table 3.2 gives the changes in offsets for different combinations of requests and offsets in response to the new request σ_{i+1}.

Let s be the current state of R. Our algorithm specifies the new required residence

set, R, after σ_{i+1} has arrived and the offsets have been updated; it assumes state a is a zero-offset state.

ALGORITHM DETEDGE.

(1) **If** $s \neq ab$ and $\omega_{i+1}(s) = \omega_{i+1}(ab)$, replicate, i.e., set $s = ab$.

(2) **If** $s = ab$ and $\omega_{i+1}(b) = D$, drop at b, i.e., set $s = a$.

THEOREM 3.6. *Algorithm* **DetEdge** *is strongly* 3*-competitive.*

Proof. We first show that for each request σ_j, $\Delta C_{Edge} + \Delta\Phi \leq 3 \cdot \Delta opt_j$ $(*)$ holds for some function $\Phi(\cdot)$ defined below. Let a be a zero-offset state, and we have $\omega_i = (0, k, l)$. At any time, we define the potential function

$$\Phi(s,k) = \begin{cases} 2 \cdot D - 2 \cdot k & \text{if } s = a, \\ 2 \cdot D - k & \text{if } s = b, \\ D - k & \text{if } s = ab. \end{cases}$$

Initially, $\Phi = 0$, and we always have $\Phi \geq 0$. When $\omega_i = (0, 0, l)$ and $s \neq ab$, s can be considered to be in state a or b, and $\Phi(a, 0) = \Phi(b, 0) = 2D$. Note that $\omega_i = \omega_{i+1}$ and $\Delta\Phi = \Delta opt_i = 0$ hold in the following cases:

(i) $\omega_i = (0, D, D)$ and $\sigma_{i+1} = a^r$ or a^w;

(ii) $\omega_i = (0, l, l), l \geq 1$, and $\sigma_{i+1} = a^r$; and

(iii) $\omega_i = (0, 0, 0)$ and $\sigma_{i+1} = a^r$ or b^r.

Our algorithm ensures that $\Delta C_{Edge} = 0$ in these cases. Let us show that $(*)$ holds for all possible combinations of state, request, and offset. The offsets and state variables below are the ones *before* the new request σ_{i+1} arrives. We consider the $k \geq 1$ cases; the $k = 0$ cases are similar to that when $k \geq 1$ and $\sigma_{i+1} = a^r$ or a^w.

Case 1. $\sigma_{i+1} = a^r$.

We have $\Delta opt_{i+1} = 0$. If $s = a$ or ab, then the left-hand side (L.H.S.) of $(*) \leq 0$ and $(*)$ holds. If $s = b$, by the last execution of the algorithm, we must have $k < l$. Then $\Delta C_{Edge} = -\Delta\Phi = 1$, and $(*)$ holds.

Case 2. $\sigma_{i+1} = a^w$.

We have $\Delta opt_{i+1} = 0$. If $s = a$, then L.H.S $(*) \leq 0$ and $(*)$ holds. If $s = b$ or ab, we must have $k < D$. Then $\Delta C_{Edge} = -\Delta\Phi = 1$, and $(*)$ holds.

Case 3. $\sigma_{i+1} = b^r$ or b^w.

We have $\Delta opt_{i+1} = 1$. In this case $\Delta C_{Edge} \leq 1$, $\Delta\Phi \leq 2$, and L.H.S. $(*) \leq 3$ hold.

Inequality $(*)$ also holds when **DetEdge** changes state. When **DetEdge** moves from state ab to state a, $\omega_i = (0, D, D)$ and $\Delta\Phi = \Delta C_{Edge} = 0$; and when **DetEdge** moves to state ab, $\Delta\Phi = -\Delta C_{Edge} = -D$. Hence, $(*)$ holds for all possible combinations of offsets, requests, and residence sets.

We claim that no deterministic algorithm is better than 3-competitive for FAP on an edge. For migration, it is known that no deterministic algorithm can be better than 3-competitive on a single edge [7]. We show that given any on-line algorithm A for FAP there exists another on-line algorithm A' such that (i) $C_{A'}(\sigma) \leq C_A(\sigma)$ for any σ with only write requests; (ii) A' always keeps only one copy of F at a node in A's residence set; and (iii) whenever A has only one copy of F, A' has a copy at the same node. Since A' is a legal algorithm for any instance of the migration problem and since the optimal cost to process σ without using replications is no less than the optimal cost with replications, A is c-competitive on write-only sequences only if A' is a c-competitive migration algorithm. This implies the claim.

Algorithm A' is obtained from A as follows. Initially, both A and A' have a copy of F at the same node. The following rules are applied whenever A changes state:

(1) If A replicates, A' does not change state.
(2) If A migrates, A' follows.
(3) If A drops a page, A' follows to the same node.
It follows from the rules above that (ii) and (iii) hold, and A' cannot incur a write cost higher than that of A. Each movement of A' in (1) or (2) corresponds to a distinct migration or earlier replication by A, respectively. So A' cannot incur a higher movement cost than A. The claim follows. □

3.2. An optimal deterministic tree algorithm. Recall that for each edge $e = (a, b)$ on the tree, request sequence σ induces a sequence σ_{ab} on (a, b). The tree algorithm is based on factoring into individual edge subproblems and simulating **DetEdge** on each subproblem. After $r \in \sigma$ is served, for each edge $e = (a, b)$ the induced request r_{ab} is computed and the offset vector for the induced subproblem is updated. The following algorithm is then executed, updating the residence set, $R(Tree)$. Initially $R(Tree)$ consists of the single node containing F.

ALGORITHM DETTREE.

(1) Examine each edge (u, v) in any order, and simulate the first step of Algorithm **DetEdge** in the induced subproblem. If **DetEdge** replicates to one of the nodes, say v, in the induced subproblem, then add v to $R(Tree)$ and replicate to v.

(2) Simulate step 2 of **DetEdge** for all edges. For any node v, if the edge algorithm for an incident edge $e = (u, v)$ requires deleting node v from e's residence set in e's induced problem, mark v.

(3) Drop at all marked nodes.

To show that **DetTree** is 3-competitive, we will show that it chooses a connected residence set and, for each edge, it induces the state required by **DetEdge**. This is not immediately obvious because the requirements of **DetEdge** on one edge might conflict with those on another edge. For example, one edge might want to drop a copy that another edge has just replicated.

We begin by analyzing the structure of the offset functions in the induced edge problems. For the rest of this subsection, the offset values and functions for each edge (a, b) refer to results from the induced sequence σ_{ab}. The next lemma characterizes the offset distribution between two adjacent edges.

LEMMA 3.7. *The following properties hold:*

(A) *At any time, there exists a root node r such that $R = \{r\}$ corresponds to a zero-offset state in the induced problems of all edges.*

(B) *For any edge (x, y) on the tree, define $S_i(x, y) = \omega_i(xy) - \omega_i(x)$. Then for any adjacent edges (x, y) and (y, z), the inequality $S_i(x, y) \le S_i(y, z)$ holds $\forall$ i.*

Following from the earlier definitions (see the beginning of section 3.2), the claim (A) above states that there is a node r such that for any edge (a, b), where a is nearer to r than b, state a is a zero-offset state for the edge. Note that the location of the root node r may not be unique, and its location changes with requests. The lemma implies the following conditions.

COROLLARY 3.8.

(C) *Let (x, y) be an edge in T such that a root r is in T_x. Let $z \neq x$ be a neighbor of y, and edges (x, y) and (y, z) have offsets $(0, k_{xy}, l_{xy})$ and $(0, k_{yz}, l_{yz})$, respectively. Then*

(C.1) *$l_{xy} \le l_{yz}$;*
(C.2) *$l_{xy} - k_{xy} \ge l_{yz} - k_{yz}$;*
(C.3) *$k_{xy} \le k_{yz}$; and*
(C.4) *if $k_{yz} = 0$, then $k_{xy} = 0$ and $l_{xy} = l_{yz}$ hold.*

(D) *Let* (x, y) *and* (y, z) *be adjacent edges with a root* r *in the subtree that is rooted at* y *and formed from removing the two edges from* T*. Let the offsets in the edges be* $(k_{xy}, 0, l_{xy})$ *and* $(0, k_{yz}, l_{yz})$*, respectively. Then* $l_{xy} \geq (l_{yz} - k_{yz})$ *holds.*

Proof of Lemma 3.7. We use induction on the number of requests. Initially, let r be the node holding the single copy of F; all the edges have offset vectors $(0, D, D)$, and the lemma holds trivially. We assume the lemma holds for $t \in \mathcal{Z}_0^+$ revealed requests and show that it remains valid after σ_{t+1} has arrived at a node w. We first show how to locate a new root. Let P represent the path from r to w. Unless specified otherwise, the offsets referred to below are the ones *before* σ_{t+1} arrives. We choose the new root, r', using the following procedure.

PROCEDURE FINDROOT.

(1) **If** (i) $w = r$ or (ii) $w \neq r$ and all the edges along P have offsets of the form $(0, k, l)$, $k \geq 1$. **Then** $r' = r$.

(2) **Otherwise**, move along P from r toward w, and cross an edge if it has offset vector of the form $(0, 0, l)$ until we cannot go any further or when w is reached. Pick the node where we stop as r'.

Let us show that r' is a valid root for the new offsets. We picture P as a chain of edges starting from r, going from left to right, and ending in w. If the condition in step (1) of the algorithm is satisfied, σ_{t+1} corresponds to a request at the zero-offset state for all edges. By Table 3.2, r remains a valid root node. Suppose (2) above is executed. For any edge that is not on P, or is on P but is to the right of r', its zero-offset state remains the same. Node r' is a valid root node for these edges. By (C.4), edges along P with offsets of the form $(0, 0, l)$ must form a connected subpath of P, starting from r and ending in r'. They have the same value for the parameter l. By Table 3.2, their offsets change from $(0, 0, l)$ to $(1, 0, \min\{l + 1, D\})$ or $(\min(1, l), 0, l)$, and r' is a valid root node for them. Hence (A) holds for our choice of r' above.

To show that (B) holds, we consider any two adjacent edges (x, y) and (y, z) whose removal will divide T into three disjoint subtrees: T_x, T_y, and T_z, with roots x, y, and z, respectively. We show that for different possible positions of r and w, (B) remains valid after σ_{t+1} has arrived; i.e., $S_{t+1}(x, y) \leq S_{t+1}(y, z)$ holds when σ_{t+1} is a *write* or a *read*, when $r \in T_x, T_y$, or T_z, and when $w \in T_x, T_y$, or T_z. We assume **(B)** holds before σ_{t+1} arrives.

Suppose σ_{t+1} is a read request, $r \in T_x$, and $w \in T_x$. For edge (x, y), $\omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = (0, \min(k_{xy} + 1, l_{xy}), l_{xy})$. For edge (y, z), $\omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz} + 1, l_{yz}), l_{yz})$. Inequality $S_{t+1}(x, y) \leq S_{t+1}(y, z)$ follows from $S_t(x, y) \leq S_t(y, z)$ or (C.1). Condition (B) can be shown to hold in other situations by a similar case analysis. Please refer to the appendix for the complete case analysis. Thus (B) holds for request $(t + 1)$, and the lemma follows. □

THEOREM 3.9. *Algorithm* **DetTree** *is strongly* 3*-competitive.*

Proof. We show that **DetTree** induces **DetEdge** on each tree edge. The theorem then follows from Lemmas 3.4 and 3.5 and Theorem 3.6.

We proceed by induction on the number of requests. Initially, $R(Tree)$ consists of a single node. Suppose $R(Tree)$ is connected after the first $t \in \mathcal{Z}_0^+$ requests, and for each edge (a, b), the state induced by $R(Tree)$ is equal to the state desired by **DetEdge** when run on σ_{ab}. Consider the processing of request $t + 1$.

Step (1): *Replication.* We do a subinduction on the number of replications done in Step (1) and show that no replication is in conflict with the state desired by any edge.

Suppose that processing edge (a, b) in Step (1) causes F to be replicated to a.

Then $r \in T_a$, $R(Tree)$ lies in T_b, inducing state $s = b$, and $\omega_{t+1} = (0, l, l)$. This follows from the definition of **DetEdge**, the definition of the induced subproblem, and the inductive hypothesis. Let Q be the path from b to the nearest node in $R(Tree)$. If $Q \neq \{b\}$, then, to avoid conflict, each edge along Q must also require replication across it. From (A) and (C.2) in Theorem 3.7, we see that each edge (x, y) in Q has an offset of the form $\omega_{t+1} = (0, l', l')$, where x is nearer to b than y and requires a replication. A similar argument holds for the case when $s = a$ and $\omega_{t+1} = (0, 0, 0)$.

Step (2): *Marking nodes to drop.* Again we perform a subinduction on the number of markings done in Step (2) and show that no marking is in conflict with the state desired by any edge and that a connected residence set results.

Suppose that processing edge (a, b) in Step (2) causes b to be marked. This occurs because (a, b) has $\omega_{t+1} = (0, D, D)$, $r \in T_a$, r a write, and $s = ab$.

Since $R(Tree)$ is connected by hypothesis, both a and b are in $R(Tree)$, and the nodes in T_b with a copy of F span a connected subtree of T_b, with b as its root. Let us call it T'_b. If $T'_b \neq \{b\}$, each edge (x, y) in T'_b is in state $s = xy$. By (A) and (C.3) in Theorem 3.7, (x, y) must have offset $\omega_{t+1} = (0, D, D)$, with x nearer to b than y is. Under **DetEdge**, (x, y) needs to drop the copy of F in node y. Hence all the nodes in T'_b are required to be removed from $R(Tree)$, the new $R(Tree)$ remains connected, and no edges are in conflict.

Thus $R(Tree)$ is connected, all induced edge algorithms match **DetEdge**, and **DetTree** is 3-competitive. □

4. Randomized algorithms for FAP on a tree. Our approach to building a randomized tree algorithm is the same as our approach in the deterministic case. We give a randomized algorithm for a two-point space, **RandEdge**, that is based on counter values assigned at the nodes. By factoring, we obtain from **RandEdge** a $(2+1/D)$-competitive algorithm, **RandTree**, for file allocation on a tree. **RandTree** requires the generation of only $O(\log D)$ random bits at the beginning of the algorithm, after which it runs completely deterministically. It is simpler than the tree algorithm in [19], which can require the generation of $\Omega(\log D)$ random bits after each request is served.

4.1. An optimal randomized edge algorithm, RandEdge. Let edge $e = (a, b)$. We maintain counters c_a and c_b on nodes a and b, respectively. They satisfy $0 \leq c_a, c_b \leq D$ and $(c_a + c_b) \geq D$. Our algorithm maintains a distribution of R dependent on the counter values. Initially, the node with a copy of F has counter value D, and the other node has counter value 0. The counter values change according to the following rules. On a read request at a, we increment c_a if $c_a < D$. On a write request at a, if $(c_a + c_b) > D$, we decrement c_b; if $(c_a + c_b) = D$ and $c_a < D$, we increment c_a. The counters change similarly for a request at b. There is no change in the counter values in other cases.

Algorithm **RandEdge** always maintains a distribution of R such that

$$p_e[a] = 1 - \frac{c_b}{D}, \tag{4.1a}$$

$$p_e[b] = 1 - \frac{c_a}{D}, \quad \text{and} \tag{4.1b}$$

$$p_e[ab] = \frac{c_a + c_b}{D} - 1. \tag{4.1c}$$

Observe that the probability of having a copy of F at node $v \in \{a, b\}$ is c_v/D.

In order to maintain this distribution, **RandEdge** simulates D deterministic algorithms, numbered from 1 to D. The moves of each deterministic algorithm are constructed (deterministically) on-line, according to rules given below. Before the first request, one of the D algorithms is picked at random. **RandEdge** then makes the same moves as the chosen deterministic algorithm. Thus $p_e[s]$, $s \in \{a, b, ab\}$, is the proportion of algorithms in state s, and the expected cost incurred by **RandEdge** is the average of the costs incurred by the D algorithms.

We define the D algorithms that achieve the probability distribution in (4.1). Suppose that initially only node a has a copy of F. Then initially the D algorithms are placed in state a. The following changes are made after a new request, σ_i, has arrived. Without loss of generality, we assume the request arises at node a. (The c_a and c_b values below refer to the counter values just *before* σ_i arrives.)

- There is no change in the algorithms if there is no change in the counter values.
- *Case* 1. If $\sigma_i = a^r$ and $c_a < D$, the lowest-numbered algorithm in state b moves to state ab.
- *Case* 2. If $\sigma_i = a^w$, $(c_a + c_b) > D$, the lowest-numbered algorithm in state ab moves to state a.
- *Case* 3. If $\sigma_i = a^w$, $(c_a + c_b) = D$, and $c_a < D$, the lowest-numbered algorithm in state b moves to state ab.

LEMMA 4.1. **RandEdge** *is feasible and maintains the probability distribution in* (4.1).

Proof. By feasible we mean that whenever a move must be made in Cases 1, 2, and 3, there is some algorithm available to make the move. The choice of lowest-numbered algorithm is only to emphasize that the choice must be independent of which algorithm **RandEdge** is actually emulating.

The lemma holds initially with $c_a = D$ and $c_b = 0$. We prove the lemma by induction on the requests and assume it holds before σ_i arrives. If there is no change in counter values after σ_i has arrived, the lemma holds trivially. By the induction hypothesis, in Case 1 above, since $c_a < D$ and $p_e[b] > 0$, at least one of the D algorithms is in state b; in Case 2, since $(c_a + c_b) > D$ and $p_e[ab] > 0$, there is an algorithm in state ab; in Case 3, since $c_a < D$, there is an algorithm in state b. Hence, **RandEdge** is feasible. It can be verified that the changes in the algorithms implement the probability distribution in (4.1) for the new counter values. ∎

THEOREM 4.2. **RandEdge** *is strongly* $(2 + 1/D)$*-competitive.*

Proof. For each node $v \in \{a, b\}$, we maintain the potential function

$$\phi_v = \begin{cases} \frac{D+1}{2} + \sum_{j=c_v}^{D-1} (2 - \frac{j}{D}), & \mathcal{OPT} \text{ has a copy of } F \text{ at } v, \\ \sum_{j=1}^{c_v} \frac{j}{D} & \text{otherwise,} \end{cases}$$

where $\mathcal{OPT}$ represents the adversary. Let the overall potential function $\Phi = \phi_a + \phi_b - (D + 1)/2$. Initially, $\Phi = 0$; at any time, $\Phi \geq 0$. We show that in response to each request and change of state,

$$\mathbf{E}(\Delta C_{\mathbf{RandEdge}}) + \mathbf{E}(\Delta M_i) + \Delta\Phi \leq (2 + 1/D) \cdot \Delta OPT \tag{4.2}$$

holds, where ΔOPT, $\mathbf{E}[\Delta C_{\mathbf{RandEdge}}]$, and $\mathbf{E}(\Delta M_i)$ are the cost incurred by the event on $\mathcal{OPT}$ and the service and movement costs incurred on **RandEdge**, respectively. The c_a and c_b values below are the counter values just before the new request σ_i arrives.

Case 1. Request $\sigma_i = a^r$.

If $c_a = D$, inequality (4.2) holds trivially. Suppose $c_a < D$. We have

$$\mathbf{E}(\Delta C_{\mathbf{RandEdge}}) = 1 - \frac{c_a}{D}, \quad \mathbf{E}(\Delta M_i) = 1,$$

$$\Delta\Phi = \begin{cases} -2 + \frac{c_a}{D}, & \mathcal{OPT} \text{ has a copy of } F \text{ at } a, \\ \frac{c_a+1}{D} & \text{otherwise.} \end{cases}$$

It follows that if $\mathcal{OPT}$ has a copy of F at a when σ_i arrives, L.H.S. (4.2) $= \Delta OPT = 0$; otherwise, L.H.S. (4.2) $= (2 + 1/D) = (2 + 1/D) \cdot \Delta OPT$. Inequality (4.2) holds.

Case 2. Request $\sigma_i = a^w$ and $(c_a + c_b) > D$.

We have

$$\mathbf{E}(\Delta C_{\mathbf{RandEdge}}) = \frac{c_b}{D}, \quad \mathbf{E}(\Delta M_i) = 0,$$

$$\Delta\Phi = \begin{cases} 2 - \frac{c_b-1}{D}, & \mathcal{OPT} \text{ has a copy of } F \text{ at } b, \\ \frac{-c_b}{D} & \text{otherwise, and} \end{cases}$$

$$\text{L.H.S. (4.2)} = \begin{cases} 2 + \frac{1}{D}, & \mathcal{OPT} \text{ has a copy of } F \text{ at } b, \\ 0 & \text{otherwise.} \end{cases}$$

Inequality (4.2) holds.

Case 3. Request $\sigma_i = a^w$ and $(c_a + c_b) = D$.

If $c_a = D$, L.H.S. (4.2) $= 0$ and (4.2) holds trivially. Suppose $c_a < D$. We have

$$\mathbf{E}(\Delta C_{\mathbf{RandEdge}}) = 1 - \frac{c_a}{D}, \quad \mathbf{E}(\Delta M_i) = 1,$$

$$\Delta\Phi = \begin{cases} -2 + \frac{c_a}{D}, & \mathcal{OPT} \text{ has a copy of } F \text{ at } a, \\ \frac{c_a+1}{D} & \text{otherwise, and} \end{cases}$$

$$\text{L.H.S. (4.2)} = \begin{cases} 0, & \mathcal{OPT} \text{ has a copy of } F \text{ at } a, \\ 2 + \frac{1}{D} & \text{otherwise.} \end{cases}$$

Hence, (4.2) holds.

Case 4. $\mathcal{OPT}$ changes state.

When $\mathcal{OPT}$ changes state, $\mathbf{E}(\Delta C_{\mathbf{RandEdge}}) = \mathbf{E}(\Delta M_i) = 0$. It can be checked from the definition of Φ that when $\mathcal{OPT}$ replicates, $\Delta OPT = D$ and $\Delta\Phi \leq (2D + 1)$ hold; when $\mathcal{OPT}$ discards a copy of F, $\Delta\Phi \leq 0$.

Since (4.2) holds for all possible events, by Theorem 4.6 **RandEdge** is strongly $(2 + 1/D)$-competitive. □

4.2. An optimal randomized tree algorithm—RandTree. We extend **RandEdge** to a randomized algorithm for FAP on a tree, T, by means of factoring. Our algorithm, **RandTree**, induces **RandEdge** on each edge for the induced request sequence for the edge.

Description of algorithm RandTree. RandTree internally simulates D deterministic algorithms. Each of them maintains a residence set that spans a subtree of T. Initially, the residence set for each of them is the single node that contains F. One of the D simulated algorithms is picked uniformly at random at the beginning, and **RandTree** behaves exactly the same as the particular algorithm chosen.

We maintain counters c_a and c_b for each edge (a, b) in the tree. Using the factoring approach (see section 3.2), we obtain an induced request sequence σ_{ab} for (a, b). The counter values change according to the same rules as described in the single edge case (section 4.1), using σ_{ab}. **RandTree** responds to each request and maintains an (induced) distribution as required by **RandEdge** in (4.1) for each of the edges.

Read request. Suppose the new request, σ_i, is a *read* request at a node g. Let T be rooted at g and $e = (a, b)$ be an edge with a nearer to g than b is. The c_a and c_b values described below are the counter values before σ_i arrives. The edges can be classified into three types:

- type 1: edges with $c_a = D$ and $c_b = 0$;
- type 2: edges with $c_a = D$ and $c_b > 0$; and
- type 3: edges with $c_a < D$.

RandEdge requires no change in probability values for the first two types of edges; for type 3 edges, it requires that $p_e[b]$ decreases by $1/D$ and $p_e[ab]$ goes up by $1/D$. For any node v, we use $T(v)$ to denote the subtree of T rooted at v. **RandTree** changes the subtree configurations maintained by the D algorithms by using the following procedure (Fig. 4.1).

(1) Let $\mathcal{F}$ be the forest of trees formed by all the type 3 edges.
(2) **While** there exists a tree $T' \in \mathcal{F}$ with at least one edge, **Do**
  (2.1) Let x be a leaf node in T' and P be the path from x to the root node of T'.
  (The root node of T' is the node in T' that is nearest to g.)
  (2.2) Pick any one of the D algorithms that maintains a subtree, Z, that includes node x and lies entirely in $T(x)$.
  Make that algorithm replicate along P, i.e., replace Z by $Z \cup P$.
  (2.3) Remove the edges in P from T' and update the forest $\mathcal{F}$.

FIG. 4.1. *Algorithm RandEdge (read requests).*

LEMMA 4.3. **RandTree** *implements the required changes for all the edges for a read request.*

Proof. We prove the lemma by induction on the requests. Suppose that **RandTree** induces **RandEdge** on all edges before σ_i arrives. Let y be the parent node of x. If **RandTree** is feasible, i.e., it can be executed, it implements the changes required by **RandEdge** as described above for all edges. We show that this is the case.

If x is a leaf node of T, since (x, y) is a type 3 edge, $p_{(x,y)}[x] > 0$ and one of the D algorithms must have the single node $\{x\}$ as its tree configuration.

Otherwise, suppose all the descending edges of x are of type 1. Let (x, w) be one of them. Then $p_{(x,w)}[xw] = p_{(x,w)}[w] = 0$; none of the algorithms maintain a subtree with any edge in $T(x)$. Since $p_{(x,y)}[x] > 0$, one of the algorithms must have $\{x\}$ as its subtree.

Otherwise, suppose x has descending type 2 edges. Let (x, w) be any one of them. Then $p_{(x,w)}[w] = 0$ and $p_{(x,w)}[xw] > 0$. Thus each of these edges is contained in the

subtree of at least one of the algorithms, and none of the algorithms has its subtree in $T(w)$. Since $p_{(x,y)}[x] > 0$, at least one of these subtrees must lie in $T(x)$ and contains node x.

Hence, our algorithm is feasible and the lemma holds. □

Write request. Suppose $\sigma_i = r^w$. We use the same notation as in the read request case. The edges can be classified into three types:

- type 1: edges with $(c_a + c_b) > D$;
- type 2: edges with $(c_a + c_b) = D$ and $c_a < D$; and
- type 3: edges with $(c_a + c_b) = D$ and $c_a = D$.

RandEdge requires no change in probability values for the type 3 edges; for type 1 edges, it requires that $p_e[ab]$ decreases by $1/D$ and $p_e[a]$ increases by the same amount; for type 2 edges, it requires that $p_e[b]$ decreases by $1/D$ and $p_e[ab]$ increases by the same amount. **RandTree** performs the following (Fig. 4.2).

(1) Let $\mathcal{F}$ be the forest of trees formed by all the type 1 edges.
(2) While there exists a tree $T' \in \mathcal{F}$ with at least one edge, **Do**
  (2.1) Let g' be the root node of T' and x be one of its children nodes in T'.
  (2.2) Pick an algorithm that has a subtree Z that includes edge (g', x).
  (2.3) If Z is contained in T', make the algorithm replace Z by the single-node subtree $\{g'\}$; otherwise, replace Z by the tree formed by edges in $(Z - T')$.
  (2.4) Replace T' in $\mathcal{F}$ by the subtrees formed by $T' - Z$.
(3) Let $\mathcal{F}$ be the forest of trees formed by all the type 2 edges.
(4) **While** there exists a tree $T' \in \mathcal{F}$ with at least one edge, **Do**
  (4.1) Let x be a leaf node of T' and P be the path from x to the root node of T'.
  (4.2) Pick any one of the D algorithms that maintains a subtree, Z, that includes node x and lies entirely in $T(x)$.
  Make that algorithm replicate along P, i.e., extends Z to $Z \cup P$.
  (4.3) Remove the edges in P from T' and update the forest $\mathcal{F}$.

FIG. 4.2. *Algorithm RandEdge (write requests).*

LEMMA 4.4. **RandTree** *implements the required changes for all the edges for a write request.*

Proof. We prove by induction and assume **RandEdge** is induced on all the edges before σ_i arrives. If **RandTree** is feasible, it implements the required changes for all the edges. We show that this is the case.

Consider the first loop of the algorithm (in step (2)). Since $p_{(g',x)}[g'x] > 0$, subtree Z must exist. **RandTree** removes edges from Z that are contained in T'. Note that the $p_e[ab]$ values for type 2 and 3 edges are zero; **RandTree** processes edges in T' in a top-down fashion, and configuration $(Z - T')$ is always a connected subtree. Thus the first loop can be executed.

Consider the second loop of the algorithm (in step (4)). Let y be a parent node of x. Then $p_{(x,y)}[xy] = 0$ and $p_{(x,y)}[x] > 0$ hold. If x is a leaf node in T, one of the D algorithms must have $\{x\}$ as its subtree. If x has a descending type 1 edge, by the first part of the algorithm, one of the D algorithms must have $\{x\}$ as its subtree after running the first loop of the algorithm. Suppose all the descending edges of x

are type 3 edges. Let (x, w) be a type 3 edge; then $p_{(x,w)}[w] = p_{(x,w)}[xw] = 0$ and $p_{(x,w)}[x] = 1$. Since $p_{(x,y)}[x] > 0$, one of the algorithms must have $\{x\}$ as its subtree.

Hence, the algorithm is feasible and the lemma follows. □

Lemmas 4.3 and 4.4 imply that **RandTree** induces **RandEdge** on all the edges. Theorem 4.5 follows from the above lemmas and Theorem 4.2.

THEOREM 4.5. *Algorithm* **RandTree** *is strongly* $(2 + 1/D)$*-competitive for FAP on a tree against an oblivious adversary.*

4.3. Lower bound. We show that the competitive ratio, $(2 + 1/D)$, obtained above is the best possible for file allocation against an oblivious adversary, even if G is a single edge.

THEOREM 4.6. *No on-line algorithm for the file allocation problem on two points* (a, b) *is* c*-competitive for any* $c < (2 + 1/D)$.

Proof. Let A be any randomized algorithm for the file allocation problem on two points. We define a potential function Ψ and give a strategy for generating adversary request sequences such that

(i) for any C there is a request sequence σ with optimum cost $\geq C$;

(ii) the cost to **RandEdge** on σ is at least $(2 + 1/D)OPT(\sigma) - B''$, for B'' bounded independent of σ;

(iii) Ψ is bounded; and

(iv) for each request generated by this adversary,

$$\Delta C_A + \Delta\Psi \;\geq\; \Delta C_{\mathbf{RandEdge}}. \tag{4.3}$$

If conditions (ii), (iii), and (iv) hold for an adversary sequence σ, then summing (4.3) over the sequence gives

$$C_A(\sigma) \geq (2 + 1/D) \cdot OPT(\sigma) - B,$$

where B is bounded. By condition (i), the adversary can make $OPT(\sigma)$ arbitrarily large, so there is no constant B' independent of σ such that $C_A(s) < (2 + 1/D) \cdot OPT(\sigma) + B'$. Hence A cannot be c-competitive for $c < 2 + 1/D$.

We now define the adversary's strategy. We assume that both the on-line and off-line algorithms start with a single copy of F at a. Our adversary will generate only requests that result in offset functions of the form $(0, i, i)$, where $0 \leq i \leq D$. A *zero-cost self-loop* is a request such that the offset function is unchanged and $\Delta opt = 0$. By a theorem of [18], there is always an optimal on-line algorithm that incurs zero expected cost on a zero-cost self-loop. We assume A has this property. This simplifies the adversary's strategy, although the result can still be proved without this assumption.

Suppose that the current offset function is $(0, i, i)$, and let p_i be probability that **RandEdge** is in state a. Suppose A is in state a with probability q. If $q < p_i$, the adversary requests a^w; otherwise the adversary requests b^r. When $i = D$ we will have $q = 1$ (a^w is a zero-cost self-loop if $i = D$), and so the adversary will request b^r. Similarly, when $i = 0$, $q = 0$ (a^r is a zero-cost self-loop) and the adversary requests a^w. Therefore the adversary can always generate a next request using the above rules, and the request sequence can be made arbitrarily long. Since there are only D offset functions that can be generated by this strategy, an arbitrarily long sequence of requests must cycle through the offset functions arbitrarily often. Notice, however, that the only cycles which cost $\mathcal{OPT}$ nothing are zero-cost self-loops. Since the adversary never uses these requests, all cycles have nonzero cost, so by continuing

long enough the adversary can generate request sequences of arbitrarily large optimum costs. Hence condition (i) holds.

Next we consider condition (ii). Recall c_a and c_b, the counter values maintained by **RandEdge**. We claim that if the offset function is $(0, i, i)$, then $c_a = D$ and $c_b = D - i$. This is true initially, when F is located only at a, and $i = D$. By inspection of **RandEdge** one can verify that whenever the adversary generates request b^r, c_b increases by 1, and that whenever the adversary generates a^w, c_b decreases by 1. Hence $p_i = i/D$ and the expected movement cost incurred by **RandEdge** is 1 on b^r and 0 on a^w. With reference to the proof of Theorem 4.2, note that the amortized cost to **RandEdge** is exactly $2 + 1/D$ times the cost to $\mathcal{OPT}$ on any request that the adversary might generate, assuming $\mathcal{OPT}$ does not move following the request. (The adversary never generates b^r if $c_b = D$ or a^w if $c_b = 0$.) The amortized cost incurred by **RandEdge** is therefore exactly $2 + 1/D$ times the cost incurred by an "optimum" algorithm that only ever has a copy of F at a. It is possible to show that this cost really is optimum for our sequences, but in any case it is certainly lower-bounded by the true optimum cost, and so (ii) holds.

Now define Ψ to be $D \cdot \max\{0, q - p_i\}$. This is trivially bounded by D, so (iii) holds. Finally, we must verify (4.3).

Case 1. The adversary requests a^w.

In this case the new offset function must be $(0, i + 1, i + 1)$. Suppose that after the request A has mass q' at a. Then $\Delta C_A = 1 - q + D \cdot \max\{0, q - q'\}$, $\Delta\Psi = D \cdot \max\{0, q' - p_{i+1}\} - D \cdot \max\{0, q - p_i\}$, and $\Delta C_{\mathbf{RandEdge}} = 1 - p_i$. Since $q < p_i < p_{i+1}$,

$$\begin{aligned}\Delta C_A + \Delta\Psi &= 1 - q + D \cdot \max\{0, q - q'\} + D \cdot \max\{0, q' - p_{i+1}\} - D \cdot \max\{0, q - p_i\} \\ &\geq 1 - q \\ &\geq 1 - p_i \\ &= \Delta C_{\mathbf{RandEdge}}.\end{aligned}$$

Case 2. The adversary requests b^r.

In this case the new offset function must be $(0, i - 1, i - 1)$. Suppose that after the request, A has mass q' at a. Then $\Delta C_A = q + D \cdot \max\{0, q - q'\}$, $\Delta\Psi = D \cdot \max\{0, q' - p_{i-1}\} - D \cdot \max\{0, q - p_i\}$, and $\Delta C_{\mathbf{RandEdge}} = p_i + 1$. Since $q \geq p_i > p_{i-1}$,

$$\begin{aligned}\Delta C_A + \Delta\Psi &= q + D \cdot \max\{0, q - q'\} + D \cdot \max\{0, q' - p_{i-1}\} - D \cdot \max\{0, q - p_i\} \\ &\geq q + D(q - p_{i-1}) - D \cdot \max\{0, q - p_i\} \\ &= q + D\,(p_i - p_{i-1}) \\ &= q + 1 \\ &\geq p_i + 1 \\ &= \Delta C_{\mathbf{RandEdge}}. \qquad \square\end{aligned}$$

5. Migration on a uniform network. We give a $(2+1/(2D))$-competitive randomized algorithm against an oblivious adversary for migration on a uniform network. This competitive ratio is optimal even for a single edge [10]. Let G be a complete graph on n nodes labeled 1 to n. Initially, only node 1 has a copy of F. Our algorithm is based on the offsets calculated on-line. Let $S = \{1, \ldots, n\}$ and the algorithm is in state s if the single copy of F is at node s. We have the cost functions

$$ser(t, \sigma_i) = \begin{cases} 0 & \text{if } t = \sigma_i, \\ 1 & \text{otherwise,} \end{cases} \qquad tran(t, s) = \begin{cases} 0 & \text{if } t = s, \\ D & \text{otherwise,} \end{cases}$$

and initially $W_0 = (0, D, \ldots, D)$.

Suppose the ith request is served and the new offset for each node, s, is calculated. Let $v_s = D - \omega_i(s)$, $k = \max\{j \in S | \sum_{m=1}^{j} v_m < 2D\}$, and $\delta = 2D - \sum_{m=1}^{k} v_m$.

ALGORITHM MIGRATE. The algorithm maintains the probability, $p[s]$, that a node, s, contains F as follows:

$$\text{If } k < n, \quad p[s] = \begin{cases} v_s/2D & \text{if } s \le k, \\ \delta/(2D) & \text{if } s = k+1, \\ 0 & \text{otherwise.} \end{cases}$$

$$\text{If } k = n, \quad p[s] = \begin{cases} v_s/2D & \text{if } s \le k \text{ and } v_s < D, \\ (v_s + \delta)/(2D) & \text{if } s \le k \text{ and } v_s = D, \\ 0 & \text{otherwise.} \end{cases}$$

After a new request has arrived, our algorithm moves to different states with transition probabilities that minimize the total expected movement cost while maintaining the new required distribution.

THEOREM 5.1. *Given any σ, the expected cost incurred by* **Migrate**, $\mathbf{E}[C_{mig}(\sigma)]$, *satisfies*

$$\mathbf{E}[C_{mig}(\sigma)] \le [2 + 1/(2D)] \cdot OPT(\sigma).$$

Proof. We show that after a request has arrived,

$$\mathbf{E}[\Delta C_{mig}] + \mathbf{E}[\Delta M] + \Delta\Phi \le \left(2 + \frac{1}{2D}\right) \cdot \Delta opt_i \tag{5.1}$$

holds, where $\mathbf{E}[\Delta C_{mig}]$ is the expected cost incurred by **Migrate**, $\mathbf{E}[\Delta M]$ is the expected movement cost, and $\Delta\Phi$ is the change in the potential function:

$$\Phi = \sum_{s=1}^{n} \sum_{j=1}^{v_s} \left(\frac{1}{2} + \frac{j}{2D}\right) - (3D+1)/4.$$

Initially, $\Phi = 0$. At any time, since at least one $v_s = D$, we have $\Phi \ge 0$.

An offset table similar to Table 3.2 for file allocation can be constructed. Since migration is equivalent to FAP with only write requests, it can be seen that if request σ_{i+1} is at a node s with $\omega_i(s) = 0$, we have $\omega_{i+1}(s) = 0$, the offsets for all other states will increase by one, subject to a maximum of D, and $\Delta opt_{i+1} = 0$. If σ_{i+1} is at a node s with $\omega_i(s) > 0$, the offset for state s will decrease by one, all other offsets remain the same, and $\Delta opt_{i+1} = 1$.

Case 1. A request at s such that $v_s < D$.

In this case, we have $\Delta opt_{i+1} = 1$, and v_s increases by one. If $s \le k$, then $\mathbf{E}[\Delta C_{mig}] \le [1 - v_s/(2D)]$. It can be verified that $\mathbf{E}[\Delta M] \le 1/2$, $\Delta\Phi \le 1/2 + (v_s + 1)/(2D)$, and L.H.S. (5.1)$\le 2 + 1/(2D)$ = right-hand side (R.H.S.) (5.1). If $s > k$, then the movement cost is zero. We also have $\mathbf{E}[\Delta C_{mig}] \le 1$, $\Delta\Phi = 1/2 + (v_s + 1)/(2D) \le 1 + 1/(2D)$; inequality (5.1) also holds.

Case 2. A request at s such that $v_s = D$.

We have $\Delta opt_{i+1} = 0$. For each $j \in S - \{s\}$ such that $v_j > 0$, v_j decreases by 1. Each such v_j contributes $-[1/2 + v_j/(2D)]$ to $\Delta\Phi$ and no more than $1/2$ to $\mathbf{E}[\Delta M]$. We have $\mathbf{E}[\Delta Cost] = (1 - p_s)$, where p_s is the probability mass at s, and

$$1 - p_s = \sum_{j \ne s} p_j \le \sum_{j \ne s, v_j > 0} \frac{v_j}{2D}.$$

Hence L.H.S. (5.1) ≤ 0. ∎

6. Replication. We give upper and lower bounds on the performance of randomized on-line algorithms for the replication problem.

6.1. Randomized on-line algorithms. Let $e_D = (1 + 1/D)^D$ and $\beta_D = e_D/(e_D - 1)$. So $\beta_D \longrightarrow e/(e-1) \approx 1.58$ as $D \longrightarrow \infty$. We describe randomized algorithms that are β_D-competitive against an oblivious adversary on the uniform network and trees. First, consider a single edge (r, b) of unit length. Initially, only r contains a copy of F. Suppose algorithm A is α-competitive, and it replicates F to node b with probability p_i after the ith request at b, where $\sum_i p_i = 1$. The p_i's and α must satisfy, for each $k \in \mathcal{Z}_0^+$,

$$\mathbf{E}[C_A(\sigma) \mid k] \leq \begin{cases} \alpha \cdot k, & k \leq D, \\ \alpha \cdot D, & k > D, \end{cases}$$

$$\text{where } \mathbf{E}[C_A(\sigma) \mid k] = \sum_{i=1}^{k} p_i \cdot (D + i) + \left(1 - \sum_{i=1}^{k} p_i\right) \cdot k$$

is the expected cost incurred by A when σ contains k requests at b. The optimal off-line strategy is to replicate a copy of F to b before the first request arrives if $k \geq D$ and does not replicate otherwise. Algorithm A incurs a cost of $(D + i)$ if it replicates right after serving the ith request, $i \leq k$; otherwise it incurs a cost of k. An optimal randomized algorithm is given by a set of p_i values that satisfy the above inequalities for all k such that α is minimized. We note that the conditions above are identical to those for the on-line block snoopy caching problem on two caches in [15]. Karlin et al. [15] showed that the optimal α value is β_D. This is achieved when $p_i = [(D+1)/D]^{i-1}/[D(e_D - 1)]$, $1 \leq i \leq D$, and other p_i's are zero. The above single edge algorithm can be applied to a uniform network by replicating F to each node v after the ith request at v, with a probability of p_i—another example of factoring.

THEOREM 6.1. *There exists a randomized algorithm that is strongly β_D-competitive against an oblivious adversary for replication on a uniform network.*

We can extend the single edge algorithm to a tree, T, rooted at r, the node that contains F initially. The algorithm responds only to requests at nodes not in the current residence set.

ALGORITHM TREE. Keep a counter c_i on each node $i \neq r$. Initially, all $c_i = 0$. Suppose a request arrives at a node x, and w is the node nearest to x that contains a copy of F. After serving the request, the counters for all the nodes along the path from w to x are increased by one. Let the nodes along the path be $w = i_0, i_1, \ldots, i_q = x$, $q \geq 1$. Perform the following procedure each time after a request has been served:

(1) Let $p_{c_{i_0}} = 1$.
(2) **For** $j = 1, \ldots, q$, **Do**
  (2.1) With probability $p_{c_{i_j}}/p_{c_{i_{j-1}}}$, replicate to node i_j from node i_{j-1}.
  (2.2) **If** F is not replicated to i_j, **STOP**.

THEOREM 6.2. *Algorithm* **Tree** *is strongly β_D-competitive against an oblivious adversary for replication on a tree network.*

Proof. Without loss of generality, we assume that a *connected* R is always maintained by any solution. Let $x \neq r$ be any node and y its parent. Before x obtains a copy of F, a cost equal to the weight of (x, y) is incurred on the edge for each request at a node in the subtree rooted at x. (This follows because R is connected and the unique path from x to r passes through (x, y).) By our single edge algorithm, the algorithm **Tree** is β_D-competitive if, for each node $x \neq r$, a copy of F is replicated to x after the ith request at the subtree rooted at x, with a probability p_i. It can be

shown by induction on the requests that before x acquires a copy of F, the counter on x records the number of requests that have arrived at the subtree rooted at x, and these counters form a nonincreasing sequence on any path moving away from r. Hence, the values $p_{c_{i_j}}/p_{c_{i_{j-1}}} = (1+1/D)^{c_{i_j}-c_{i_{j-1}}} \leq 1$, $j = 2, \ldots, q$, are defined probability values. It is simple to verify that each time a node at the subtree rooted at x receives a request, a copy of F is replicated to x with probability p_{c_x}, where c_x is x's new counter value. The theorem follows. ∎

6.2. Lower bound. We show that no randomized algorithm can be better than 2-competitive against an adaptive on-line adversary. We use n to denote the number of nodes in G.

THEOREM 6.3. *Let ϵ be any positive function of n and D, taking values between 0 and 1. No on-line algorithm is better than $(2-\epsilon(D,n))$-competitive for replication against an adaptive on-line adversary.*

Proof. Let node a have the initial copy of F, and let (a,b) be any edge in G. Let A be any on-line algorithm which replicates to b after the jth request at b with probability p_j, $j \in \mathcal{Z}^+$. The adversary issues requests at b until A replicates or until N_ε requests have been issued, whichever first happens. Algorithm A incurs an expected cost of

$$\mathbf{E}[C_A(\sigma)] = \sum_{j=1}^{N_\varepsilon} p_j \cdot (j+D) + \sum_{j=N_\varepsilon+1}^{\infty} p_j \cdot N_\varepsilon. \tag{6.1}$$

We choose different N_ε's for two different cases.

Suppose $\sum_{j=1}^{\infty} p_j \cdot j > D$. Let N_ε be the minimum positive integer that is greater than D and such that $\sum_{j=1}^{N_\varepsilon} p_j \cdot j \geq D$. Given D, parameter N_ε is a finite and unique constant. The adversary replicates to b before the first request arrives, incurring a cost of D. From (6.1), we have $\mathbf{E}[C_A(\sigma)] \geq D + \sum_{j=1}^{N_\varepsilon} p_j \cdot j \geq 2D$, giving a ratio of at least 2.

Suppose $\sum_{j=1}^{\infty} p_j \cdot j \leq D$. Let $\varepsilon = \epsilon(n,D)$. The adversary does not replicate and incurs an expected cost of

$$\sum_{j=1}^{N_\varepsilon} p_j \cdot j + \sum_{j=N_\varepsilon+1}^{\infty} p_j \cdot N_\varepsilon. \tag{6.2}$$

From (6.1), we have

$$\mathbf{E}[C_A(\sigma)] = \sum_{j=1}^{N_\varepsilon} p_j \cdot j + D \cdot \sum_{j=1}^{N_\varepsilon} p_j + \sum_{j=N_\varepsilon+1}^{\infty} p_j \cdot N_\varepsilon \tag{6.3a}$$

$$\geq \left(1 + \sum_{j=1}^{N_\varepsilon} p_j\right) \cdot \sum_{j=1}^{N_\varepsilon} p_j \cdot j + \sum_{j=N_\varepsilon+1}^{\infty} p_j \cdot N_\varepsilon. \tag{6.3b}$$

Given the p_j's, one can choose N_ε so that $\sum_{j=1}^{N_\varepsilon} p_j$ is arbitrarily close to 1. Since the series $\sum_{j=1}^{\infty} j \cdot p_j$ is bounded, one can also choose N_ε so that $\sum_{j=N_\varepsilon+1}^{\infty} p_j \cdot N_\varepsilon$ is arbitrarily close to zero. Thus, by comparing (6.2) and (6.3b), we see that given any ε, one can choose a finite N_ε so that the ratio is as close to $(2-\varepsilon)$ as desired. ∎

7. Off-line replication and file allocation. We show that the off-line replication problem is NP-hard, and the off-line file allocation problem on the uniform network can be solved in polynomial time.

7.1. The off-line replication problem. Awerbuch, Bartel, and Fiat find interesting relationships between the on-line Steiner tree problem [14, 23] and on-line FAP. We show that the (off-line) replication problem is NP-hard by using a straightforward reduction from the Steiner tree problem [12, 16] (see section 2 for the definition). The proof involves creating an instance for the replication problem in which $(D+1)$ requests are issued at each of the terminal nodes for the Steiner tree problem instance, forcing the optimal algorithm to replicate to these nodes.

THEOREM 7.1. *The replication problem is NP-hard, even if G is bipartite and unweighted or if G is planar.*

7.2. Off-line solution for file allocation on a uniform network. We show that the off-line file allocation problem on a uniform network can be solved in polynomial time by reducing it to a min-cost max-flow problem. A similar reduction was obtained by Chrobak et al. [8] for the k-server problem. We convert an instance of the FAP on a uniform network on nodes $1, \ldots, n$, to a min-cost max-flow problem on an acyclic layered network, N, with $O(n \cdot |\sigma|)$ nodes and $O(n^2|\sigma|)$ arcs. Initially node 1 has a copy of F. An integral maximum flow in N defines a dynamic allocation of F in the uniform network. The arc costs in N are chosen so that the min-cost max-flow in N incurs a cost that differs from the minimum cost for FAP on the uniform network by a constant. Network N is constructed as follows.

Nodes. Network N has $(|\sigma|+1)$ layers of nodes, $(2n-1)$ nodes in each layer, a source node s, and a sink node t. Layer k, $0 \le k \le |\sigma|$, has nodes $\{v_1^k, \ldots, v_n^k\}$ and $\{u_1^k, \ldots, u_{(n-1)}^k\}$. Each node allows a maximum flow of one unit into and out of it. The v_j^k nodes correspond to the nodes in the uniform network. Layer k of N corresponds to the state of the uniform network after σ_k has been served.

Arcs. There is an arc going from each layer k node to each layer $(k+1)$ node, $0 \le k \le (|\sigma|-1)$; there is an arc from each layer $|\sigma|$ node to t, arc (s, v_1^0), and arcs $(s, u_j^0), 1 \le j \le (n-1)$. All the arcs have unit capacity.

A flow. A maximum flow in N has a value of n. Given integer arc costs, there is a min-cost max-flow solution with only an integral flow of either 0 or 1 in each arc. A flow of 1 into v_j^k represents the presence of a copy of F at node j just before request k arrives. If the flow comes from $v_i^{(k-1)}$, it represents a copy of F being moved from node i to node j after serving the $(k-1)$st request; if the flow comes from $u_i^{(k-1)}$, it represents a replication to node j. A flow from $v_j^{(k-1)}$ to u_w^k represents that the copy of F at j is dropped after $\sigma_{(k-1)}$ is served. Thus an integral flow in N defines a strategy for relocating copies of F. Since there are $(n-1)$ u nodes in each layer, an integral max-flow must include a flow of 1 unit into at least one of the v nodes in each layer. This corresponds to the requirement that there is always at least a copy of F in the uniform network.

Edge costs. Edge costs are chosen so that the optimal flow has cost equal to the optimal off-line cost for FAP minus the number of read requests, J, in σ. Arcs with one end point at s or t have zero costs. Let (a, b) be any other arc, going between layer k and $(k+1)$. Its cost is equal to the sum of its associated *movement* and *service* costs. Suppose b is $v_i^{(k+1)}$. Then (a, b)'s associated movement cost is D unless a is v_i^k. If $\sigma_{(k+1)}$ is a write at some node other than node i, the service cost is 1; if $\sigma_{(k+1)}$ is a read at node i, the service cost is -1. The costs for all other cases are zero. The movement and service costs account for the cost for replication and serving requests, except a node is charged -1 when a read request arrives and it has a copy of F. If we add J to the cost of the optimal flow, thus charging each read request 1 in advance,

the sum is equal to the cost of an optimal dynamic allocation of F.

Using the algorithm in [21] for solving the min-cost max-flow problem on acyclic networks, FAP on a uniform network can be solved in polynomial time.

THEOREM 7.2. *An optimal (off-line) file allocation on a uniform network can be found in* $O(n^3 \cdot |\sigma| \cdot (1 + \log_n |\sigma|))$ *time.*

8. Open problems. Interesting open problems include finding a strongly competitive randomized algorithm for FAP on a uniform network. Awerbuch, Bartel, and Fiat [2] conjecture that if there exists a c_n-competitive algorithm for the on-line Steiner tree problem [14, 23], then there exists a $O(c_n)$-competitive deterministic algorithm for FAP. This conjecture is still open. For the migration problem, there is a gap between the best known bounds [4, 10].

Appendix: Complete proof for (B) in Lemma 3.7. The following is the complete proof for condition (B) in Lemma 3.7. It will be shown that

$$S_i(x, y) \le S_i(y, z) \tag{8.1}$$

holds for $i = (t + 1)$ after request $(t + 1)$ has arrived, by considering the change in offset values using Table 3.2 for all possible locations of the root node r, request node w, and offset vectors. In each case, the required inequality follows from the property that all offsets satisfy $0 \le k \le l$ and the assumption that (8.1) holds initially for $i = t$. We will be referring to the conditions (C.1) to (C.4) and (D) that are implied by (8.1) for $i = t$. We use L.H.S. and R.H.S. to denote the left- and right-hand sides of the inequality under consideration, respectively.

Suppose the request is a **READ** request:

Case 1: $r \in T_x$ and $w \in T_x$.
For (x, y): $\omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = (0, \min(k_{xy} + 1, l_{xy}), l_{xy})$.
For (y, z): $\omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz} + 1, l_{yz}), l_{yz})$.
$S_{t+1}(x, y) \le S_{t+1}(y, z)$ follows from $S_t(x, y) \le S_t(y, z)$ or (C.1).

Case 2: $r \in T_y$ and $w \in T_y$.
For (x, y): $\omega_t = (k_{xy}, 0, l_{xy})$ and $\omega_{t+1} = (\min(k_{xy} + 1, l_{xy}), 0, l_{xy})$.
For (y, z): $\omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz} + 1, l_{yz}), l_{yz})$.
$S_{t+1}(x, y) \le S_{t+1}(y, z) \Leftrightarrow l_{xy} - \min(k_{xy} + 1, l_{xy}) \le l_{yz}$ follows from (D).

Case 3: $r \in T_x$ and $w \in T_y$.
For (x, y): $\omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = \begin{cases} (0, k_{xy} - 1, l_{xy} - 1) & \text{if } k_{xy} \ge 1, \\ (\min(1, l_{xy}), 0, l_{xy}) & \text{if } k_{xy} = 0. \end{cases}$
For (y, z): $\omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz} + 1, l_{yz}), l_{yz})$.
$k_{xy} \ge 1$: $S_{t+1}(x, y) \le S_{t+1}(y, z) \Leftrightarrow l_{xy} - 1 \le l_{yz}$ follows from (C.1).
$k_{xy} = 0$: $S_{t+1}(x, y) \le S_{t+1}(y, z) \Leftrightarrow l_{xy} - \min(1, l_{xy}) \le l_{yz}$ follows from (C.1).

Case 4: $r \in T_x$ and $w \in T_z$.
For (x, y): $\omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = \begin{cases} (0, k_{xy} - 1, l_{xy} - 1) & \text{if } k_{xy} \ge 1, \\ (\min(1, l_{xy}), 0, l_{xy}) & \text{if } k_{xy} = 0. \end{cases}$
For (z, y): $\omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = \begin{cases} (0, k_{yz} - 1, l_{yz} - 1) & \text{if } k_{yz} \ge 1, \\ (\min(1, l_{yz}), 0, l_{yz}) & \text{if } k_{yz} = 0. \end{cases}$
$k_{xy}, k_{yz} \ge 1$: $S_{t+1}(x, y) \le S_{t+1}(y, z) \Leftrightarrow l_{xy} \le l_{yz}$ follows from (C.1).
$k_{xy} = k_{yz} = 0$: By (C.4), $l_{xy} = l_{yz}$; hence $S_{t+1}(x, y) = S_{t+1}(y, z)$ holds in this case.
$k_{xy} \ge 1, k_{yz} = 0$: By (C.3), this case cannot happen.
$k_{xy} = 0, k_{yz} \ge 1$: $S_{t+1}(x, y) \le S_{t+1}(y, z) \Leftrightarrow l_{xy} - \min(1, l_{xy}) \le l_{yz}$ follows from (C.1).

Case 5: $r \in T_y$ and $w \in T_z$.
For (x, y): $\omega_t = (k_{xy}, 0, l_{xy})$ and $\omega_{t+1} = (\min(k_{xy} + 1, l_{xy}), 0, l_{xy})$.

For (z,y): $\omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = \begin{cases} (0, k_{yz}-1, l_{yz}-1) & \text{if } k_{yz} \geq 1, \\ (\min(1, l_{yz}), 0, l_{yz}) & \text{if } k_{yz} = 0. \end{cases}$

$k_{yz} \geq 1$: $S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow l_{xy} - \min(k_{xy}+1, l_{xy}) \leq l_{yz}$ follows from (D).

$k_{yz} = 0$:

We need to show that $S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow l_{xy} - \min(k_{xy}+1, l_{xy}) \leq l_{yz} - \min(1, l_{yz})$ holds, given $S_t(x,y) \leq S_t(y,z) \Leftrightarrow l_{xy} - k_{xy} \leq l_{yz}$ holds initially. The inequality holds because $l_{xy} - l_{xy} = 0 = l_{yz} - l_{yz} \leq$ R.H.S., and $l_{xy} - k_{xy} - 1 \leq l_{yz} - 1 \leq$ R.H.S.

Suppose the request is a **WRITE** request:

Case 1: $r \in T_x$ and $w \in T_z$.

For (x,y): $\omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = \begin{cases} (0, k_{xy}-1, l_{xy}) & \text{if } k_{xy} \geq 1, \\ (1, 0, \min(l_{xy}+1, D)) & \text{if } k_{xy} = 0. \end{cases}$

For (y,z): $\omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = \begin{cases} (0, k_{yz}-1, l_{yz}) & \text{if } k_{yz} \geq 1, \\ (1, 0, \min(l_{yz}+1, D)) & \text{if } k_{yz} = 0. \end{cases}$

$k_{xy}, k_{yz} \geq 1 : S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow l_{xy} \leq l_{yz}$ follows from (C.1).

$k_{xy} \geq 1, k_{yz} = 0$: By (C.3), this case cannot happen.

$k_{xy} = k_{yz} = 0$: By (C.4), $l_{xy} = l_{yz}$; hence $S_{t+1}(x,y) = S_{t+1}(y,z)$ holds in this case.

$k_{xy} = 0, k_{yz} \geq 1$ $S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow \min(l_{xy}+1, D) - 1 \leq l_{yz}$ follows from $S_t(x,y) \leq S_t(y,z) \Leftrightarrow l_{xy} \leq l_{yz}$.

Case 2: $r \in T_y$ and $w \in T_z$.

For (x,y): $\omega_t = (k_{xy}, 0, l_{xy})$ and $\omega_{t+1} = (\min(k_{xy}+1, D), 0, \min(l_{xy}+1, D))$.

For (y,z): $\omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = \begin{cases} (0, k_{yz}-1, l_{yz}) & \text{if } k_{yz} \geq 1, \\ (1, 0, \min(l_{yz}+1, D)) & \text{if } k_{yz} = 0. \end{cases}$

$k_{yz} \geq 1$: We need to show that

$$S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow \min(l_{xy}+1, D) - \min(k_{xy}+1, D) \leq l_{yz} \text{ holds, given that} \tag{8.2}$$

$$S_t(x,y) \leq S_t(y,z) \Leftrightarrow l_{xy} - k_{xy} \leq l_{yz} \text{ holds initially.}$$

It can be proved as follows. If $k_{xy} = D$, then L.H.S. $= 0 \leq$ R.H.S.

If $k_{xy} < D$, then L.H.S. $\leq l_{xy} + 1 - (k_{xy}+1) \leq l_{yz} =$ R.H.S. Hence (8.2) holds.

$k_{yz} = 0$: We need to show that

$$S_{t+1}(x,y) \leq S_{t+1}(y,z) \Leftrightarrow \min(l_{xy}+1, D) - \min(k_{xy}+1, D) \leq \min(l_{yz}+1, D) - 1 \text{ holds, given that} \tag{8.3}$$

$$S_t(x,y) \leq S_t(y,z) \Leftrightarrow l_{xy} - k_{xy} \leq l_{yz} \text{ holds initially.}$$

If $l_{yz} < D$, (8.3) follows from (8.2). If $k_{xy} = D$, then L.H.S. $= 0 \leq$ R.H.S. Suppose $l_{yz} \leq (D-1)$ and $(k_{xy}+1) \leq D$. In this case L.H.S. $\leq l_{xy} - k_{xy} \leq l_{yz} =$ R.H.S. Hence (8.3) holds.

Case 3: $r \in T_y$ and $w \in T_y$.

For (x,y): $\omega_t = (k_{xy}, 0, l_{xy})$ and $\omega_{t+1} = (\min(k_{xy}+1, D), 0, \min(l_{xy}+1, D))$.

For (y,z): $\omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz}+1, D), \min(l_{yz}+1, D))$.

We need to show that $S_{t+1}(x,y) \leq S_{t+1}(y,z)$

$\Leftrightarrow$ $\min(l_{xy}+1, D) - \min(k_{xy}+1, D) \leq \min(l_{yz}+1, D)$ holds,

given that $S_t(x,y) \le S_t(y,z) \Leftrightarrow l_{xy} - k_{xy} \le l_{yz}$ holds initially which follows from (8.3).

Case 4: $r \in T_x$ and $w \in T_y$.

For (x,y): $\omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = \begin{cases} (0, k_{xy}-1, l_{xy}) & \text{if } k_{xy} \ge 1, \\ (1, 0, \min(l_{xy}+1, D)) & \text{if } k_{xy} = 0. \end{cases}$

For (y,z): $\omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz}+1, D), \min(l_{yz}+1, D))$.

$k_{xy} \ge 1$: $S_{t+1}(x,y) \le S_{t+1}(y,z) \Leftrightarrow l_{xy} \le \min(l_{yz}+1, D)$ follows from $S_t(x,y) \le S_t(y,z) \Leftrightarrow l_{xy} \le l_{yz}$.

$k_{xy} = 0$:

$S_{t+1}(x,y) \le S_{t+1}(y,z) \Leftrightarrow \min(l_{xy}+1, D)-1 \le \min(l_{yz}+1, D)$ follows from $S_t(x,y) \le S_t(y,z) \Leftrightarrow l_{xy} \le l_{yz}$.

Case 5: $r \in T_x$ and $w \in T_x$.

For (x,y): $\omega_t = (0, k_{xy}, l_{xy})$ and $\omega_{t+1} = (0, \min(k_{xy}+1, D), \min(l_{xy}+1, D))$.

For (y,z): $\omega_t = (0, k_{yz}, l_{yz})$ and $\omega_{t+1} = (0, \min(k_{yz}+1, D), \min(l_{yz}+1, D))$.

$S_{t+1}(x,y) \le S_{t+1}(y,z) \Leftrightarrow \min(l_{xy}+1, D) \le \min(l_{yz}+1, D)$ follows from $S_t(x,y) \le S_t(y,z) \Leftrightarrow l_{xy} \le l_{yz}$.

Thus we have shown that (B) holds after σ_{t+1} has arrived.

REFERENCES

[1] S. Albers and H. Koga, *New on-line algorithms for the page replication problem*, in Proc. Fourth Scandinavian Workshop on Algorithmic Theory, Aarhus, Denmark, 1994, Lecture Notes in Comput. Sci. 824, Springer-Verlag, New York, pp. 25–36.

[2] B. Awerbuch, Y. Bartal, and A. Fiat, *Competitive distributed file allocation*, in Proc. 25th ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 164–173.

[3] B. Gavish and O. R. L. Sheng, *Dynamic file migration in distributed computer systems*, Comm. ACM, 33 (1990), pp. 177–189.

[4] Y. Bartal, M. Charikar, and P. Indyk, *On page migration and other relaxed task systems*, in Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, 1997, pp. 43–52.

[5] Y. Bartal, A. Fiat, and Y. Rabani, *Competitive algorithms for distributed data management*, in Proc. 24th Annual ACM Symposium on the Theory of Computing, Victoria, BC, 1992, pp. 39–50.

[6] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson, *On the power of randomization in online algorithms*, Algorithmica, 11 (1994), pp. 2–14.

[7] D. L. Black and D. D. Sleator, *Competitive Algorithms for Replication and Migration Problems*, Tech. Report CMU-CS-89-201, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1989.

[8] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan, *New results on server problems*, SIAM J. Discrete Math., 4 (1991), pp. 172–181.

[9] M. Chrobak and L. L. Larmore, *The server problem and on-line games*, in Proc. of the DIMACS Workshop on On-Line Algorithms, AMS, Providence, RI, 1991, pp. 11–64.

[10] M. Chrobak, L. L. Larmore, N. Reingold, and J. Westbrook, *Page migration algorithms using work functions*, in Proc. 4th International Symposium on Algorithms and Computation, Hong Kong, 1993, Lecture Notes in Comput. Sci. 762, Springer-Verlag, New York, pp. 406–415.

[11] D. Dowdy and D. Foster, *Comparative models of the file assignment problem*, Comput. Surveys, 14 (1982), pp. 287–313.

[12] M. R. Garey and D. S. Johnson, *The rectilinear Steiner tree problem is NP-complete*, SIAM J. Appl. Math., 32 (1977), pp. 826–834.

[13] F. K. Hwang and D. Richards, *Steiner tree problems*, Networks, 22 (1992), pp. 55–89.

[14] M. Imase and B. M. Waxman, *Dynamic Steiner tree problem*, SIAM J. Discrete Math., 4 (1991), pp. 369–384.

[15] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, *Competitive randomized algorithms for non-uniform problems*, in Proc. 1st ACM-SIAM Symposium on Discrete Algorithms, 1990, SIAM, Philadelphia, PA, pp. 301–309.

[16] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.
[17] H. Koga, *Randomized on-line algorithms for the page replication problem*, in Proc. 4th International Symposium on Algorithms and Computation, Hong Kong, 1993, Lecture Notes in Comput. Sci. 762, Springer-Verlag, New York, pp. 436–445.
[18] C. Lund and N. Reingold, *Linear programs for randomized on-line algorithms*, in Proc. 5th ACM-SIAM Symposium on Discrete Algorithms, 1994, SIAM, Philadelphia, PA, pp. 382–391.
[19] C. Lund, N. Reingold, J. Westbrook, and D. Yan, *On-line distributed data management*, in Proc. 2nd Annual European Symposium on Algorithms, Utrecht, The Netherlands, Lecture Notes in Comput. Sci. 855, 1994, Springer-Verlag, New York, pp. 202–214.
[20] N. Reingold, J. Westbrook, and D. D. Sleator, *Randomized competitive algorithms for the list update problem*, Algorithmica, 11 (1994), pp. 15–32.
[21] R. E. Tarjan, *Data Structures and Network Algorithms*, CBMS-NSF Regional Conf. Ser. in Appl. Math. 44, SIAM, Philadelphia, PA, 1983.
[22] J. Westbrook, *Randomized algorithms for multiprocessor page migration*, SIAM J. Comput., 23 (1994), pp. 951–965.
[23] J. Westbrook and D. C. K. Yan, *The performance of greedy algorithms for the on-line Steiner tree and related problems*, Math. Systems Theory, 28 (1995), pp. 451–468.
[24] P. Winter, *Steiner problem in networks: A survey*, Networks, 17 (1987), pp. 129–167.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 1112–1135

EFFICIENT DATABASE UPDATES WITH INDEPENDENT SCHEMES*

RICCARDO TORLONE† AND PAOLO ATZENI†

Abstract. The weak instance model is a framework for considering the relations in a database as a whole, regardless of the way attributes are grouped in the individual relations. Queries and updates can be performed for any set of attributes. The management of updates is based on a lattice structure on the set of legal states, and inconsistencies and ambiguities can arise.

In the general case, the test for consistency and determinism may involve the whole database. In this paper it is shown how, for the highly significant class of independent schemes, updates can be handled efficiently, considering only the relevant portion of the database.

Key words. relational database, weak instance model, lattice on database states, update operations, chase procedure, optimization

AMS subject classifications. 68P15, 68Q25, 06B99

PII. S009753979325799X

1. Introduction.

In a relational database, the universe of discourse is represented by means of a set of relations. The *weak instance approach* [6, 18, 22, 23, 24, 31] provides a framework to consider a database as a whole, regardless of the way attributes appear in the various relation schemes. The information content of a database state is considered to be embodied in the *representative instance*, a sort of relation with variables, obtained by extending all relations to the global set of attributes (called the *universe*) and then by *chasing* [21] their union. Query answering is performed by first computing the *total projection* [26] of the representative instance on the set of attributes involved in the query and then executing whatever further operations are needed. Any subset of the universe can in principle be queried.

EXAMPLE 1. *Consider a database scheme with relations* $R_1(EDP)$, $R_2(DMP)$ *and the functional dependencies* $E \to D$, $D \to M$ *as constraints. Figures* 1, 2, *and* 3, *show a consistent database state on this scheme, the corresponding representative instance, and a total projection, respectively.*

Following a new interest on the theory of database updates [1], we proposed a formal approach to updates in the weak instance model [9]; in the same way as for the management of queries, which allows the retrieval of tuples over any subset of the universe, insertions and deletions over any subset of the universe are allowed. Problems of consistency and determinism arise and have been completely characterized.

EXAMPLE 2. *If we want to insert in the state in Figure* 1 *a tuple defined on the attributes* EM, *with values* Jim *for* E *and* White *for* M, *we can consistently add the tuple* ⟨MS, White, C⟩ *to the second relation. Because of the dependency* $D \to M$, *the chase would combine the tuple* ⟨Jim, MS, C⟩ *already in* r_1, *with this tuple, generating a tuple in the representative instance with values* Jim *for* E *and* White *for* M. *Conversely, if we want to insert into the same state a tuple defined on* EPM,

*Received by the editors November 1, 1993; accepted for publication (in revised form) July 8, 1997; published electronically February 19, 1999. A preliminary version of this paper appeared in *Proc. ACM SIGMOD International Conf. on Management of Data*, Atlantic City, NJ, May 1990. This work was partially supported by MURST and by Consiglio Nazionale delle Ricerche.

http://www.siam.org/journals/sicomp/28-3/25799.html

†Dipartimento di Informatica e Automazione, Università di Roma Tre, Via della Vasca Navale 79, 00146 Roma, Italy (torlone@dia.uniroma3.it, atzeni@dia.uniroma3.it).

r_1:

Employee	Dept.	Project
John	CS	A
John	CS	B
Bob	EE	B
Jim	MS	C

r_2:

Dept.	Manager	Project
CS	Smith	A
IE	White	B
EE	Jones	B
CS	Smith	B

FIG. 1. *A database state.*

Employee	Dept.	Project	Manager
John	CS	A	Smith
John	CS	B	Smith
Bob	EE	B	Jones
Jim	MS	C	v_1
v_2	CS	A	Smith
v_3	IE	B	White
v_4	EE	B	Jones

FIG. 2. *A representative instance.*

Employee	Manager
John	Smith
Bob	Jones

FIG. 3. *A total projection of the representative instance.*

with values Dan *for* E, C *for* P, *and* Moore *for* M, *we obtain a potential result only if we add one tuple to* r_1 *and one tuple to* r_2, *with the given values for* EPM *and with the same value for* D *whichever it is (provided that the constraints are not violated). In this case, some further piece of information has to be added, and there are several possible choices—this is a case of nondeterminism. Finally, an inconsistency arises if we try to insert a tuple on* EM *with* John *for* E *and* White *for* M, *since the functional dependencies imply that the only manager of* John *is* Smith.

In the general case, the characterizations for consistency and determinism of insertions require the construction of the representative instance of the state, by means of the application of the chase procedure to a set of data that involves the whole database [9]. However, updating a state often involves only a small part of the database. The case is therefore similar to that of query answering, where the crucial step is the computation of the representative instance, which has to be completely constructed in the general case. To solve this problem, various classes of schemes were introduced, beginning with *independent schemes* [17, 19, 25, 26], where queries can be answered by means of simple relational expressions, optimizable and independent of the actual database state [5, 19, 27, 28]. In this paper we show that a similar approach can be followed for updating as well: we study updates to relational databases through weak instances and show that they can be implemented efficiently.

The paper is organized as follows. In section 2 we briefly review the needed background. In section 3 we review definitions and characterizations of updates in the weak instance model. In section 4 we consider independent schemes and characterize consistency and determinism with this class of schemes. On the basis of these results, in section 5 we present practical and efficient algorithms for update operations, and in section 6 we show that some step of these algorithms can be simplified under certain

further assumptions. In section 7, we discuss modifications, another important class of database update operations, and finally, in section 8, we conclude by summarizing our contribution.

2. Background definitions and notation.

2.1. Relations, databases, and tableaux. The *universe* U is a finite set of symbols $\{A_1A_2 \dots A_m\}$, called *attributes*. As usual, we use the same notation A to indicate both the single attribute A and the singleton set $\{A\}$. Also, we indicate the union of attributes (or sets thereof) by means of the juxtaposition of their names. A *relation scheme* is an object $R(X)$, where R is the *name* of the relation scheme and X is a subset of U. A *database scheme* is a collection of relation schemes $\mathbf{R} = \{R_1(X_1), \dots, R_n(X_n)\}$, with distinct relation names (which therefore can be used to identify the relation schemes) and such that the union of the X_i's is the universe U.

The *domain* D is the disjoint union of two countably infinite sets, the set of *constants* and the set of *variables*. (For the sake of simplicity, we assume that all attributes have the same domain.) A *tuple* on a set of attributes X is a function t from X to D. If t is a tuple on X and if $Y \subseteq X$, then $t[Y]$ denotes the restriction of the mapping t to Y and is therefore a tuple on Y. A tuple t on a set of attribute X is *total* if it does not involve variables.

A *tableau* T is a set of tuples on the universe U. We say that a variable or a constant is *unique* in T if it appears only once. A *relation* of a relation scheme $R(X)$ is a finite set of total tuples on X. A (*database*) *state* of a database scheme $\mathbf{R}$ is a function $\mathbf{r}$ that maps each relation scheme $R_i(X_i) \in \mathbf{R}$ to a relation on $R_i(X_i)$. With a slight abuse of notation, given $\mathbf{R} = \{R_1, \dots, R_n\}$, we write $\mathbf{r} = \{r_1, \dots, r_n\}$.

Given a database state $\mathbf{r}$, the *state tableau* for $\mathbf{r}$ is a tableau (denoted by $T_{\mathbf{r}}$) formed by taking the union of all the relations in $\mathbf{r}$ extended to U by means of unique variables.

The *total projection* ($\pi^{\downarrow}$) is an operator on tableaux that produces relations, given a tableau T and a subset X of U, generating the set of total tuples on X that are restrictions of tuples in T: $\pi^{\downarrow}_X(T) = \{t[X] | t \in T \text{ and } t[X] \text{ is total}\}$. We will use two (orthogonal) generalizations of the total projection: (1) the *restricted total projection* of a tableau T on $X \subseteq U$ with respect to a set of constants C, denoted by $\pi^{\downarrow}_X[C](T)$, is the set of total tuples on X that do not contain constants in C: $\pi^{\downarrow}_X[C](T) = \{t[X] | t \in T, t[X] \text{ is total and, for each } A_i \in X, t[A_i] \notin C\}$; and (2) the *total projection of a tableau* T *on a database scheme* $\mathbf{R}$, denoted by $\pi^{\downarrow}_{\mathbf{R}}(T)$, is the state obtained by totally projecting T on the various relation schemes.

In this paper, we shall consider *relational expressions* whose only operators are select (σ), project (π), natural join ($\bowtie$), and union ($\cup$) and whose operands are relational schemes of a fixed database scheme. Given a database state $\mathbf{r}$ of a scheme $\mathbf{R}$ and a relational expression E with operands in $\mathbf{R}$, we denote by $E(\mathbf{r})$ the relation obtained by substituting $\mathbf{r}$ into the corresponding relation variables in E and evaluating E according to the usual definitions of relational operators [7, 20, 29]. The *target* of E is the set of attributes in U on which $E(\mathbf{r})$ is defined.

2.2. Constraints: Local satisfaction and global consistency. Usually associated with a database scheme is a set of *constraints*, that is, properties that are satisfied by the legal states. There are two notions of satisfaction: *local* satisfaction, defined on individual relations, and *global* satisfaction, or *consistency*, defined on the database state. We introduce the two concepts in turn. Various classes of constraints

have been defined in the literature [20, 29]; here, we limit our attention to functional dependencies.

Let $YZ \subseteq X \subseteq U$; a relation r on a scheme $R(X)$ *(locally) satisfies* the *functional dependency (FD)* $Y \rightarrow Z$ if, for every pair of tuples $t_1, t_2 \in r$ such that $t_1[Y] = t_2[Y]$, it is the case that $t_1[Z] = t_2[Z]$. Without loss of generality, in the following we will often assume that all FDs have the form $Y \rightarrow A$, where A is a single attribute.

Given a set of FDs, it usually happens that there are additional FDs implied by this set. The *closure* of a set of FDs F, denoted by F^+, is the set of dependencies that are logically implied by F, and the *closure* of a set of attributes X with respect to a set of FDs F, denoted by X_F^+ (or simply X^+ when F is understood from context), is the set of attributes $\{A \mid X \rightarrow A \in F^+\}$. A set of FDs F is said to be *nonredundant* if there is no $Y \rightarrow A \in F$ such that $(F - \{Y \rightarrow A\})^+ = F^+$. Let $R(X)$ be a relation scheme and F be a set of FDs defined on $R(X)$; a set of attributes $Y \subseteq X$ is a *superkey* of R if $Y \rightarrow X \in F^+$.

Let $\mathbf{R} = \{R_1(X_1), \ldots, R_n(X_n)\}$ be a database scheme; without loss of generality we associate with $\mathbf{R}$ a set of nonredundant FDs $F = \cup_{i=1}^n F_i$, where, for every $1 \leq i \leq n$, the FDs in F_i are defined on $R_i(X_i)$. Note that we assume the FDs to be embedded in relation schemes.

A state $\mathbf{r} = \{r_1, \ldots, \mathbf{r}_n\}$ *globally satisfies* [18] a set of FDs F if there is a relation w on the universe U (called a *weak instance* for $\mathbf{r}$ with respect to F) that (locally) satisfies F and contains the relations of $\mathbf{r}$ in its projections over the respective relation schemes: $\pi_{X_i}(w) \supseteq r_i$ for $1 \leq i \leq n$. A state that globally satisfies the set of dependencies associated with the database scheme is also said to be *(globally) consistent*.

2.3. The chase procedure. The chase [21] is a procedure that receives as input a tableau T and generates a tableau $CHASE_F(T)$ that, if possible, satisfies the given dependencies F. If only functional dependencies are considered, the process modifies values in the tableau, by equating variables and "promoting" variables to constants as follows. We use $\tau = \langle t_1, t_2, Y \rightarrow A \rangle$ to denote a *chase step*, with reference to a tableau T, where t_1 and t_2 are tuples in T and $Y \rightarrow A \in F$. We say that a chase step τ is *valid* if $t_1[Y] = t_2[Y]$ and that τ is *applied* to T, denoted $\tau(T)$, if it is valid and the A-value of t_1 and t_2 are modified as follows: if one of them is a constant and the other is a variable, then the variable is changed (is *promoted*) to the constant; otherwise the values are equated. If a chase step tries to identify two constants, then we say that the chase encounters a *contradiction*, and the process stops, generating a special tableau that we call the *inconsistent tableau* and denoted by T_∞. The tableau $CHASE_F(T)$ is obtained from T by applying all valid chase steps exhaustively to T. Lemma 1 states a property of the chase that will be used in the following discussion.

LEMMA 1 (see [5, Lemma 4.1]). *Let t be a tuple defined on U and $\mathbf{r} = \pi_{\mathbf{R}}^{\downarrow}(\{t\})$. Let $R_{i_0}(X_{i_0}) \in \mathbf{R}$ and t_{i_0} be the tuple in $CHASE_F(T_{\mathbf{r}})$ corresponding to $\pi_{X_{i_0}}(\{t\})$. Then $t_{i_0}[X_{i_0}^+] = t[X_{i_0}^+]$.*

2.4. Query answering in the weak instance model. The definition of global satisfaction is clearly not practical since, in general, there may be many weak instances (often infinitely many). However, the existence of a weak instance can be studied by means of the notion of representative instance, a tableau on the universe, U, of the attributes.

The *representative instance* for a state $\mathbf{r}$, denoted by $\mathrm{RI}_{\mathbf{r}}$, is the tableau obtained by chasing the state tableau $T_{\mathbf{r}}$ of $\mathbf{r}$ with respect to the dependencies associated with the database state.

The main property of the representative instance is that a database state is consistent if and only if the corresponding representative instance is not the inconsistent tableau [18]. Also, for every consistent state $\mathbf{r}$ and for every X, the X-total projection of the representative instance of $\mathbf{r}$ is equal to the set of tuples that appear in the projection on X of every weak instance of $\mathbf{r}$ [23].

The *weak instance approach* to query answering allows queries to be formulated on databases as if they were composed of just one relation over the universe U. For each query, being $X \subseteq U$ the set of attributes involved, the evaluation requires a first step that computes the relation over X implied by the current state: because of the result just mentioned on total projections [23], it follows that the X-total projection of the representative instance is the natural content of this relation.

We say that a tuple t over a set of attributes X *x-belongs* (in symbols, $t\widehat{\in}\mathbf{r}$) to a consistent state $\mathbf{r}$ of a database scheme $\mathbf{R}$ with a universe $U \supseteq X$ if t belongs to the X-total projection of the representative instance of $\mathbf{r}$.

Finally, the *completion* $\mathbf{r}^*$ of a consistent state $\mathbf{r}$ is the state obtained by projecting the representative instance of $\mathbf{r}$ on the scheme $\mathbf{R}$; that is, $\mathbf{r}^* = \pi^{\downarrow}_{\mathbf{R}}(\mathrm{RI}_{\mathbf{r}})$ [24]. A consistent state $\mathbf{r}$ is *complete* if it coincides with its completion; that is, if $\mathbf{r} = \pi^{\downarrow}_{\mathbf{R}}(\mathrm{RI}_{\mathbf{r}})$.

2.5. Tableau and relational expression containment. A *valuation function* v is a function from D to D that is the identity on constants. A valuation function v can be extended to tuples and tableaux as follows: (i) given a tuple t on a set of attributes X, $v(t)$ is a tuple on X such that $v(t)[A] = v(t[A])$ for every $A \in X$; (ii) given a tableau $T = \{t_1, \ldots, t_n\}$, $v(T) = \{v(t_1), \ldots, v(t_n)\}$.

Given two tableaux T_1, T_2, we say that T_1 is *contained* in T_2 (in symbols $T_1 \leq T_2$) if T_2 is the inconsistent tableau T_∞, or there is a valuation function ψ (called in this case *containment mapping*) defined on all the symbols appearing in T_1 such that $\psi(T_1) \subseteq T_2$.[1] If both $T_1 \leq T_2$ and $T_2 \leq T_1$, the two tableaux are *equivalent*. Note that, by definition, the inconsistent tableau properly contains every other tableau. We now recall some useful properties of tableaux and chase.

LEMMA 2 (see [9, Lemma 1]). *For every tableau T, T_1 and for every set F of FDs, the following statements hold:*

1. $T \leq \mathit{CHASE}_F(T)$;
2. *if* $T \leq T_1$, *then* $\mathit{CHASE}_F(T) \leq \mathit{CHASE}_F(T_1)$;
3. *if* $T \leq T_1$ *and* $T_1 = \mathit{CHASE}_F(T_1)$, *then* $\mathit{CHASE}_F(T) \leq T_1$.

2.6. Independent schemes. A scheme is *independent* [17, 25] if, for all its states, local satisfaction implies global satisfaction. Independent schemes are clearly important from the practical point of view because the global consistency of their states can be verified in a local manner, looking at the individual relations, without having to build and chase the state tableau. Graham and Yannakakis [17] derived an efficient test for independence (later improved in [19, 27]).

Independent schemes are also important in the weak instance approach to query answering, because they guarantee the efficient computation of total projection of the representative instance [5, 19]. Atzeni and Chan [5], Ito, Iwasaki, and Kasami [19], and Sagiv [28] showed that, for every independent scheme and for every subset X of its universe, there is a relational algebra expression E_X that computes the total projection of the representative instance for every state of the scheme. In the approach

[1] Note that tableau containment is defined in the opposite direction when it refers to containment of queries.

of Atzeni and Chan, E_X is a union of *simple chase join expressions* (scjes) [5, 4, 13], a restricted form of a project-join expression, that we recall next.

A preliminary concept is needed: a *derivation sequence* (*ds*) of some relation scheme $R_{i_0}(X_{i_0})$ is a finite sequence of FDs $\sigma = \langle Y_1 \to Z_1, \ldots, Y_m \to Z_m \rangle$ from F such that, for all $1 \leq j \leq m$: $Y_j \subseteq X_{i_0}Z_1 \cdots Z_{j-1}$ and $Z_j \cap X_{i_0}Z_1 \cdots Z_{j-1} = \emptyset$. We say that σ *covers* a set of attributes X if $X_{i_0}Z_1 \cdots Z_j \supseteq X$. Essentially, a ds of $R_{i_0}(X_{i_0})$ is a sequence of FDs used in computing (part of) the closure of X_{i_0}.

Given a ds $\sigma = \langle Y_1 \to Z_1, \ldots, Y_m \to Z_m \rangle$ covering a set of attributes X, the scjes *for* σ *over* X is the expression

$$\pi_X(R_{i_0} \bowtie \pi_{Y_1Z_1}(R_{i_1}) \bowtie \cdots \bowtie \pi_{Y_mZ_m}(R_{i_m})),$$

where $Y_j \to A_j$, for $1 \leq j \leq m$, is an FD in F_{i_j} and is therefore embedded in R_{i_j}. The subexpressions $R_{i_0}, \pi_{Y_1Z_1}(R_{i_1}), \ldots, \pi_{Y_mZ_m}(R_{i_m})$ are called the *components* of the scje.

Before closing this section, we mention an important property of independent schemes that will be often used in the sequel: each derived value in the chase of $T_{\mathbf{r}}$ is "uniquely" derived for an independent scheme.

LEMMA 3 (see [16, Lemma 4]). *Let* $\mathbf{R}$ *be an independent scheme. Then for any consistent state* $\mathbf{r}$*, for any* $R_j \in \mathbf{R}$*, and for any FD* $Y \to A \in F_j$*, it is the case that* $\pi^{\downarrow}_{YA}(CHASE_F(T_{\mathbf{r}})) = \pi_{YA}(r_j)$.

3. Updating in the weak instance model. In this section we briefly review our approach to updates in the weak instance model [9]. Similar to the approach to query answering, it allows updates to be formulated on every subset of the universe. As a preliminary tool, we introduce a partial order on states, which extends a known notion of equivalence of states [24]; then we discuss insertions, and finally deletions.

3.1. A lattice on states. A state $\mathbf{r}_1$ is *weaker* than a state $\mathbf{r}_2$ ($\mathbf{r}_1 \preceq \mathbf{r}_2$) if every weak instance of $\mathbf{r}_2$ is also a weak instance of $\mathbf{r}_1$. Two states $\mathbf{r}_1$, $\mathbf{r}_2$ are *equivalent* ($\mathbf{r}_1 \sim \mathbf{r}_2$) if both $\mathbf{r}_1 \preceq \mathbf{r}_2$ and $\mathbf{r}_2 \preceq \mathbf{r}_1$. The relation $\preceq$ is a partial order on the set of the complete states, since it is reflexive, antisymmetric, and transitive. Also, it is strongly related to (tableau) containment of representative instances, (set) containment of total projections, and relations, as stated in the next theorem.

THEOREM 1 (see [9, Theorem 1]). *Let* $\mathbf{r}_1 = \{r_{1,1}, \ldots, r_{1,n}\}$ *and* $\mathbf{r}_2 = \{r_{2,1}, \ldots, r_{2,n}\}$ *be two states. Properties* 1, 2, *and* 3 *below are equivalent; if the states are complete, then property* 4 *is also equivalent to the others.*

1. *The state* $\mathbf{r}_1$ *is weaker than the state* $\mathbf{r}_2$: $\mathbf{r}_1 \preceq \mathbf{r}_2$.
2. *The representative instance of* $\mathbf{r}_1$ *is contained in the representative instance of* $\mathbf{r}_2$: $\mathrm{RI}_{\mathbf{r}_1} \leq \mathrm{RI}_{\mathbf{r}_2}$.
3. *For every* $X \subseteq U$*, the* X*-total projection of* $\mathrm{RI}_{\mathbf{r}_1}$ *is a subset of the* X*-total projection of* $\mathrm{RI}_{\mathbf{r}_2}$: $\pi^{\downarrow}_X(\mathrm{RI}_{\mathbf{r}_1}) \subseteq \pi^{\downarrow}_X(\mathrm{RI}_{\mathbf{r}_2})$.
4. *The state* $\mathbf{r}_1$ *is a relationwise subset of the state* $\mathbf{r}_2$: *for every* $R_i \in \mathbf{R}$*, it is the case that* $r_{1,i} \subseteq r_{2,i}$.

By the equivalence of properties 1 and 3 of Theorem 1 above, it follows that two states are equivalent if and only if, for every X, their X-total projections are equal, that is if they have identical query answering behavior. Therefore, it makes sense to consider equivalence classes of states. To represent the various classes, we will use the set of complete states since it is known that each consistent state is equivalent to one and only one complete state [24, section 3].

In [9] we showed that the partial order $\preceq$ extended to the *complete inconsistent state* (a special state defined as the projection of the inconsistent tableau on the database scheme) induces a complete lattice [12] on the set of complete states, that is, every set of complete states has both a greatest lower bound (glb) and a least upper bound (lub).

3.2. Insertions. Let $\mathbf{R} = \{R_1(X_1), \ldots, R_n(X_n)\}$ be a database scheme, with $U = X_1X_2 \ldots X_n$. Given a state $\mathbf{r}$ of $\mathbf{R}$ and a tuple t over a set of attributes $X \subseteq U$, we consider the *insertion* of t into $\mathbf{r}$ defined through the following notion of result.

A state $\mathbf{r}_p$ is a *potential result* for the insertion of t into $\mathbf{r}$ if $\mathbf{r} \preceq \mathbf{r}_p$ and $t\widehat{\in}\mathbf{r}_p$. Various cases for an insertion of a tuple in a consistent state exist: the insertion of a tuple t over X in a state is *possible* if there is a consistent state $\mathbf{r}'$ such that $t\widehat{\in}\mathbf{r}'$, a possible insertion is *consistent* if it has a consistent potential result, and a possible and consistent insertion is *deterministic* if the glb of the potential results is a potential result. Note that the notion of determinism is defined only for possible and consistent insertions. When an insertion is deterministic, we consider the glb of the potential results as *the* result of the insertion. In other words, an insertion is possible if the dependencies allow us to generate t in the representative instance of a state $\mathbf{r}'$ possibly unrelated to the given state, it is consistent when the new tuple does not contradict the information content of the original state, and it is deterministic when the insertion can be univocally performed by adding only the information that is strictly needed.

In [9] we showed the following general characterizations for possibility, consistency, and determinism.

THEOREM 2 (see [9, Theorem 2]). *The insertion of t in a state is possible if and only if there is a relation scheme $R_i(X_i) \in \mathbf{R}$ such that F implies the FD $X_i \rightarrow X$.*

Let $\mathrm{RI}_{\mathbf{r}}$ be the representative instance of $\mathbf{r}$. The characterization of both consistency and determinism is based on the construction of a tableau obtained by adding to $\mathrm{RI}_{\mathbf{r}}$ a tuple t_e obtained by extending t to the universe U by means of unique variables. Let $T_{t,\mathbf{r}}$ be such a tableau.

THEOREM 3 (see [9, Theorem 3]). *Let the insertion of t in $\mathbf{r}$ be possible. It is consistent if and only if $CHASE_F(T_{t,\mathbf{r}}) \neq T_\infty$.*

DEFINITION 1 (state $\mathbf{r}_{+t}$). *The state $\mathbf{r}_{+t}$, or simply $\mathbf{r}_+$ when t is understood from context, is obtained from $\mathbf{r}$ and t by (totally) projecting $CHASE_F(T_{t,\mathbf{r}})$ on the database scheme: $\mathbf{r}_{+t} = \pi^{\downarrow}_{\mathbf{R}}(CHASE_F(T_{t,\mathbf{r}}))$.*

LEMMA 4 (see [9, Lemma 8]). *Let the insertion of t in $\mathbf{r}$ be possible and consistent. Then $\mathbf{r}_+$ is the glb of the potential results.*

THEOREM 4 (see [9, Theorem 4]). *Let the insertion of t in $\mathbf{r}$ be possible and consistent. It is deterministic if and only if $CHASE_F(T_{t,\mathbf{r}}) \equiv \mathrm{RI}_{\mathbf{r}_+}$.*

COROLLARY 1 (see [9, Corollary 1]). *Let the insertion of t in $\mathbf{r}$ be possible and consistent; it is deterministic if and only if $t\widehat{\in}\mathbf{r}_+$.*

Corollary 1 gives an effective characterization of determinism: given $\mathbf{r}$ and t, we can build $T_{t,\mathbf{r}}$, chase it with respect to the given constraints, then generate $\mathbf{r}_+$ and compute its representative instance $\mathrm{RI}_{\mathbf{r}_+}$, and finally check whether the total projection $\pi^{\downarrow}_X(\mathrm{RI}_{\mathbf{r}_+})$ contains t.

EXAMPLE 3. *Consider the first insertion in Example 2. The insertion is possible since for $R_1(X_1)$ we have $X_1 \rightarrow X \in F^+$. Then, following the definitions, we could build the tableau $T_{t,\mathbf{r}}$ and then chase it; the tableaux that we obtain are reported in Figure 4. It is possible to see that, if we project $CHASE_F(T_{t,\mathbf{r}})$ on the database scheme, we obtain the state we suggested as a result.*

$T_{t,\mathbf{r}}$

Employee	Dept.	Project	Manager
John	CS	A	Smith
John	CS	B	Smith
Bob	EE	B	Jones
Jim	MS	C	v_1
v_2	CS	A	Smith
v_3	IE	B	White
v_4	EE	B	Jones
Jim	v_5	v_6	White

$CHASE_F(T_{t,\mathbf{r}})$

Employee	Dept.	Project	Manager
John	CS	A	Smith
John	CS	B	Smith
Bob	EE	B	Jones
Jim	MS	C	White
v_2	CS	A	Smith
v_3	IE	B	White
v_4	EE	B	Jones
Jim	MS	v_5	White

r_{+_1}

Employee	Dept.	Project
John	CS	A
John	CS	B
Bob	EE	B
Jim	MS	C

r_{+_2}

Dept.	Manager	Project
CS	Smith	A
IE	White	B
EE	Jones	B
CS	Smith	B
MS	White	C

FIG. 4. $T_{t,\mathbf{r}}$, $CHASE_F(T_{t,\mathbf{r}})$ *and* $\mathbf{r}_+$ *for Example* 3.

Note that the insertion of a tuple on a set of attributes $X \subseteq U$ allows a form of "side effect," since it may cause the addition of some extra information for the attributes not in X. For instance, in Example 3, we showed that the insertion of the tuple with values *Jim* for E and *White* for M produces the insertion of the tuple $\langle \text{MS}, \text{White}, \text{C} \rangle$ in r_2. This tuple states that *White* is the manager of the *MS* department and is involved in project C and that this information is not provided directly by the user. This fact, however, is just a consequence of the weak instance model framework in which the FDs allow us to derive, in the representative instance, further information from tuples of a database. Thus, the insertion of a new tuple (over a relation or any set of attributes) may induce new values for old tuples in the representative instance. However, with our approach, this side effect is always minimal because we have defined the result of an insertion as the glb of all the potential results; in this way the original state is always changed as little as possible. We will come back on this issue in section 6, where we will show that, under certain conditions, the side effect can also be kept "under control."

3.3. Deletions. The definitions of deletions are somewhat symmetric with respect to those concerning insertions.

A state $\mathbf{r}_p$ is a *potential result* for the deletion of a tuple t from a state $\mathbf{r}$ if $\mathbf{r}_p \preceq \mathbf{r}$ and $t \widehat{\notin} \mathbf{r}_p$. The empty state is a consistent potential result for every deletion, and so there is no need to define the notions of possible and consistent results for deletions. A deletion is *deterministic* if the lub of the potential results is a potential result. When a deletion is deterministic, we consider the lub of the potential results to be *the* result of the deletion.

Let $\mathbf{r}$ be a consistent state and t be a tuple on X that x-belongs to $\mathbf{r}$. We derived the following characterizations for deletions.

LEMMA 5 (see [9, Lemma 9]). *The deletion of t from $\mathbf{r}$ is deterministic only if there is a relation scheme $R_i(X_i)$ such that $X \subseteq X_i$.*

DEFINITION 2 (state $\mathbf{r}_{-t}$). *The state $\mathbf{r}_{-t}$, or simply $\mathbf{r}_-$ when t is understood from the context, is obtained from $\mathbf{r}$ and t by removing, from each relation r_i such that $X \subseteq X_i$, each tuple t' such that $t'[X] = t[X]$.*

THEOREM 5 (see [9, Theorem 5]). *The deletion of t from $\mathbf{r}$ is deterministic if and only if* (i) *there is a relation scheme $R_i(X_i)$ such that $X \subseteq X_i$ and* (ii) $t \widehat{\notin} \mathbf{r}_-$.

4. Insertions for independent schemes. In the same way as query answering can be efficiently performed for independent schemes, we want to show that, for this meaningful class of schemes, updates defined over any subset of the universe can be managed efficiently.

With respect to deletions, Theorem 5 already gives an efficient way to check for determinism and to perform the update. With respect to insertions, the problem is more complex in general. Regarding possibility, Theorem 2 gives a complete characterization at the scheme level, which can be efficiently verified by using the closure algorithm proposed by Bernstein [11], but with regard to consistency and determinism, the tests require the chase of $T_{t,\mathbf{r}}$ (Theorems 3 and 4), a tableau involving the whole database state. Since computing the chase of a tableau takes polynomial time in the number of the rows of the tableau [2], it follows that the tests for consistency and determinism of an insertion require time and size polynomial with respect to the size of the database state.

In the next section we show that it is possible to derive alternative methods for checking consistency and determinism of insertions to independent schemes that are easier to implement and optimize.

4.1. Consistency. Throughout this section we will consider a consistent state $\mathbf{r}$ on a scheme $\mathbf{R} = \{R_1(X_1), \ldots, R_n(X_n)\}$ and the insertion of a tuple t over $X \subseteq U$ in $\mathbf{r}$, assuming that it is possible.

Let $\bar{t}$ be the "extension" of t with respect to $\mathbf{r}$ generated by Algorithm 1 (shown in Figure 5). It turns out that $\bar{t}$ has interesting properties, which make it fundamental in the efficient check of both consistency and determinism.

Let us introduce a property to be used shortly.

CONDITION 1. *There is no $V \to A \in F_j$ for $j \in \{1, \ldots, n\}$ such that*

(i) *$VA \subseteq \overline{X}$, and*

(ii) *there exists $t' \in r_j$ such that $t'[V] = \bar{t}[V]$ and $t'[A] \neq \bar{t}[A]$.*

LEMMA 6. *If Condition 1 holds, then $\bar{t}$ is uniquely defined.*

Proof. Condition 1 implies that when there are two or more alternatives at any step, those not chosen remain valid after the transformation and therefore can be applied later. ∎

ALGORITHM 1.
Input : *a tuple* t *over* $X \subseteq U$ *and a database state* $\mathbf{r}$;
Output: *the "extension"* $\overline{t}$ *of* t *with respect to* $\mathbf{r}$ *and the set of attributes* $\overline{X}$;
begin
  $t_U[X] := t$; /* t_U *is a tuple of distinct variables over* U */
  $W := X$;
  while *there exists some* $V \to A \in F_j$, $1 \le j \le n$, *such that* $V \subseteq W, A \notin W$
    and *there exists* $t' \in r_j$ *such that* $t'[V] = t_U[V]$
  do begin
    $t_U[A] := t'[A]$;
    $W := W \cup A$
  end;
  return $t_U[W]$ *and* W
end.

FIG. 5. *Algorithm for the computation of the extension of a tuple.*

DEFINITION 3 (state $\tilde{\mathbf{r}}$). *Let* $\overline{t}$ *be the extension of* t *with respect to* $\mathbf{r}$ *and* $\tilde{t}$ *be a tuple over the universe* U *obtained by extending* $\overline{t}$ *to* U *by means of "new constants," that is, a unique constants not already appearing in* $\mathbf{r}$. *Then the state* $\tilde{\mathbf{r}}$ *is obtained from* $\mathbf{r}$ *and* $\tilde{t}$ *by adding the tuple* $\tilde{t}[X_j]$ *to each relation* $r_j \in \mathbf{r}$ *on* $R_j(X_j)$.

LEMMA 7. *If* $\mathbf{R}$ *is independent and Condition* 1 *holds, then* $\tilde{\mathbf{r}}$ *is a consistent potential result for the insertion of* t *in* $\mathbf{r}$.

Proof. We have to show that (1) $\tilde{\mathbf{r}}$ is consistent and (2) $\tilde{\mathbf{r}}$ is a potential result.

(1) $\tilde{\mathbf{r}}$ is consistent: Since $\mathbf{R}$ is independent, it is sufficient to show that $\tilde{\mathbf{r}}$ is locally consistent. By way of contradiction assume that it is not. Let $\tilde{r}_j$ be a relation that violates the respective FDs F_j. Since $\mathbf{r}$ is consistent, r_j satisfies F_j, and so the violation has to involve the new tuple $\tilde{t}[X_j]$ together with a tuple $t' \in r_j$; that is, there is an FD $V \to A \in F_j$ such that $t'[V] = \tilde{t}[V]$ and $t'[A] \neq \tilde{t}[A]$. Since the values of $\tilde{t}$ over attributes in $U - \overline{X}$ are all new constants, $\tilde{t}[V] = t'[V]$ implies $V \subseteq \overline{X}$ and so $\overline{t}[V] = t'[V]$. Then the computation of $\overline{t}$ would also add A to W and so to $\overline{X}$ because of $t' \in r_j$, and therefore $\tilde{t}[A] = \overline{t}[A]$. Then, we would have $t'[V] = \overline{t}[V]$ and $t'[A] \neq \overline{t}[A]$, for some $V \to A \in F_j$ with $VA \subseteq \overline{X}$, against Condition 1.

(2) $\tilde{\mathbf{r}}$ is a potential result: By construction $\mathbf{r} \preceq \tilde{\mathbf{r}}$. Also, since the insertion of t in $\mathbf{r}$ is possible, by Theorem 2, there is a relation scheme $R_i(X_i) \in \mathbf{R}$ such that F implies the FD $X_i \to X$ and so $X_i^+ \supseteq X$. Let t' be the tuple in $CHASE_F(T_{\tilde{\mathbf{r}}})$ originating from $\pi_{X_i}(\tilde{t})$. Then, by Lemma 1, $t'[X_i^+] = \tilde{t}[X_i^+]$, and since $X_i^+ \supseteq X$, by definition of $\tilde{t}$ we have $t'[X] = t$. It follows that $t \in \pi_X(\mathrm{RI}_{\tilde{\mathbf{r}}})$, and so $\tilde{\mathbf{r}}$ is a potential result. □

THEOREM 6. *The insertion of* t *in a state* $\mathbf{r}$ *of an independent scheme is consistent if and only if Condition* 1 *holds.*

Proof. Only if. The construction of $\overline{t}$ can be seen as an initial sequence in the chase of $T_{t,\mathbf{r}}$ with respect to F. Then, violation of Condition 1 implies that $CHASE_F(T_{t,\mathbf{r}}) = T_\infty$ and so, by Theorem 3, the claims follows.

If. This proof is made by Lemma 7. □

This theorem gives us an effective and efficient method to check for consistency of insertions in independent schemes: instead of performing the chase of $T_{t,\mathbf{r}}$ (as requested by Theorem 3), it is sufficient to apply Algorithm 1 and to check for violations of FDs involving $\overline{t}$.

EXAMPLE 4. *The scheme in Example* 1 *is clearly independent. Now consider the first insertions in Example* 2. *We have* $\overline{t} = \langle \text{Jim}, \text{MS}, \text{White} \rangle$, *so the insertion is consistent since such a tuple does not violate the FDs that involves. Conversely, if we want to insert the tuple* $\langle \text{John}, \text{White} \rangle$, *we would have an inconsistent insertion since* $\overline{t} = \langle \text{John}, \text{CS}, \text{White} \rangle$, *and Condition* 1 *does not hold for the FD* $D \rightarrow M$ *and, for instance, the tuple* $\langle \text{CS}, \text{Smith}, \text{A} \rangle$ *of* r_2.

4.2. Determinism. Throughout this section we will consider a consistent database state $\mathbf{r}$ of a scheme $\mathbf{R} = \{R_1(X_1), \ldots, R_n(X_n)\}$ and the insertion of a tuple t over $X \subseteq U$ in $\mathbf{r}$, assuming that it is possible and consistent.

As in section 4.1, let $\overline{t}$ be the extension of t with respect to $\mathbf{r}$ generated by Algorithm 1, let $\tilde{t}$ be the tuple obtained by extending $\overline{t}$ to the universe U by means of a set of new constants C_{new}, and let $\tilde{\mathbf{r}}$ be the state obtained by adding to the original state $\mathbf{r}$ the projections of $\tilde{t}$ over the various relation schemes.

DEFINITION 4 (state $\hat{\mathbf{r}}$). *Let* $\tilde{\mathbf{r}}^*$ *be the completion of* $\tilde{\mathbf{r}}$, *that is,* $\tilde{\mathbf{r}}^* = \pi^{\downarrow}_{\mathbf{R}}(\text{RI}_{\tilde{\mathbf{r}}})$. *Then,* $\hat{\mathbf{r}}$ *is the state obtained from* $\tilde{\mathbf{r}}^*$ *by removing all the tuples having some new constant:* $\hat{\mathbf{r}} = \pi^{\downarrow}_{\mathbf{R}}[C_{new}](\text{RI}_{\tilde{\mathbf{r}}})$.

We have the following result for the state $\hat{\mathbf{r}}$ (we recall that $\mathbf{r}_+ = \pi^{\downarrow}_{\mathbf{R}} CHASE_F(T_{t,\mathbf{r}})$, where $T_{t,\mathbf{r}} = \text{RI}_{\mathbf{r}} \cup \{t_e\}$ and t_e is the tuple obtained by extending t to the universe U by means of unique variables).

THEOREM 7. *The state* $\hat{\mathbf{r}}$ *coincides with the state* $\mathbf{r}_+$.

Proof. By Lemma 7, $\tilde{\mathbf{r}}$ is a potential result, and therefore—since by Lemma 4 the state $\mathbf{r}_+$ is the glb of the potential result—we have $\mathbf{r}_+ \preceq \tilde{\mathbf{r}}$ and thus $\mathbf{r}_+ \preceq \tilde{\mathbf{r}}^*$ as every state is equivalent to its completion. Since $\mathbf{r}_+$ is complete (being constructed as the projection of a chased tableau), by equivalence of parts 1 and 4 of Theorem 1, we have that $\mathbf{r}_+$ is a relationwise subset of $\tilde{\mathbf{r}}^*$, and thus of $\hat{\mathbf{r}}$, which is obtained from $\tilde{\mathbf{r}}^*$ by eliminating tuples that do not appear in $\mathbf{r}_+$. Thus, to complete the proof we need to show that for every $R_i \in \mathbf{R}$, it is also the case that $\hat{r}_i \subseteq r_{+_i}$, where $\hat{r}_i \in \hat{\mathbf{r}}$ and $r_{+_i} \in \mathbf{r}_+$. We will prove this part by showing that it is possible to build a tableau T, such that $T \leq CHASE_F(T_{t,\mathbf{r}})$ and $\hat{\mathbf{r}} = \pi^{\downarrow}_{\mathbf{R}}(T)$. Since $\mathbf{r}_+ = \pi^{\downarrow}_{\mathbf{R}}(CHASE_F(T_{t,\mathbf{r}}))$, the fact that $\hat{\mathbf{r}}$ is a relationwise subset of $\mathbf{r}_+$ would then follow directly from the definitions of containment mapping and total projection.

Let $T_1 = \text{RI}_{\mathbf{r}} \cup \{\overline{t}_e\}$, where $\overline{t}_e$ is obtained by extending $\overline{t}$ to the universe U by means of unique variables. Since $\overline{t}$ is built using chase steps that are valid in $T_{t,\mathbf{r}}$ and since the chase is independent of the order of the individual chase steps [21], it follows that T_1 can be seen as an intermediate result in chasing $T_{t,\mathbf{r}}$ and that $CHASE_F(T_1) = CHASE_F(T_{t,\mathbf{r}})$. Similarly, let $\mathbf{r}_{\tilde{t}}$ be the state obtained by projecting $\tilde{t}$ over the various relational schemes and let $T_2 = \text{RI}_{\mathbf{r}} \cup CHASE_F(T_{\mathbf{r}_{\tilde{t}}})$. By construction, $T_{\tilde{\mathbf{r}}} = T_{\mathbf{r}} \cup T_{\mathbf{r}_{\tilde{t}}}$, so T_2 can be considered as an intermediate result in chasing $T_{\tilde{\mathbf{r}}}$, and $CHASE_F(T_2) = CHASE_F(T_{\tilde{\mathbf{r}}}) = \text{RI}_{\tilde{\mathbf{r}}}$. The tableau T_2 contains all the tuples of $\text{RI}_{\mathbf{r}}$ and, by Lemma 1, n tuples t_i such that $t_i[X_i^+] = \tilde{t}[X_i^+]$ for $1 \leq i \leq n$. Then, let ϕ be a function from D to D that (i) maps the new constants in $\tilde{t}$ to unique variables in T_2 and (ii) is the identity on all the other elements in D, and consider the tableau $T_3 = \phi(T_2)$. This tableau is composed of the tuples of $\text{RI}_{\mathbf{r}}$ and n tuples t'_i such that $t'_i[A] = \overline{t}[A]$ if $A \in \overline{X}$, and $t'_i[A]$ is a variable otherwise. It easily follows that $T_3 \leq T_1$ because of a containment mapping that is the identity on $\text{RI}_{\mathbf{r}}$ and maps the n tuples t'_i to $\overline{t}_e$. Now, let us consider the chase of T_2 and let ψ be the function that maps each symbol appearing in T_2 to the symbols to which it is changed by the chase, that is, $\psi(T_2) = CHASE_F(T_2)$, and let $\psi' = \phi \circ \psi$. The function ψ' coincides with ψ except for the variables v that have been changed to new constants by chasing T_2, that is, $\psi'(v) =$

$\phi(\psi(v)) = v_j$ if $\psi(s)$ is a new constants c_j such that $\phi(c_j) = v_j$, and $\psi'(s) = \psi(s)$ for all the other symbols s in T_2. Since all the chase steps that can be applied to T_2 are also valid in the tableau T_3 if we replace the new constants with unique variables, it follows that $\psi'(T_2)$ coincides with the chase of T_3, that is, $\psi'(T_2) = CHASE_F(T_3)$. If we let $T_4 = \phi(\mathrm{RI}_{\tilde{\mathbf{r}}})$, we have $T_4 = \phi(\psi(T_2)) = \psi'(T_2) = CHASE_F(T_3)$, and therefore, since $T_3 \leq T_1$, by part 2 of Lemma 2, we have $CHASE_F(T_3) \leq CHASE_F(T_1)$, and so $T_4 \leq CHASE_F(T_1) = CHASE_F(T_{t,\mathbf{r}})$.

Now, since ϕ maps new constants to new variables and is the identity on the other elements of D, we have that $\pi^{\downarrow}_{\mathbf{R}}(T_4)$ exactly coincides with $\hat{\mathbf{r}}$. As argued above, it follows that $\hat{\mathbf{r}}$ is a relationwise subset of $\mathbf{r}_+ = \pi^{\downarrow}_{\mathbf{R}}(CHASE_F(T_{t,\mathbf{r}}))$. □

COROLLARY 2. *The insertion of a tuple t in a state $\mathbf{r}$ is deterministic if and only if $t \widehat{\in} \hat{\mathbf{r}}$.*

Proof. The proof follows by Corollary 1 and Theorem 7. □

Corollary 2 gives us an alternative method to check for determinism that does not requires the chase of the special tableau $T_{t,\mathbf{r}}$ over the whole database.

EXAMPLE 5. *Consider again the insertion of $t = \langle \mathrm{Jim}, \mathrm{White} \rangle$ over EM in the state in Figure 1. We have in this case $\bar{t} = \langle \mathrm{Jim}, \mathrm{MS}, \mathrm{White} \rangle$ over EDM and $\tilde{t} = \langle \mathrm{Jim}, \mathrm{MS}, \mathrm{c}_1, \mathrm{White} \rangle$ over U. Hence, $\tilde{\mathbf{r}}$ is obtained by adding $\langle \mathrm{Jim}, \mathrm{MS}, \mathrm{c}_1 \rangle$ to r_1 and $\langle \mathrm{MS}, \mathrm{White}, \mathrm{c}_1 \rangle$ to r_2. In computing the completion of $\tilde{\mathbf{r}}$, the tuple $\langle \mathrm{MS}, \mathrm{White}, \mathrm{C} \rangle$ is also added to the second relation and so, by deleting the tuples with the new constant c_1, we obtain again the state we suggest as the result.*

In the following section we will show how the new method can be efficiently implemented if the scheme is independent. For this purpose, we now mention some properties of $T_{t,\mathbf{r}}$ and $\hat{\mathbf{r}}$ for the class of independent schemes. We recall that, given a tuple t over a set of attributes $X \subseteq U$, the tuple t_e denotes its extension to U by means of unique variables.

LEMMA 8. *If $\mathbf{R}$ is independent, then for any $R_j \in \mathbf{R}$ and for any $Y \to A \in F_j$ it is the case that $\pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}})) = \pi_{YA}(r_j) \cup \pi^{\downarrow}_{YA}(\{\bar{t}_e\})$.*

Proof. It is sufficient to prove that $\pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}})) \subseteq \pi_{YA}(r_j) \cup \pi^{\downarrow}_{YA}(\{\bar{t}_e\})$ as the other containment, by construction, is trivial. Let us consider the state $\tilde{\mathbf{r}}$: Since, by Lemma 7, it is a consistent potential result, we have that $\mathbf{r} \preceq \tilde{\mathbf{r}}$ and $t \widehat{\in} \tilde{\mathbf{r}}$. Hence, $\mathrm{RI}_{\mathbf{r}} \leq \mathrm{RI}_{\tilde{\mathbf{r}}}$ and $t \in \pi^{\downarrow}_{X}(\mathrm{RI}_{\tilde{\mathbf{r}}})$ and therefore, since all the variables in t_e are unique, $T_{t,\mathbf{r}} = \mathrm{RI}_{\mathbf{r}} \cup \{t_e\} \leq \mathrm{RI}_{\tilde{\mathbf{r}}}$. Thus, by part 3 of Lemma 2, it follows that $CHASE_F(T_{t,\mathbf{r}}) \leq CHASE_F(T_{\tilde{\mathbf{r}}})$. Now let $t' \in \pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}}))$. By equivalence of parts 2 and 3 of Lemma 1 we have $\pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}})) \subseteq \pi^{\downarrow}_{YA}(CHASE_F(T_{\tilde{\mathbf{r}}}))$; hence, $t' \in \pi^{\downarrow}_{YA}(CHASE_F(T_{\tilde{\mathbf{r}}}))$. Since $\mathbf{R}$ is independent and, by Lemma 3, $\pi^{\downarrow}_{YA}(CHASE_F(T_{\tilde{\mathbf{r}}})) = \pi_{YA}(\tilde{r}_j)$, it follows that $t' \in \pi_{YA}(\tilde{r}_j)$. Now, by definition of $\tilde{\mathbf{r}}$, we have that $t' \in \pi_{YA}(r_j)$ or $t' = \tilde{t}[YA]$. In the first case, the proof is complete. In the second case, we have two subcases: (i) $YA \subseteq \overline{X}$ and so $\tilde{t}[YA] = \bar{t}[YA]$, which would again prove the claim; (ii) $YA \not\subseteq \overline{X}$ and therefore $\tilde{t}[YA]$ contains constants not appearing in $CHASE_F(T_{t,\mathbf{r}})$ and therefore $\tilde{t}[YA] \notin \pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}}))$, which, however, contradicts $\tilde{t}[YA] = t' \in \pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}}))$. □

LEMMA 9. *If $\mathbf{R}$ is independent and the insertion of t in $\mathbf{r}$ is deterministic, then for any $R_i(X_i)$ and for any $Y \to A \in F_i$, it is the case that $\pi_{YA}(\hat{r}_i) = \pi_{YA}(r_i) \cup \pi^{\downarrow}_{YA}(\{\bar{t}_e\})$.*

Proof. If the insertion is deterministic, then by Theorem 4, we have that $\mathrm{RI}_{\mathbf{r}_+} \equiv CHASE_F(T_{t,\mathbf{r}})$, and so by Theorem 7 and by equivalence of parts 1 and 2 of Theorem 1, $\mathrm{RI}_{\hat{\mathbf{r}}} \equiv CHASE_F(T_{t,\mathbf{r}})$. Also, since $\mathbf{R}$ is independent, by Lemma 3 we have that

ALGORITHM 2.
Input: *a state* $\mathbf{s}$*, a tuple* t_Y *over* $Y \subseteq U$*, a set of attributes* $V \supseteq Y$*, a set of constants* C;
Output: *a relation* s_{out};
begin
  let E_V *be the AC-expression for* V;
  repeat
    select a scje $E_i = \pi_V(R_{i_0} \bowtie \pi_{Y_1 Z_1}(R_{i_1}) \bowtie \cdots \bowtie \pi_{Y_m Z_m}(R_{i_m}))$ *from* E_V;
    $s^{(0)} := \{t_Y\} \bowtie s_{i_0}$; /* $s_{i_0} \in \mathbf{s}$ *on* R_{i_0} */
    $W := Y \cup X_{i_0}$;
    $k := 0$;
    repeat
      select a component $\pi_{Y_j Z_j}(R_{i_j})$ *from* E_i *such that* $W \supseteq Y_j$*, choosing first those such that* $Y \cap Y_j Z_j \neq \emptyset$ *(if any)*;
      $s^{(k+1)} := s^{(k)} \bowtie \pi_{Y_j Z_j}(s_{i_j})$ /* $s_{i_j} \in \mathbf{s}$ *on* R_{i_j} */;
      $W := W \cup Y_j Z_j$;
      $k := k + 1$;
    until $(s^{(k)} = \emptyset)$ **or** (*all the components of* E_i *have been selected*);
    $s_{out} := \pi_V(s^{(k)}) - \{$*tuples with constants in* $C\}$;
  until $(s_{out} \neq \emptyset)$ **or** (*all scje of* E_V *have been selected*);
  return s_{out}
end.

FIG. 6. *Basic algorithm for update operations.*

for any $R_i(X_i) \in \mathbf{R}$ and for any FD $X \to A \in F_i$, it is the case that $\pi_{YA}(\hat{r}_i) = \pi^{\downarrow}_{YA}(\mathrm{RI}_{\hat{\mathbf{r}}})$. By equivalence of parts 2 and 3 of Theorem 1, it follows that $\pi_{YA}(\hat{r}_i) = \pi^{\downarrow}_{YA}(CHASE_F(T_{t,\mathbf{r}}))$, and so, by Lemma 8, $\pi_{YA}(\hat{r}_i) = \pi_{YA}(r_i) \cup \pi^{\downarrow}_{YA}(\{\bar{t}_e\})$. □

5. Algorithms for update operations. On the basis of the results of the previous sections, we present in this section efficient algorithms that can be used to update relational databases on independent schemes. Such algorithms exploit some known result on query answering for this class of schemes. In particular, we will use the results of Atzeni and Chan [5], who proved that, for independent schemes, the total projection of the representative instance can be obtained by means of a union of scjes and proposed an algorithm to compute and optimize this expression, which runs in polynomial time with respect to the size of the database scheme. In the following, we will denote by $E_X = \cup E_i$ the union of scjes for a set of attributes $X \subseteq U$ [5, Algorithm 5.8], and we will refer to E_X as the *AC-expression* for X.

5.1. Basic algorithm. In this subsection we provide an algorithm (shown in Figure 6) that will be used for several purposes in the rest of the paper, and whose aim is to test efficiently whether certain total tuple appears in the representative instance of a state defined on an independent scheme. More specifically, given a consistent state $\mathbf{s}$ of an independent scheme $\mathbf{R}$, a tuple t_Y over $Y \subseteq U$, a set of attributes $V \subseteq U$ such that $Y \subseteq V$, and a set of constants C, Algorithm 2 verifies whether there exists total tuples on V in $\mathrm{RI}_{\mathbf{s}}$, without constants in C, that coincide with t_Y on the attributes Y, that is, whether $\sigma_{Y=t_Y}(\pi^{\downarrow}_V[C](\mathrm{RI}_{\mathbf{s}}))$ is not empty. If $Y = V$ and $C = \emptyset$, the algorithm simply tests whether $t_Y \widehat{\in} \mathbf{s}$. The role of the set of attributes V and of the set of constants C will be clarified later.

The search is efficiently performed by joining t_Y with tuples of $\mathbf{s}$ on the basis of the scje in the AC-expression E_V for V, giving precedence to the components having some attribute in Y. This corresponds in performing selections on the values in t_Y as early as possible while computing $\sigma_{Y=t_Y}(E_V(\mathbf{r}))$.

In the inner loop of the algorithm the expression $\sigma_{Y=t_Y}(E_i(\mathbf{r}))$ for an scje $E_i \in E_V$ is evaluated. The loop halts as soon as the result turns out to be empty or when the scje has been completely computed. Then, the derived tuples having no constants in C are stored in the relation s_{out}. The algorithm stops as soon as s_{out} is not empty or when all the scjes in E_V have been examined. Thus, at termination, we have that s_{out} contains tuples of $\sigma_{Y=t_Y}(E_V(\mathbf{s}))$ without constants in C and that $s_{out} \neq \emptyset$ if and only if $\sigma_{Y=t_Y}(E_V(\mathbf{s}))$ contains at least one tuple without constants in C. Since in [5] it is proved that a tuple $t' \in \pi^{\downarrow}_V(\mathrm{RI}_{\mathbf{s}})$ if and only if $t' \in E_V(\mathbf{s})$, the following result easily follows.

LEMMA 10. *Assume that Algorithm* 2 *receives as input a tuple* t_Y *over* $Y \subseteq U$, *a consistent state* $\mathbf{s}$ *of a scheme* $\mathbf{R}$, *a set of attributes* $V \subseteq U$ *such that* $Y \subseteq V$ *and a set of constants* C. *Then, at termination,* (1) $s_{out} \subseteq \pi^{\downarrow}_V[C](\sigma_{Y=t_Y}(\mathrm{RI}_{\mathbf{s}}))$, *and* (2) $s_{out} \neq \emptyset$ *if and only if* $\pi^{\downarrow}_V[C](\sigma_{Y=t_Y}(\mathrm{RI}_{\mathbf{s}})) \neq \emptyset$.

COROLLARY 3. *Let* $Y = V$ *and* $C = \emptyset$, *then the output relation* s_{out} *of Algorithm* 2 *is not empty if and only if* $t_Y \widehat{\in} \mathbf{s}$.

Note that, for efficiency purposes, Algorithm 2 does not compute a complete total projection of the representative instance. Note also that if $\pi^{\downarrow}_V(\sigma_{Y=t_Y}(\mathrm{RI}_{\mathbf{s}}))$ is not empty, then s_{out} is not deterministic, since it depends on the order in which the scjes have been selected. However, this is not important since, as we will see, we just need a relation satisfying the above properties.

5.2. Performing insertions. Let $\mathbf{r}$ be a consistent state of a scheme $\mathbf{R}$, and consider the insertion of a tuple t over $X \subseteq U$ in $\mathbf{r}$, assuming that it is possible and consistent. In section 4 it has been shown that the property of determinism for insertions can be verified on the state $\hat{\mathbf{r}}$. The construction of this state requires the computation of the completion of the state $\tilde{\mathbf{r}}$. We will show now that we do not need to compute the full completion of $\tilde{\mathbf{r}}$, since, as suggested by Lemma 9, it is sufficient to find only those tuples without new constants of $\hat{\mathbf{r}}^*$ that coincide with $\bar{t}$ on the attributes involved in some FD.

More specifically, let us consider the following database state. Again, $\bar{t}$ denotes the extension of t with respect to $\mathbf{r}$, $\tilde{t}$ the tuple obtained by extending $\bar{t}$ to the universe U by means of new constants, $\tilde{\mathbf{r}}$ the state obtained by adding to the original state $\mathbf{r}$ the projections of $\tilde{t}$ over the various relation schemes, and $\tilde{\mathbf{r}}^*$ its completion.

DEFINITION 5 (state $\check{\mathbf{r}}$). *A state* $\check{\mathbf{r}}$ *is obtained from* $\mathbf{r}$ *and* $\tilde{\mathbf{r}}$ (1) *by adding the tuple* $\bar{t}[X_i]$ *to each relation* $r_i \in \mathbf{r}$ *on* $R_i(X_i)$ *such that* $\overline{X} \supseteq X_i$, *and* (2) *by adding at least one tuple without new constants* $t_j \in \tilde{r}^*_j$, *such that* $t_j[YA] = \bar{t}[YA]$ *and* $t_j \notin r_j$, *to each relation* $r_j \in \mathbf{r}$ *over* $R_j(X_j)$ *such that* $\overline{X} \not\supseteq X_j$ *and* $\overline{X} \supseteq YA$ *for some* $Y \to A \in F_j$,

An important point here is that, if the deterministic condition is satisfied, in case (2) above there must exist at least one tuple without new constants t_j in $\tilde{r}^*_j$ such that $t_j[YA] = \bar{t}[YA]$ and $t_j \notin r_j$. This follows from the fact that, by Lemma 9, $\pi_{YA}(\hat{r}_j) = \pi_{YA}(r_i) \cup \pi^{\downarrow}_{YA}(\{\bar{t}_e\})$ for each $Y \to A \in F_j$ and the fact that $\hat{r}_j$ is obtained from $\tilde{r}^*_j$ by just deleting tuples with new constants. Note also that, in general, several tuples may satisfy this property.

We have the following results for $\check{\mathbf{r}}$.

LEMMA 11. *If* $\mathbf{R}$ *is independent and the insertion of* t *in* $\mathbf{r}$ *is deterministic, then*

$\check{\mathbf{r}} \sim \hat{\mathbf{r}}$.

Proof. We have to show that for deterministic insertions (1) $\hat{\mathbf{r}} \preceq \check{\mathbf{r}}$ and (2) $\check{\mathbf{r}} \preceq \hat{\mathbf{r}}$.

(1) $\hat{\mathbf{r}} \preceq \check{\mathbf{r}}$. We prove this part by showing that, for every $R_i(X_i) \in \mathbf{R}$ and for every $t_i \in \hat{r}_i$, either (i) $t_i \in \check{r}_i$ or (ii) $t_i \in \pi^{\downarrow}_{X_i}(\mathrm{RI}_{\check{\mathbf{r}}})$. It would follow that every $t_i \in \hat{r}_i$ belongs to the relation $\check{r}_i^*$ in the completion $\check{\mathbf{r}}^*$ of $\check{\mathbf{r}}$. Since, by Theorem 7, $\hat{\mathbf{r}}$ coincides with a complete state and is therefore itself complete, the fact that $\hat{\mathbf{r}} \preceq \check{\mathbf{r}}$ then follows by the equivalence of parts 1 and 4 of Theorem 1.

Thus, let $t_i \in \hat{r}_i$ for some $R_i(X_i) \in \mathbf{R}$. By construction of $\hat{\mathbf{r}}$, we have three possible cases: (a) $t_i \in r_i$, where $r_i \in \mathbf{r}$—that is, t_i belongs to a relation of the original state; (b) $t_i \notin r_i$ and $\overline{X} \supseteq X_i$, and so $t_i = \bar{t}[X_i]$; and (c) $t_i \notin r_i$ and $\overline{X} \not\supseteq X_i$, and so t_i has been generated in chasing $T_{\check{\mathbf{r}}}$. In the first two cases we also have $t_i \in \check{r}_i$ by construction. With respect to the third case, we will show that t_i can be generated by chasing $T_{\check{\mathbf{r}}}$.

Thus, let t_i be a tuple of $\hat{r}_i$ on $R_i(X_i) \in \mathbf{R}$, such that $t_i \notin r_i$ and $\overline{X} \not\supseteq X_i$. Since, by Theorem 7, $\hat{\mathbf{r}} = \mathbf{r}_+$, we have that, by construction of $\mathbf{r}_+$, the tuple t_i is also generated by chasing $T_{t,\mathbf{r}}$. Moreover, since the insertion is deterministic, by Theorem 4, $\mathit{CHASE}_F(T_{t,\mathbf{r}})$ is equivalent to the representative instance of a database state. It follows that t_i originates in $\mathit{CHASE}_F(T_{t,\mathbf{r}})$ from a tuple t_0 of a relation $r_0 \in \mathbf{r}$ (indeed, t_0 can not originate from $\bar{t}$ since this would imply that $t_i = \bar{t}[X_i]$ and so $\overline{X} \supseteq X_i$). Let $\chi = \tau_1, \ldots, \tau_m$ be the sequence of chase steps that allows us to generate t_i from t_0 in the chase of $T_{t,\mathbf{r}}$. Since $\mathbf{R}$ is independent, by Lemma 8, each τ_k in χ promotes a variable to a constant in $\mathbf{r}$ or in t, because of a tuple t_j and an FD $Y \to A$, such that $t_j[YA] \in \pi_{YA}(r_j) \cup \pi^{\downarrow}_{YA}(\{\bar{t}_e\})$.

Now, for each $Y \to A \in F_j$ such that $\overline{X} \supseteq YA$, if $\overline{X} \supseteq X_j$, the tuple $\bar{t}[X_i]$ is in $\check{r}_j$ by construction. Moreover, if $\overline{X} \supseteq YA$ and $\overline{X} \not\supseteq X_j$, we have argued above that, for deterministic insertions, there must exist at least one tuple without new constants t'_j in $\tilde{r}_j^*$ such that $t'_j[YA] = \bar{t}[YA]$ and so, by construction, a tuple satisfying this property is also in $\check{r}_j$. It follows that the above sequence of chase step χ is also valid in $T_{\check{\mathbf{r}}}$; that is, it allows us to generate t_i from t_0 (which, by construction, belongs to $\check{\mathbf{r}}$), by chasing $T_{\check{\mathbf{r}}}$. As argued above, this concludes part (1) of the proof.

(2) $\check{\mathbf{r}} \preceq \hat{\mathbf{r}}$: the proof easily follows from their definitions. □

THEOREM 8. *For independent schemes the insertion of t in $\mathbf{r}$ is deterministic if and only if $t \widehat{\in} \check{\mathbf{r}}$.*

Proof. Only if. If the insertion is deterministic, then by Corollary 2, $t \widehat{\in} \hat{\mathbf{r}}$ and, by Lemma 11, $\check{\mathbf{r}} \sim \hat{\mathbf{r}}$. It follows that $t \widehat{\in} \check{\mathbf{r}}$.

If. Since, by construction, $\check{\mathbf{r}} \preceq \hat{\mathbf{r}}$, if $t \widehat{\in} \check{\mathbf{r}}$, then it is also the case that $t \widehat{\in} \hat{\mathbf{r}}$ and therefore, by Corollary 2, the insertion is deterministic. □

A state $\check{\mathbf{r}}$ can be efficiently derived from $\mathbf{r}$ and $\tilde{\mathbf{r}}$ by using Algorithm 3, shown in Figure 7. In this algorithm, C_{new} denotes the set of new constants used in the construction of $\tilde{t}$, and, for each $R_i(X_i) \in \mathbf{R}$, $Y^+_{F_i}$ denotes the *local closure* of a set of attributes $Y \subseteq X_i$ with respect to F_i. The following lemma confirms the correctness of the algorithm.

LEMMA 12. *Assume that Algorithm* 3 *receives as input a state $\mathbf{r}$ of an independent scheme $\mathbf{R}$, a tuple t over $X \subseteq U$, and its extensions $\bar{t}$ and $\tilde{t}$. Then, the output of the algorithm is a state $\check{\mathbf{r}}$.*

Proof. Since the tuples added to $\mathbf{r}_{out}$ in step (a) of Algorithm 3 belong to $\check{\mathbf{r}}$ by definition, it is sufficient to show that in step (b), only tuples satisfying condition (2) of the definition of $\check{\mathbf{r}}$, are added to $\mathbf{r}_{out}$, and nothing else. First, note that if there is a tuple $t_i \in \tilde{r}_i$ such that $t_i[YA] = \bar{t}[YA]$ for some $Y \to A$ in F_i, it easily

ALGORITHM 3.
Input : $\mathbf{r}$, t *over* $X \subseteq U$, $\bar{t}$ *over* $\overline{X}$, $\tilde{t}$ *over* U;
Output: *a state* $\check{\mathbf{r}}$;
begin
  $\mathbf{r}_{out} := \mathbf{r}$;
  for each $R_i(X_i) \in \mathbf{R}$
  do if $\overline{X} \supseteq X_i$
(a) **then** $r_{out_i} := r_{out_i} \cup \{\bar{t}[X_i]\}$
  else for each $Y \to A \in F_i$ *such that* $\overline{X} \supseteq YA$
    do if *does not exist* $t' \in r_{out_i}$ *such that* $t'[YA] = \bar{t}[YA]$
      then begin
        execute Algorithm 2 *over* $\tilde{\mathbf{r}}$, $\bar{t}[Y_{F_i}^+]$, X_i *and* C_{new};
(b) $r_{out_i} := r_{out_i} \cup s_{out}$
      end;
  return $\mathbf{r}_{out}$
end.

FIG. 7. *Algorithm for the generation of a state* $\check{\mathbf{r}}$.

follows that $\bar{t}$ is defined on $Y_{F_i}^+$ and $t_i[Y_{F_i}^+] = \bar{t}[Y_{F_i}^+]$. Then, by Lemma 10, in step (b) of Algorithm 3, we add to $\mathbf{r}_{out}$ at least one tuple (if any) without new constants in $\pi^{\downarrow}_{X_i}[C_{new}](\sigma_{Y_{F_i}^+ = \bar{t}[Y_{F_i}^+]}(\mathrm{RI}_{\tilde{\mathbf{r}}}))$. It follows that, for each relation $r_j \in \mathbf{r}$ over $R_j(X_j) \in \mathbf{R}$ such that $\overline{X} \not\supseteq X_j$ and $\overline{X} \supseteq YA$ for some $Y \to A \in F_j$, we add, in step (b) of Algorithm 3, at least one tuple $t_j \in \tilde{r}_j^*$ without new constants such that $t_j[YA] = \bar{t}[YA]$ and $t_j \notin r$, and nothing else. □

EXAMPLE 6. *Let* $\mathbf{r}$ *be the state in Figure* 1 *and* $t = \langle \mathrm{Jim}, \mathrm{White} \rangle$ *over* EM. *We have shown in Example* 4 *that* $\bar{t} = \langle \mathrm{Jim}, \mathrm{MS}, \mathrm{White} \rangle$ *over* $\overline{X} = EDM$. *Let* $\tilde{t} = \langle \mathrm{Jim}, \mathrm{MS}, \mathrm{c}_1, \mathrm{White} \rangle$ *and consider the execution of Algorithm* 3 *over these inputs. We have* $\overline{X} \not\supseteq X_1 = EDP$ *and* $\overline{X} \supseteq ED$, *where* $E \to D \in F_1$, *but* $\bar{t}[ED] \in \pi_{ED}(r_1)$ *and therefore* $r_{out_1} = r_1$. *We then have* $\overline{X} \not\supseteq X_2 = DMP$, $\overline{X} \supseteq DM$ *for the functional dependency* $D \to M \in F_2$, *and* $\bar{t}[DM] \notin \pi_{DM}(r_2)$. *Thus, in this case, Algorithm* 2 *needs to be executed with* $\tilde{\mathbf{r}}$, $\langle \mathrm{MS}, \mathrm{White} \rangle$, DMP, *and* $\{c_1\}$ *as inputs. We have* $E_{DMP} = R_2 \cup \pi_{DMP}(R_1 \bowtie \pi_{DM}(R_2))$. *For the first scje in* E_{DMP} *we obtain* $s_{out} = \emptyset$, *since* $\bar{t}[DM] \bowtie \tilde{r}_2 = \{\langle \mathrm{MS}, \mathrm{c}_1, \mathrm{White} \rangle\}$. *For the second scje we have* $\pi_{DMP}(\bar{t}[DM] \bowtie \tilde{r}_1 \bowtie \pi_{DM}(\tilde{r}_2)) = \{\langle \mathrm{MS}, \mathrm{C}, \mathrm{White} \rangle, \langle \mathrm{MS}, \mathrm{c}_1, \mathrm{White} \rangle\}$. *Hence, we obtain:* $r_{out_2} = r_2 \cup \{\langle \mathrm{MS}, \mathrm{C}, \mathrm{White} \rangle\}$. *Thus, in this case,* $\check{\mathbf{r}}$ *coincides with* $\hat{\mathbf{r}}$ (*see Example* 5) *and* $t \widehat{\in} \check{\mathbf{r}}$—*this confirms the determinism of the insertion of* t *in* $\mathbf{r}$.

We are now ready to present Algorithm 4 (shown in Figure 8), which summarizes all phases of insert operations to independent schemes.

Step (1) of Algorithm 4 checks for possibility (Theorem 2). This test requires the computation of closures of sets of attributes, an operation that can be performed in time $O(\|F\|)$, where $\|F\|$ is the size of the description of F, by using Bernstein's algorithm [10]. Since the closure has to be performed $|\mathbf{R}|$ times in the worst case, where $|\mathbf{R}|$ is the number of relation schemes in $\mathbf{R}$, it follows that testing for possibility is bounded by $O(|\mathbf{R}| \times \|F\|)$.

In step (2) the extension $\bar{t}$ of t with respect to the state $\mathbf{r}$ is computed by using Algorithm 1. This algorithm requires the computation of the closure of a set of attributes and, at each step of the computation, the selection of a tuple given a value

ALGORITHM 4.
Input : **r** *and* t *over* $X \subseteq U$*;*
Output: "not possible" | "not consistent" | "not deterministic" |
the result of the insertion of t *in* **r***;*
begin
(1) **if** *not exists* $R_i(X_i) \in \mathbf{R}$ *such that* $X_i^+ \supseteq X$
then return "not possible" **and stop***;*
(2) *compute* $\overline{t}$ *and* $\overline{X}$ *using Algorithm* 1;
(3) **if** *Condition* 1 *does not hold*
then return "not consistent" **and stop***;*
(4) *compute* $\check{\mathbf{r}}$ *using Algorithm* 3;
(5) *verify that* $t \widehat{\in} \check{\mathbf{r}}$ *using Algorithm* 2;
(6) **if** $s_{out} = \emptyset$
then return "not deterministic"
else return $\check{\mathbf{r}}$
end.

FIG. 8. *Algorithm for the execution of insert operations.*

on the left-hand side of an FD. The selection time depends on the cardinality of the relation, but it can be strongly reduced by defining indexes on the left-hand side of all the FDs in F. Let $k_{F,\mathbf{r}}$ be the maximum time needed to search for tuples in a database state **r**, given a value on the left-hand side of an FD in F. The cost of step (2) is then proportional to $\|F\| \times k_{F,\mathbf{r}}$. Under the same hypothesis, Condition 1 in step (3) (which, by Theorem 6, checks for possibility) can be tested in time $O(|F| \times k_{F,\mathbf{r}})$, where $|F|$ is the cardinality of the set of FDs F. This is because in an independent scheme each FD is embedded in at most one relational scheme [17].

In step (4) the state $\check{\mathbf{r}}$ is computed with Algorithm 3.

This algorithm requires, in the worst case, the execution of Algorithm 2 a number of times that, by the above property of independent schemes, is bounded by $|F|$. For any given X, there are at most as many scjes in E_X as relation schemes in **R** [5, 3]. It follows that the execution of Algorithm 2 corresponds to the execution of a number of relational algebra expressions bounded by $|F'| \times |\mathbf{R}|$, where F' denotes the FDs $Y' \rightarrow A'$ in F, such that there is no other FD $Y \rightarrow A$ in some $F_i \subseteq F$, for which $Y_{F_i}^+ \supseteq Y'A'$. The cost of each expression is bounded by $|r_{max}| \times k_{F,\mathbf{r}} \times |F|$, where $|r_{max}|$ is the size of the largest relation in **r**. In fact, the computation starts with a relation not larger than $|r_{max}|$ and then performs a number of joins, bounded by $|F|$, that simply require for each tuple in the intermediate relation the search for a tuple in a relation, given a value for its key (which is the left-hand side of an FD). This is because, at each step, the attributes of the intermediate relation include the left-hand side of the FD over which each component is projected. Note also that the size of the computed relation is always bounded by $|r_{max}|$. In sum, step (4) is bounded by $|r_{max}| \times |F|^2 \times |\mathbf{R}| \times k_{F,\mathbf{r}}$. On average however, it turns out that the cost of this operation is quite limited. This is because (i) updates often involve only a very small portion of FDs in F, (ii) relational expressions are optimized as described in [5] and selections are performed as early as possible, and (iii) the number of scjes for a set of attributes is small when the scheme enjoys the desirable property of "independent updatability" [16].

Finally, step (5) tests for determinism (Theorem 8 and Corollary 3) and requires

ALGORITHM 5.
Input : $\mathbf{r}$ *and* t *over* $X \subseteq U$*;*
Output: "not deterministic" | *the result of the deletion of* t *from* $\mathbf{r}$;
begin
(1) **if** *not exists* $R_i(X_i) \in \mathbf{R}$ *such that* $X_i \supseteq X$
    then return "not deterministic" **and stop**;
(2) **for each** $R_i(X_i) \in \mathbf{R}$ **do if** $X_i \supseteq X$
    then $r_{-_i} := r_i - \sigma_{X=t}(r_i)$;
(3) *verify that* $t \widehat{\in} \mathbf{r}_-$ *using Algorithm* 2;
(4) **if** verified
    then return "not deterministic" **and stop**
    else return $\mathbf{r}_-$
end.

FIG. 9. *Algorithm for the execution of delete operations.*

the execution of Algorithm 2 once more. Note that the state $\tilde{\mathbf{r}}$ as well as the state $\check{\mathbf{r}}$ do not need to be effectively constructed. So, Algorithms 2 and 3 could be slightly modified in order to work on the tuples in $\mathbf{r}$, $\tilde{\mathbf{r}}$, and $\check{\mathbf{r}}$, without actually adding tuples to $\mathbf{r}$.

From the discussion above, it turns out that Algorithm 4 provides a practical and efficient way to perform the insertion of a tuple t over any set of attributes $X \subseteq U$ in a state of an independent scheme.

EXAMPLE 7. *Examples* 4 *and* 6 *deal with the execution of all the steps of Algorithm* 4 *for the tuple* $t = \langle \text{Jim}, \text{White} \rangle$ *over* EM *and the state in Figure* 1.

5.3. Performing deletions. By the results of the previous sections it is also possible to give an efficient method for performing delete operations on a database state of an independent scheme. This algorithm is shown in Figure 9.

In step (1), the necessary condition for determinism of Lemma 5 is tested: it requires time proportional to $|\mathbf{R}|$. Then, in step (2), the state $\mathbf{r}_-$ is computed by executing a number of selections, again bounded by $|\mathbf{R}|$. Then, by Theorem 5 and Corollary 3, step (3) tests for determinism. This requires one execution of Algorithm 2, which is optimized as discussed in the previous subsection. Also in this case, the algorithm can be slightly modified in order to work on the tuples $\mathbf{r}$ and $\mathbf{r}_-$ without actually deleting tuples from $\mathbf{r}$.

Thus, again, given a state $\mathbf{r}$ of an independent scheme and a tuple t over any set of attributes $X \subseteq U$, Algorithm 5 provides a practical and efficient way to perform the deletion of t from $\mathbf{r}$.

EXAMPLE 8. *Assume that we want to delete the tuple* $t = \langle \text{Smith}, \text{B} \rangle$ *over* MP *from the state in Figure* 1. *The condition in step* (1) *of Algorithm* 5 *is verified for the scheme* $R_2(DMP)$. *Then, the state* $\mathbf{r}_-$ *is obtained in step* (2) *by deleting the tuple* $\langle \text{CS}, \text{Smith}, \text{B} \rangle$ *from* r_2. *However, it is easy to see that this tuple can be reconstructed from the tuple* $\langle \text{John}, \text{CS}, \text{B} \rangle$ *in* r_1 *and the tuple* $\langle \text{CS}, \text{Smith}, \text{A} \rangle$ *in* r_2. *It follows that in step* (3) *we obtain* $t \widehat{\in} \mathbf{r}_-$ *and therefore the insertion is not deterministic. Conversely, the deletion of* $t = \langle \text{White}, \text{B} \rangle$ *over the same attributes is deterministic since in this case we would have* $t \widehat{\notin} \mathbf{r}_-$.

6. Possible simplifications of the algorithms. In this section we show that, under certain further assumptions, update operations can be managed more easily.

6.1. Insertions. We recall that a database scheme $\mathbf{R}$ is *separable* if it is independent and every consistent state on $\mathbf{R}$ is complete [16]. (Chan and Mendelzon also provided an efficient test for separability.)

Now, let $\mathbf{r}$ be a state of a database scheme $\mathbf{R}$, t be a tuple over $X \subseteq U$, and $\bar{t}$ be the extension of t with respect to $\mathbf{r}$. Let us consider the following state.

DEFINITION 6 (state $\bar{\mathbf{r}}$). *The state $\bar{\mathbf{r}}$ is obtained from $\mathbf{r}$ and $\bar{t}$ by adding the tuple $\bar{t}[X_j]$ to each relation $r_j \in \mathbf{r}$ on $R_j(X_j) \in \mathbf{R}$ such that $X_j \subseteq \overline{X}$.*

THEOREM 9. *If $\mathbf{R}$ is separable, then $\bar{\mathbf{r}} = \mathbf{r}_+$.*

Proof. If $\mathbf{R}$ is separable, then the state $\tilde{\mathbf{r}}$ coincides with its completion $\tilde{\mathbf{r}}^*$, and therefore, by construction, $\hat{\mathbf{r}}$ can be obtained by eliminating from $\tilde{\mathbf{r}}$ the tuples with new constants. The state we obtain clearly coincides with $\bar{\mathbf{r}}$, and therefore, by Theorem 7, the claim follows. □

COROLLARY 4. *The insertion of a tuple t in a state $\mathbf{r}$ on a separable scheme is deterministic if and only if $t \widehat{\in} \bar{\mathbf{r}}$.*

Proof. The proof follows by Corollary 1 and Theorem 9. □

By this result, the test for determinism and the computation of the minimum result for insertions to separable schemes can be done more efficiently since in this case it is not required to compute the state $\check{\mathbf{r}}$ as it suffices to refer to the state $\bar{\mathbf{r}}$ which can be easily generated.

EXAMPLE 9. *The scheme of the state $\mathbf{r}$ in Figure 1 is independent but not separable since, for instance, the state obtained by deleting the tuple $\langle \mathrm{CS}, \mathrm{Smith}, \mathrm{B} \rangle$ from r_2 is not complete (see Example 8). Therefore, we have in general $\bar{\mathbf{r}} \neq \mathbf{r}_+$. In fact, for to the tuple $t = \langle \mathrm{Jim}, \mathrm{White} \rangle$ over EM, we have that $\bar{t} = \langle \mathrm{Jim}, \mathrm{MS}, \mathrm{White} \rangle$ and so $\bar{\mathbf{r}} = \mathbf{r}$, whereas we have shown that the insertion of t in $\mathbf{r}$ is deterministic. It is easy to show that, if the second relation would contain only the attributes D and P, the scheme would be separable. In this case the insertion above can be performed by adding to r_2 the tuple $\langle \mathrm{MS}, \mathrm{White} \rangle$, which is indeed embedded in $\bar{t}$.*

The state $\bar{\mathbf{r}}$ has an interesting property: it contains only the tuples that can be derived directly from t by extending this tuple with values from tuples of $\mathbf{r}$, using the FDs in F. Then, by simply computing the extension of t with Algorithm 1, we immediately know not only the tuples to insert in the original database, but also the side effect generated by the insertion. Therefore, when $\bar{\mathbf{r}}$ is the result of the insertion, we can keep the side effect under control. Unfortunately, as shown in Example 9, even for independent schemes, $\bar{\mathbf{r}}$ is not always the correct result, and insertion operations require, in the general case, a more involved computation, as described in section 5.

Interestingly however, it is possible to give for independent schemes "local" conditions at scheme level (which therefore can be efficiently tested) that allow us to refer to the state $\bar{\mathbf{r}}$ for insert operations even for schemes that are nonseparable. Let us consider the following property which refer to the insertion of a tuple t over $X \subseteq U$ in a state $\mathbf{r}$ of a scheme $\mathbf{R}$.

CONDITION 2. *For every relational scheme $R_i(X_i) \in \mathbf{R}$, at least one of the following holds:*

(i) $X_i \subseteq \overline{X}$,

(ii) $YA \not\subseteq \overline{X}$ *for any* $Y \rightarrow A \in F_i$,

(iii) F_i *contains an FD whose left-hand side is a superkey of* X_i.

THEOREM 10. *Let $\mathbf{R}$ be independent and assume that Condition 2 holds. Then, the insertion of t in $\mathbf{r}$ is deterministic if and only if $t \widehat{\in} \bar{\mathbf{r}}$.*

Proof. Only if. We prove this part by showing that if, for an independent scheme, Condition 2 holds and the insertion is deterministic, then for every $R_i(X_i) \in \mathbf{R}$

such that $X_i \not\subseteq \overline{X}$ it is the case that $\hat{r}_i = r_i$. The fact that $t \widehat{\in} \overline{\mathbf{r}}$ would then follow by Theorem 8 and the fact that in this case, by construction, $\check{\mathbf{r}} = \overline{\mathbf{r}}$. So, by way of contradiction, assume that $\mathbf{R}$ is independent, Condition 2 holds, and there is a scheme $R_i(X_i) \in \mathbf{R}$ such that $X_i \not\subseteq \overline{X}$ and $\hat{r}_i \neq r_i$. By definition of $\hat{\mathbf{r}}$, this implies that (i) $YA \subseteq \overline{X}$ for some $Y \rightarrow A \in F_i$, (ii) there is no tuple in r_i which coincides with $\overline{t}$ on YA, and (iii) there is a tuple $t' \in \tilde{r}_i^*$ without new constants such that $t'[YA] = \overline{t}[YA]$. But because of Condition 2, if $X_i \not\subseteq \overline{X}$ and $YA \subseteq \overline{X}$ for some $Y \rightarrow A \in F_i$, then there must exist an FD $Z \rightarrow B \in F_i$ such that Z is a superkey of X_i. Now, since the scheme is independent, by Lemma 3, $\pi^{\downarrow}_{ZB}(CHASE_F(T_{\check{\mathbf{r}}})) = \pi_{ZB}(\tilde{r}_i)$, and since Z is a superkey, this implies $\pi^{\downarrow}_{X_i}(CHASE_F(T_{\check{\mathbf{r}}})) = \tilde{r}_i$ and therefore $t' = \tilde{t}[X_i]$. This, in turn, implies that $t' = \overline{t}[X_i]$, since t' does not contain new constants, and therefore $X_i \subseteq \overline{X}$—a contradiction.

If. If $t \widehat{\in} \overline{\mathbf{r}}$, then it is also the case that $t \widehat{\in} \check{\mathbf{r}}$ since, by construction, $\overline{\mathbf{r}} \preceq \check{\mathbf{r}}$. Then, the claim follows by Theorem 8. □

EXAMPLE 10. *Consider again the insertion of the tuple* $t = \langle \text{Jim}, \text{White} \rangle$ *over* EM *in the state* $\mathbf{r}$ *in Figure* 1: *In this case Condition* 2 *is not verified since* $\overline{X} = EDM$. *For* $R_1(X_1)$ *we have* $X_1 \not\subseteq \overline{X}$, $E \rightarrow D \in F_1$ *and is embedded in* $\overline{X}$, *and the left-hand side of* $E \rightarrow D$ *does not contain a superkey of* X_1. *However, if we consider the insertion of* $t = \langle \text{Jim}, \text{D} \rangle$ *over* EP *in the same state, we obtain* $\overline{t} = \langle \text{Jim}, \text{MS}, \text{D} \rangle$, *and since in this case Condition* 2 *holds (*$X_1 \subseteq \overline{X}$ *and* $YA \not\subseteq \overline{X}$ *for any* $Y \rightarrow A \in F_2$*), the result of this insertion can be obtained by simply adding* $\overline{t}$ *to the first relation.*

Testing for Condition 2 on an independent scheme requires, in the worst case, time proportional to $|F| \times \|F\|$, since, as we have said before, each FD is embedded in at most one relational scheme. This test can be performed after step (3) of Algorithm 4, and if it succeeds, then steps (4)–(6) of Algorithm 4 can be replaced by just one step, verifying (by using Algorithm 2) that $t \widehat{\in} \overline{\mathbf{r}}$.

6.2. Deletions. With respect to deletions, we can test for determinism more efficiently if it is the case that every piece of information that is defined on some subset of a relation scheme is explicitly represented in the database. This is the property of the *embedded-complete* schemes [15]: a scheme $\mathbf{R}$ is *embedded-complete* if for any consistent state $\mathbf{r}$ of $\mathbf{R}$ and for any $X \subseteq U$ such that there exists $R_i(X_i) \in \mathbf{R}$ with $X_i \supseteq X$, it is the case that $\pi^{\downarrow}_X(\text{RI}_{\mathbf{r}}) = \bigcup_{X_j \supseteq X} \pi_X(r_j)$.

Let us consider the deletion of any tuple over a set of attributes $X \subseteq U$ from a state $\mathbf{r}$ of the scheme $\mathbf{R}$.

THEOREM 11. *The deletion of a tuple* t *from a state* $\mathbf{r}$ *on an embedded-complete scheme is deterministic if and only if there is a relational scheme* $R_i(X_i)$ *such that* $X_i \supseteq X$.

Proof. Only if. The proof follows by Lemma 5.

If. This part follows by Theorem 5 and the fact that if the database scheme is embedded-complete, then no tuple over a set of attributes which is contained in a relational scheme can be generated by the chase from other tuples, and so it is never the case that $t \widehat{\notin} \mathbf{r}_-$. □

Also in this case, it is possible to state local conditions at scheme level for independent schemes that allow us to test for determinism of a deletion of a tuple t over a set of attributes $X \subseteq U$ as follows.

CONDITION 3. *For every relation scheme* $R_i(X_i)$ *in* $\mathbf{R}$, *at least one of the following holds:*

(i) $X \not\subseteq X_i$,

(ii) *for every relation scheme* $R_j(X_j)$ *in* $\mathbf{R}$, $i \neq j$, $X \not\subseteq X_j^+$,

(iii) F_i *contains an FD whose left-hand side is a superkey of* X.

THEOREM 12. *Let* $\mathbf{R}$ *be independent and assume that Condition* 3 *holds. Then the deletion of* t *from* $\mathbf{r}$ *is deterministic if and only if there is a relational scheme* $R_i(X_i)$ *such that* $X \subseteq X_i$.

Proof. Only if. The proof is made by Lemma 5.

If. Assume by way of contradiction that, for an independent scheme $\mathbf{R}$, Condition 3 holds and that there is a relational scheme $R_i(X_i) \in \mathbf{R}$ such that $X \subseteq X_i$ but the deletion is not deterministic. By Theorem 5 it follows that $t \widehat{\in} \mathbf{r}_-$, and therefore there is a tuple t' in $\mathrm{RI}_{\mathbf{r}_-}$ such that $t'[X] = t$. Then let t_j be the tuple from which t' originates and assume that $t_j \in r_{-_j}$ over $R_j(X_j)$. Clearly, $X \not\subseteq X_j$ and so $i \neq j$. Moreover, since we have assumed that $t \widehat{\in} \mathbf{r}$, by Theorem 2, $X_j \rightarrow X \in F^+$, and so, $X \subseteq X_j^+$. This implies that there is an FD $Y \rightarrow A \in F_i$ such that Y is a superkey of X; otherwise Condition 3 would be false. Since $\mathbf{R}$ is independent, by Lemma 3, we have $\pi^{\downarrow}_{YA}(\mathrm{RI}_{\mathbf{r}_-}) = \pi_{YA}(r_{-_i})$, and since Y is a superkey of X, we have that $\pi^{\downarrow}_{X}(\mathrm{RI}_{\mathbf{r}_-}) = \pi_X(r_{-_i})$. But, by construction, $t \notin \pi_X(r_{-_i})$ and so it follows that $t \widehat{\notin} \mathbf{r}_-$—a contradiction. ∎

EXAMPLE 11. *Consider the deletion* $t = \langle \mathrm{CS}, \mathrm{Smith}, \mathrm{B} \rangle$ *over* $X = DMP$ *from the state* $\mathbf{r}$ *in Figure* 1. *Condition* 3 *does not hold since* $X \subseteq X_2 = DMP$, $X \subseteq X_1^+ = U$, *and the only FD* $D \rightarrow M$ *in* F_2 *has a left-hand side that is not a superkey of* X. *Therefore, for deletions defined on EDP we need to check whether* $t \notin \mathbf{r}_-$ *(it turns out that the deletion of* t *is not deterministic as shown in Example* 8*). Note that the scheme of* $\mathbf{r}$ *is not embedded-complete. Conversely, Condition* 3 *is verified for the tuple* $t' = \langle \mathrm{CS}, \mathrm{Smith} \rangle$ *over* DM *since we have* $DM \subseteq X_2$ *and* $DM \subseteq X_1^+$, *but in this case* D *is a superkey of* DM. *Hence, the deletion of* t' *from* $\mathbf{r}$ *is deterministic and the result of the deletion can be obtained by deleting the tuples* $\langle \mathrm{CS}, \mathrm{Smith}, \mathrm{A} \rangle$ *and* $\langle \mathrm{CS}, \mathrm{Smith}, \mathrm{B} \rangle$ *from* r_2.

Condition 3 can be tested for independent schemes in time bounded by $\|F\| \times (|\mathbf{R}|^2 + |F|)$ and can be performed after step (2) of Algorithm 5. If such a condition is verified, then the test for determinism in step (3) is no longer necessary, since in this case the state $\mathbf{r}_-$ is surely the maximum result.

7. Modification operations. The present paper studies insertions and deletions of tuples as basic database update operations. However, modifications form another important class of update operations, often used in practical situations. The goal of this section is to discuss briefly the modification of tuples: we show that they naturally fit in our framework and that, in general, insertion and deletion results can be used to characterize them.

A simple modification operation consists of changing the values of a single tuple. Therefore, we can represent a modification of a state $\mathbf{r}$ by means of a pair (t_{old}, t_{new}), where t_{old} and t_{new} are tuples defined over the same set of attributes $X \subseteq U$; the intended meaning of this operation is clearly to substitute t_{old} by t_{new} in $\mathbf{r}$.

According to the definition of insertions and deletions, this operation should be realized by altering the information content of the original state as little as possible. Thus, in the framework we have defined, a modification (t_{old}, t_{new}), defined over any set of attributes $X \subseteq U$, of a database state $\mathbf{r}$ for a scheme $\mathbf{R}$, can be defined through the following notion of result.

A state $\mathbf{r}_p$ is a *potential result* for the modification (t_{old}, t_{new}) to $\mathbf{r}$ if (1) $t_{old} \widehat{\notin} \mathbf{r}_p$, (2) $t_{new} \widehat{\in} \mathbf{r}_p$, and (3) for every state $\mathbf{r}' \preceq \mathbf{r}$ of $\mathbf{R}$ such that (a) $t_{old} \widehat{\notin} \mathbf{r}'$ and (b) the insertion of t_{new} in $\mathbf{r}'$ is consistent, $\mathbf{r}' \preceq \mathbf{r}_p$.

If we assume that $t_{old} \widehat{\in} \mathbf{r}$, a modification is always possible, but as with insertions, inconsistency and nondeterminism may arise. We then say that a modification is

consistent if it has a consistent potential result and is *deterministic* if the glb of the potential results is itself a potential result.

By the definition above, it turns out that a modification can be implemented, in most cases, through a deletion followed by an insertion. Specifically, we can easily show the following results

LEMMA 13. *Let* $\mathbf{r}$ *be a database state of a scheme* $\mathbf{R}$*, and* (t_{old}, t_{new}) *be a modification defined over a set of attributes* $X \subseteq U$*. Then the following properties hold:*

(i) *If the deletion of* t_{old} *from* $\mathbf{r}$ *is deterministic and the insertion of* t_{new} *to* $\mathbf{r}_{-t_{old}}$ *is consistent, then the modification* (t_{old}, t_{new}) *of* $\mathbf{r}$ *is consistent.*

(ii) *If the deletion of* t_{old} *from* $\mathbf{r}$ *is deterministic and the insertion of* t_{new} *to* $\mathbf{r}_{-t_{old}}$ *is deterministic, then the modification* (t_{old}, t_{new}) *of* $\mathbf{r}$ *is deterministic.*

(iii) *If the deletion of* t_{old} *from* $\mathbf{r}$ *is deterministic then* $(\mathbf{r}_{-t_{old}})_{+t_{new}}$ *is the glb of the potential results.*

We note that the converses of the above results do not hold in general. This is shown in the following example.

EXAMPLE 12. *Consider the database scheme* $\mathbf{R} = \{R_1(AB), R_2(BC)\}$*, with the FDs* $A \rightarrow B$ *and* $B \rightarrow C$ *defined for it, and the state of* $\mathbf{R}$*:*

$$\mathbf{r} = \{r_1 = \{\langle 1, 2\rangle\}, r_2 = \{\langle 2, 3\rangle\}\}.$$

Consider now the modification $(t_{old}, t_{new}) = (\langle 1, 2, 3\rangle, \langle 1, 2, 5\rangle)$ *defined over* ABC*. The deletion of* t_{old} *from* $\mathbf{r}$ *is not deterministic, since it can be realized either by deleting the tuple* $\langle 1, 2\rangle$ *from* r_1 *or the tuple* $\langle 2, 3\rangle$ *from* r_2*. However, we have that the modification is indeed deterministic. In fact, let* $\mathbf{r}'$ *and* $\mathbf{r}''$ *be the potential results for the deletion of* t_{old} *from* $\mathbf{r}$*. Then*

$$\mathbf{r}' = \{r'_1 = \emptyset, r'_2 = \{\langle 2, 3\rangle\}\} \text{ and } \mathbf{r}'' = \{r''_1 = \{\langle 1, 2\rangle\}, r''_2 = \emptyset\}.$$

Then, it is easy to see that the insertion of t_{new} *in* $\mathbf{r}'$ *is inconsistent, whereas the insertion of* t_{new} *in* $\mathbf{r}''$ *is consistent and deterministic. It follows that the glb of the potential results is indeed a potential result and can be obtained by substituting* $\langle 2, 5\rangle$ *for* $\langle 2, 3\rangle$ *in* r_2*.*

This example shows that in general, in order to implement a modification (t_{old}, t_{new}) of a state $\mathbf{r}$, we first have to find all the *maximal potential results* for the deletion of t_{old} from $\mathbf{r}$. We recall that a maximal potential result $\mathbf{r}_M$ for a deletion is a potential result for which there is no other potential result $\mathbf{r}_p$ such that $\mathbf{r}_M \preceq \mathbf{r}_p$ [9]. Then we have to select from among them the states for which the insertion of t_{new} is consistent and deterministic. Finally, the results for the insertion of t_{new} to the selected states have to be compared: if there is one that is weaker than each other, the modification is deterministic.

From the discussion above, it turns out that the algorithms derived for implementing insertions and deletions can be also used to implement modification operations.

8. Conclusions. In this paper we have shown that, similar to what has been done with respect to query answering, efficient algorithms for characterizing and performing update operations defined over any subset of the universe can be given for the highly significant class of independent schemes. In fact, the various characterizations, which require time and space polynomial with respect to the size of the database state [9], can be verified for this class of schemes efficiently, as in this case we can derive a restricted number of optimized relational expression that allow us to refer only to the relevant portion of the database.

In particular, with respect to insert operations, we have first shown that the property of consistency can be efficiently tested by considering only to the "extension" of the tuple to be inserted (which is obtained by adding to t further values derived from the original state and the constraints) and the involved FDs. With respect to the property of determinism, we have first provided an alternative method that does not require the construction of a special tableau over the whole database. This method refers to a state obtained from the original database by adding tuples that, for independent schemes, can be efficiently derived. If the insertion is deterministic, then this special state corresponds to the result of the insertion. We have then provided for both insertions and deletions practical algorithms implementing the various characterizations. We have finally shown that under some further conditions, update operations can be managed more easily.

Clearly, when nondeterministic updates arise, the system should not simply reject them but rather try to resolve these situations in some way. Indeed, this can be done in several ways since, in general, the problem is that some information for satisfying the request is missing and there are several possible choices for providing this extra information. For instance, potential ambiguities can be solved by means of a dialogue with the users, similar to the approach described in [8]. Therefore, the algorithms we have presented can be extended in several ways in order to try to resolve nondeterministic update operations.

New classes of database schemes, generalizing the class of independent schemes, have been introduced [14, 30]. Similar to the independent ones, these schemes enjoy the property that the consistency of a database state after a simple update to a base relation can be efficiently verified. Thus, it could be interesting to extend the results of this paper to these more general classes of schemes.

Acknowledgment. We would like to thank the anonymous referees for their very helpful comments and suggestions.

REFERENCES

[1] S. Abiteboul, *Updates, a new frontier*, in Second International Conference on Data Base Theory (ICDT'88), Bruges, Lecture Notes in Comput. Sci. 326, Springer-Verlag, New York, 1988, pp. 1–18.

[2] A. Aho, C. Beeri, and J. Ullman, *The theory of joins in relational databases*, ACM Trans. Database Syst., 4 (1979), pp. 297–314.

[3] P. Atzeni and E. Chan, *Efficient query answering in the representative instance approach*, in Proc. 4th ACM SIGACT SIGMOD Symp. on Principles of Database Systems, Portland, OR, 1985, pp. 181–188.

[4] P. Atzeni and E. Chan, *Efficient optimization of simple chase join expressions*, ACM Trans. Database Syst., 14 (1989), pp. 212–230.

[5] P. Atzeni and E. Chan, *Efficient and optimal query answering on independent schemes*, Theoret. Comput. Sci., 77 (1990), pp. 291–308.

[6] P. Atzeni and M. De Bernardis, *A new interpretation for null values in the weak instance model*, J. Comput. System Sci., 41 (1990), pp. 25–43.

[7] P. Atzeni and V. DeAntonellis, *Relational Database Theory: A Comprehensive Introduction*, Benjamin Cummings, Menlo Park, CA, 1993.

[8] P. Atzeni and R. Torlone, *Solving ambiguities in updating deductive databases*, in Mathematical Fundamentals of Data Base Systems (MFDBS'91), Rostock, Germany, Lecture Notes in Comput. Sci. 495, Springer-Verlag, New York, 1991, pp. 104–118.

[9] P. Atzeni and R. Torlone, *Updating relational databases through weak instance interfaces*, ACM Trans. Database Syst., 17 (1992), pp. 718–746.

[10] C. Beeri and P. Bernstein, *Computational problems related to the design of normal form relational schemas*, ACM Trans. Database Syst., 4 (1979), pp. 30–59.

[11] P. Bernstein, *Synthesizing third normal form relations from functional dependencies*, ACM Trans. Database Syst., 1 (1976), pp. 277–298.
[12] G. Birkhoff, *Lattice Theory*, 3rd ed., Colloq. Publ. XXV, AMS, Providence, RI, 1967.
[13] E. Chan, *Optimal computation of total projections with unions of simple chase join expressions*, in Proc. ACM SIGMOD International Conf. on Management of Data, Boston, MA, 1984, pp. 149–163.
[14] E. Chan and H. Hernandez, *Independence-reducible database schemes*, J. ACM, 38 (1991), pp. 856–886.
[15] E. Chan and A. Mendelzon, *Answering queries on embedded-complete database schemes*, J. ACM, 34 (1987), pp. 349–375.
[16] E. Chan and A. Mendelzon, *Independent and separable database schemes*, SIAM J. Comput., 16 (1987), pp. 841–851.
[17] M. Graham and M. Yannakakis, *Independent database schemas*, J. Comput. System Sci., 28 (1984), pp. 121–141.
[18] P. Honeyman, *Testing satisfaction of functional dependencies*, J. ACM, 29 (1982), pp. 668–677.
[19] M. Ito, M. Iwasaki, and T. Kasami, *Some results on the representative instance in relational databases*, SIAM J. Comput., 14 (1985), pp. 334–354.
[20] D. Maier, *The Theory of Relational Databases*, Computer Science Press, Potomac, MD, 1983.
[21] D. Maier, A. Mendelzon, and Y. Sagiv, *Testing implications of data dependencies*, ACM Trans. Database Syst., 4 (1979), pp. 455–468.
[22] D. Maier, D. Rozenshtein, and D. Warren, *Window functions*, in Advances in Computing Research, Vol. 3, P. Kanellakis and F. Preparata, eds., JAI Press, Greenwich, CT, 1986, pp. 213–246.
[23] D. Maier, J. Ullman, and M. Vardi, *On the foundations of the universal relation model*, ACM Trans. Database Syst., 9 (1984), pp. 283–308.
[24] A. Mendelzon, *Database states and their tableaux*, ACM Trans. Database Syst., 9 (1984), pp. 264–282.
[25] Y. Sagiv, *Can we use the universal instance assumption without using nulls?*, in Proc. ACM SIGMOD International Conf. on Management of Data, Ann Arbor, MI, 1981, pp. 108–120.
[26] Y. Sagiv, *A characterization of globally consistent databases and their correct access paths*, ACM Trans. Database Syst., 8 (1983), pp. 266–286.
[27] Y. Sagiv, *On computing restricted projections of the representative instance*, in Proc. 4th ACM SIGACT SIGMOD Symp. on Principles of Database Systems, Portland, OR, 1985, pp. 173–180.
[28] Y. Sagiv, *Evaluation of queries in independent database schemes*, J. ACM, 38 (1991), pp. 120–161.
[29] J. Ullman, *Principles of Database Systems*, 2nd ed., Computer Science Press, Potomac, MD, 1982.
[30] K. Wang and M. Graham, *Constant-time maintainability: A generalization of independence*, ACM Trans. Database Syst., 17 (1992), pp. 201–246.
[31] M. Yannakakis, *Querying weak instances*, in Advances in Computing Research, Vol. 3, P. Kanellakis and F. Preparata, eds., JAI Press, Greenwich, CT, 1986, pp. 185–211.

SIAM J. COMPUT.
Vol. 28, No. 3, pp. 1136–1153

LEARNING DNF OVER THE UNIFORM DISTRIBUTION USING A QUANTUM EXAMPLE ORACLE*

NADER H. BSHOUTY† AND JEFFREY C. JACKSON‡

Abstract. We generalize the notion of probably approximately correct (PAC) learning from an example oracle to a notion of efficient learning on a quantum computer using a quantum example oracle. This quantum example oracle is a natural extension of the traditional PAC example oracle, and it immediately follows that all PAC-learnable function classes are learnable in the quantum model. Furthermore, we obtain positive quantum learning results for classes that are not known to be PAC learnable. Specifically, we show that disjunctive normal form (DNF) is efficiently learnable with respect to the uniform distribution by a quantum algorithm using a quantum example oracle. While it was already known that DNF is uniform-learnable using a membership oracle, we prove that a quantum example oracle with respect to uniform is less powerful than a membership oracle.

Key words. quantum example oracle, quantum computing, disjunctive normal form (DNF), machine learning, Fourier transform

AMS subject classifications. 68Q20, 68Q05

PII. S0097539795293123

1. Introduction.

Recently, there has been significant interest in the extent to which quantum physical effects can be used to solve problems that appear to be computationally difficult, using traditional methods [16, 8, 7, 6, 33, 31, 30]. In this paper, we apply quantum methods to questions in computational learning theory. In particular, we focus on the problem of learning—from examples alone—the class DNF of polynomial-size disjunctive normal form expressions.

The DNF learning problem has a long history. Valiant [32] introduced the problem and gave efficient algorithms for learning certain subclasses of DNF. Since then, learning algorithms have been developed for a number of other subclasses of DNF [25, 4, 2, 21, 3, 1, 11, 27, 13, 10] and recently for the unrestricted class of DNF expressions [22], but almost all of these results—in particular the results for the unrestricted class—use membership queries. (The learner is told the output value of the target function on learner-specified inputs.) While Angluin and Kharitonov [5] have shown that if DNF is PAC learnable in a distribution-independent sense (definitions are given in the next section) with membership queries, then it is learnable with respect to polynomial-time computable distributions without these queries, the question of the extent to which membership queries are necessary for distribution-dependent DNF learning is still open. In particular, Jackson's harmonic sieve [22] uses membership queries to learn DNF with respect to the uniform distribution. Can DNF be learned with respect to uniform from a weaker form of oracle?

We show that DNF is efficiently learnable with respect to the uniform distribution by a quantum algorithm that receives its information about the target function from a *quantum example oracle*. This oracle generalizes the traditional PAC example oracle

*Received by the editors October 12, 1995; accepted for publication (in revised form) September 22, 1997; published electronically February 19, 1999.
http://www.siam.org/journals/sicomp/28-3/29312.html

†Department of Computer Science, University of Calgary, Calgary, AB, T2N 1N4 Canada (bshouty@cpsc.ucalgary.ca). The work of this author was supported by NSERC.

‡Department of Mathematics and Computer Science, Duquesne University, Pittsburgh, PA 15282. The work of this author was sponsored by the National Science Foundation under grant CCR-9119319 (jackson@mathcs.duq.edu).

in a natural way. Specifically, the quantum oracle $QEX(f, D)$ is a traditional PAC example oracle $EX(f, D)$ except that $QEX(f, D)$ produces the example $\langle x, f(x)\rangle$ with amplitude $\sqrt{D(x)}$ rather than with probability $D(x)$. We also show that, with respect to the uniform distribution, a quantum example oracle can be simulated by a membership oracle but not vice versa.

To obtain our quantum DNF learning algorithm, we modify the harmonic sieve algorithm (HS) for learning DNF with respect to uniform using membership queries [22]. In fact, HS properly learns the larger class $\widehat{PT}_1$ of functions expressible as a threshold of a polynomial number of parity functions, and our algorithm properly learns this class as well. The harmonic sieve uses membership queries to locate parity functions that correlate well with the target function with respect to various near-uniform distributions. The heart of our result is showing that these parities can be located efficiently by a quantum algorithm, using only a quantum example oracle.

Our primary result, then, is to show that DNF is quantum-learnable, using an oracle that is strictly weaker than a membership oracle. Our algorithm also possesses a somewhat better asymptotic bound on running time than the harmonic sieve. We consider the potential significance of these results in more detail in the concluding section.

2. Definitions and notation.

2.1. Functions and function classes. We will be interested in the learnability of sets (*classes*) of Boolean functions over $\{0,1\}^n$ for fixed positive values of n. It will be convenient to use different definitions for "Boolean" in different contexts within this paper. In particular, at times we will think of a Boolean function as mapping to $\{0,1\}$ and at other times to $\{-1,+1\}$; the choice will either be indicated explicitly or clear from context. We call $\{0,1\}^n$ the *instance space* of a Boolean function f, an element x in the instance space an *instance*, and the pair $\langle x, f(x)\rangle$ an *example* of f. We denote by x_i the ith bit of instance x.

Intuitively, a learning algorithm should be allowed to run in time polynomial in the complexity of the function f to be learned; we will use the *size* of a function as a measure of its complexity. The size measure will depend on the function class to be learned. In particular, each function class $\mathcal{F}$ that we study implicitly defines a natural class $\mathcal{R}_{\mathcal{F}}$ of representations of the functions in $\mathcal{F}$. We define the *size of a function* $f \in \mathcal{F}$ as the minimum, over all $r \in \mathcal{R}_{\mathcal{F}}$ such that r represents f, of the size of r, and we define below the size measure for each representation class of interest.

A DNF expression is a disjunction of *terms*, where each term is a conjunction of *literals* and a literal is either a variable or its negation. The *size of a DNF expression* r is the number of terms in r. The *DNF function class* is the set of all functions that can be represented as a DNF expression of size polynomial in n.

Following Bruck [12], we use $\widehat{PT}_1$ to denote the class of functions on $\{0,1\}^n$ expressible as a depth-2 circuit with a majority gate at the root and polynomially many parity gates at the leaves. All gates have unbounded fanin and fanout 1. The *size of a* $\widehat{PT}_1$ *circuit* r is the number of parity gates in r.

2.2. Quantum turing machines. We now review the model of quantum computation defined by Bernstein and Vazirani [6]. First we define how the specification (program) of a *quantum Turing machine* (QTM) is written down. Then we describe how a QTM operates.

The specification of a QTM is almost exactly the same as the specification of a probabilistic TM (PTM). Recall that the transition table of a PTM specifies, for each state and input symbol, a set of moves—a move is a next state, new tape symbol, and head movement direction—along with associated probabilities that each move will be chosen. Of course, these probabilities must be nonnegative and sum to 1. In a QTM specification, the transition probabilities between PTM configurations are replaced with complex-valued numbers (*amplitudes*) that satisfy a certain *well-formedness* property. Loosely speaking, if the sum of the squares of the amplitudes for the transitions corresponding to each state/symbol pair is 1, then the QTM satisfies the well-formedness property. A somewhat peculiar aspect of amplitudes is that they may have negative real components, unlike the probabilities of the PTM model. Formally, we define well-formedness as follows. For a QTM M, let R_M be the (infinite-dimensional) matrix where each row and each column is labeled with a distinct machine configuration (c_r and c_c, respectively) and each entry in R_M is the amplitude assigned by M to the transition from configuration c_c to c_r. Then M satisfies the well-formedness property if R_M is unitary ($R_M^\dagger R_M = R_M R_M^\dagger = I$, where $R_M^\dagger$ is the transpose conjugate of R_M). A QTM specification also contains a set of states (including all of the final states) in which an *Obs* operation is performed; we define this operation below.

To describe the operation of a QTM, we use the notion of a *superposition* of configurations. For example, consider a probabilistic Turing machine M' that at step i flips a fair coin and chooses to transition to one of two configurations c_1 and c_2. While we would generally think of M' as being in exactly one of these configurations at step $i+1$, we can equivalently think of M' as being in *both* states, each with probability $1/2$. Continuing in this fashion, for each step until M' terminates we can think of M' as being in a superposition of states, each state with an associated probability. After M' takes its final step, each of its final states will have some associated probability (we assume without loss of generality that all computation paths in M' have the same length). If M' now "chooses" to be in one of these final states σ_f randomly according to the induced probability distribution on final states, then the probability of being in σ_f is exactly the same in this model as it is in the traditional PTM model.

In summary, we can view a PTM M' as being in a superposition of configurations at each step, where a superposition is represented by a vector of probabilities, one for each possible configuration of M'. Likewise, we view a QTM M as being in a superposition of configurations at each step, but now the superposition vector contains an amplitude for each possible configuration of M. The initial superposition vector in both cases is the all-zero vector except for a single 1 in the position corresponding to the initial configuration of the machine. Note that each step of a PTM M' can be accomplished by multiplying the current superposition vector by a matrix $R_{M'}$, which is defined analogously with R_M above. In the same way, each step of a QTM M is accomplished by multiplying its current superposition vector by R_M. The difference between the machines comes at the point(s) where M "chooses" to be in a single configuration rather than in a superposition of configurations. M does this (conceptually) by transitioning to a superposition of configurations all of which are in one of the *Obs* states mentioned above. The superposition vector is then changed so that a single configuration has amplitude 1 and all others are 0. This is exactly analogous to the PTM M' choosing its final state, except that the probability of choosing each configuration c_i is now the *square* of the magnitude of the amplitude associated with c_i in M's current superposition vector.

The notation

$$\sum_x a_x|x\rangle$$

denotes a superposition of configurations x, each having amplitude a_x. While in general this sum is over all possible configurations of the QTM, when we use this notation it will be the case (unless otherwise noted) that all of the configurations having nonzero amplitude are in the same state and have the tape head at the same position. In this case, the configurations x are distinguished only by their tape contents, so we will treat x as if it is merely the tape content and ignore the other configuration parameters. We will also assume that all tapes contain the same number of nonblank characters unless otherwise noted.

Given this notation, we now more formally define the *Obs* operation. Intuitively, an *Obs* will collapse a superposition S to one of two possible superpositions, S_0 or S_1, with the choice of superposition based on a probability that is a function of the value of the first bit of the tape (we are implicitly assuming that when an *Obs* is performed the configurations in a superposition differ only in terms of what is on the respective tapes, which will be sufficient for our purposes). Specifically, let $b \in \{0, 1\}$. Then

$$Obs\left(\sum_{bx} a_{bx}|bx\rangle\right)$$

$$= \begin{cases} \sum_x \frac{a_{0x}}{\sum_y |a_{0y}|^2}|0x\rangle & \text{with probability } \sum_x |a_{0x}|^2, \\ \sum_x \frac{a_{1x}}{\sum_y |a_{1y}|^2}|1x\rangle & \text{with probability } \sum_x |a_{1x}|^2. \end{cases}$$

Note that by permuting bits of the tape and performing successive *Obs* operations we can simulate the informal definition of *Obs* given earlier. We say that a language L is in BQP if there exists a QTM M such that, at the end of a number of steps by M polynomial in the length of the input to M, an *Obs* fixes the first tape cell to 1 with probability at least 2/3 if the input is in L and fixes it to 0 with probability at least 2/3 otherwise. We will also sometimes think of an *Obs* as simply computing the probability that the first cell will be fixed to 1.

Finally, we will at times want to introduce deterministic transitions into our QTMs while preserving the well-formedness property of the transition matrices. It can be shown [15] that as long as a deterministic computation is *reversible*, then the computation can be carried out on a QTM. Informally, a computation is reversible if given the result of the computation it is possible to determine the input to the computation. (See [6] for a formal definition and discussion of reversibility in the context of quantum computation.)

2.3. Standard learning models. We begin by defining the well-known PAC learning model and then generalize this to a quantum model of learning. First, we define several supporting concepts. Given a function f and probability distribution D on the instance space of f, we say that the Boolean function h having as its domain the instance space of f is an *ϵ-approximator for f with respect to D* if $\Pr_D[h = f] \geq 1 - \epsilon$. An *example oracle for f with respect to D* ($EX(f, D)$) is an oracle that on request draws an instance x at random according to probability distribution D and returns the example $\langle x, f(x)\rangle$. A *membership oracle for f* ($MEM(f)$) is an oracle that given

any instance x returns the value $f(x)$. Let $\mathcal{D}_n$ denote a nonempty set of probability distributions on $\{0,1\}^n$. Any set $\mathcal{D} = \cup_n \mathcal{D}_n$ is called a *distribution class*. We let $\mathcal{U}_n$ represent the uniform distribution on $\{0,1\}^n$ and call $\mathcal{U} = \cup_n \mathcal{U}_n$ simply the *uniform distribution.*

Now we formally define the probably approximately correct (PAC) model of learnability [32]. Let ϵ and δ be positive values (called the *accuracy* and *confidence* of the learning procedure, respectively). Then we say that the function class $\mathcal{F}$ is *(strongly) PAC learnable* if there is an algorithm $\mathcal{A}$ such that for any ϵ and δ, any $f \in \mathcal{F}$ (the *target function*), and any distribution D on the instance space of f (the *target distribution*), with probability at least $1 - \delta$ algorithm $\mathcal{A}(EX(f,D), \epsilon, \delta)$ produces an ϵ-approximation for f with respect to D in time polynomial in n, the size of f, $1/\epsilon$, and $1/\delta$. The probability that $\mathcal{A}$ succeeds is taken over the random choices made by EX and $\mathcal{A}$ (if any). We generally drop the "PAC" from "PAC learnable" when the model of learning is clear from context.

We will consider several variations on the basic learning models. Let $\mathcal{M}$ be any model of learning (e.g., PAC). If $\mathcal{F}$ is $\mathcal{M}$-learnable by an algorithm $\mathcal{A}$ that requires a membership oracle, then $\mathcal{F}$ is *$\mathcal{M}$-learnable, using membership queries*. If $\mathcal{F}$ is $\mathcal{M}$-learnable for $\epsilon = 1/2 - 1/p(n,s)$, where p is a fixed polynomial and s is the size of f, then $\mathcal{F}$ is *weakly $\mathcal{M}$-learnable*. We say that $\mathcal{F}$ *is $\mathcal{M}$-learnable by $\mathcal{H}$* if $\mathcal{F}$ is $\mathcal{M}$-learnable by an algorithm $\mathcal{A}$ that always outputs a function $h \in \mathcal{H}$. If $\mathcal{F}$ is $\mathcal{M}$-learnable by $\mathcal{F}$, then we say that $\mathcal{F}$ is *properly* $\mathcal{M}$-learnable. Finally, note that the PAC model places no restriction on the example distribution D; we sometimes refer to such learning models as *distribution-independent*. If $\mathcal{F}$ is $\mathcal{M}$-learnable for all distributions D in distribution class $\mathcal{D}$ then $\mathcal{F}$ is *$\mathcal{M}$-learnable with respect to $\mathcal{D}$*. Learning models which place a restriction on the distributions learned against we call *distribution-dependent* models.

2.4. The Fourier transform. We will make substantial use of the discrete Fourier transform in our analysis; this approach was introduced in machine learning by Linial, Mansour, and Nisan [28]. In this section we give some basic definitions and standard theorems.

For each bit vector $a \in \{0,1\}^n$ we define the function $\chi_a : \{0,1\}^n \to \{-1,+1\}$ as

$$\chi_a(x) = (-1)^{\sum_{i=1}^n a_i x_i} = 1 - 2\left(\sum_{i=1}^n a_i x_i \bmod 2\right).$$

That is, $\chi_a(x)$ is the Boolean function that is 1 when the parity of the bits in x indexed by a is even and is -1 otherwise. With inner product defined by[1] $\langle f, g\rangle = \mathbf{E}_x[f(x)\cdot g(x)] \equiv \mathbf{E}[fg]$ and norm defined by $\|f\| = \sqrt{\mathbf{E}[f^2]}$, $\{\chi_a \mid a \in \{0,1\}^n\}$ is an orthonormal basis for the vector space of real-valued functions on the Boolean cube $\mathbf{Z}_2^n$. That is, every function $f : \{0,1\}^n \to \mathbf{R}$ can be uniquely expressed as a linear combination of parity functions:

$$f = \sum_{a\in\{0,1\}^n} \hat{f}(a)\chi_a,$$

where $\hat{f}(a) = \mathbf{E}[f\chi_a]$. We call the vector of coefficients $\hat{f}$ the *Fourier transform* of f. Note that for f mapping to $\{-1,+1\}$, $\hat{f}(a)$ represents the correlation of f and χ_a

[1] Expectations and probabilities here and elsewhere are with respect to the uniform distribution over the instance space unless otherwise indicated.

with respect to the uniform distribution. Also, let 0^n represent the vector of n zeros. Then $\hat{f}(0^n) = \mathbf{E}[f\chi_{0^n}] = \mathbf{E}[f]$, since χ_{0^n} is the constant function $+1$.

By Parseval's identity, for every real-valued function f, $\mathbf{E}[f^2] = \sum_{a\in\{0,1\}^n} \hat{f}^2(a)$. For f mapping to $\{-1,+1\}$ it follows that $\sum_a \hat{f}^2(a) = 1$. More generally, it can be shown that for any real-valued functions f and g, $\mathbf{E}[fg] = \sum_a \hat{f}(a)\hat{g}(a)$.

3. The quantum example oracle. While it has been shown that DNF is learnable with respect to the uniform distribution if membership queries are available to the learning algorithm [22], it is desirable to have a DNF learning algorithm that can learn from examples alone. This seems to be a hard problem, using conventional computing paradigms. However, other problems—such as integer factorization—which had seemed to be hard have recently been shown to have efficient quantum solutions [30]. Thus it is natural to ask the following question: Is there a QTM M such that, given access to a traditional PAC example oracle $EX(f,D)$ for any function f in DNF, M efficiently learns an ϵ-approximation to f?

In this paper, we consider a related question that we hope may shed some light on the question posed above. Specifically, we consider the question of learning DNF from a *quantum example oracle*. This oracle, which we define below, generalizes the PAC example oracle to the quantum setting in a natural way. It should be noted that questions about the power of quantum computing relative to oracles has been investigated previously; for example, Berthiaume and Brassard [8, 7] consider the relative abilities of quantum and more traditional Turing machines to answer decision questions relative to a membership oracle.

We now define the quantum example oracle. Note that each call to the traditional PAC example oracle $EX(f,D)$ can be viewed as defining a superposition of 2^n configurations, each containing a distinct $\langle x, f(x)\rangle$ pair and having probability of occurrence $D(x)$. We generalize this to the quantum setting in a natural way. A *quantum example oracle for f with respect to D* ($QEX(f,D)$) is an oracle running coherently with a QTM M that changes M's tape $|y\rangle$ to

$$\sum_x \sqrt{D(x)}|y, x, f(x)\rangle.$$

That is, QEX defines a superposition of 2^n configurations much as EX does, but QEX assigns each configuration an amplitude $\sqrt{D(x)}$. As with the *Obs* operation, calls to QEX will be invoked by transitioning into designated states of M, and we will assume that any valid QTM program has the property that at each step either all of the states with nonzero amplitude in a superposition call QEX or none do. Note that for any f and D, a call to $QEX(f,D)$ followed by an appropriate *Obs* operation is equivalent to a call to $EX(f,D)$. We say that $\mathcal{F}$ is *quantum learnable* if $\mathcal{F}$ is PAC learnable by a QTM M using a quantum example oracle. Because every efficient TM computation can be simulated efficiently by a QTM [6] and because EX can be simulated by QEX, we have that every PAC-learnable function class is also quantum learnable.

For both standard and quantum example oracles, we use $EX(f)$ (respectively, $QEX(f)$) to represent learning with respect to the uniform distribution, that is, $EX(f,\mathcal{U}_\setminus)$ (respectively, $QEX(f,\mathcal{U}_\setminus)$).

4. Learning DNF from a membership oracle. In the next section we present our primary result, that DNF is quantum learnable with respect to the uniform distribution. Our result builds on the harmonic sieve (`HS`) algorithm for learning DNF with

respect to the uniform distribution using membership queries [22]. In this section we briefly review the harmonic sieve.

4.1. Overview of HS. The HS algorithm depends on a key fact about DNF expressions: for every DNF f and distribution D there is a parity χ_a that is a weak approximator to f with respect to D [22]. Also, for any function weakly approximable with respect to uniform by a parity function, a technique originally due to Goldreich and Levin [20] and first applied within a learning algorithm (KM) by Kushilevitz and Mansour [26] can be used to *find* such a parity (using membership queries). Combining these two facts gives that the KM algorithm weakly learns DNF with respect to uniform [9].

An obvious method to consider for turning this weak learner into a strong learner is some form of hypothesis boosting [29, 18, 17, 19]. In fact, HS is based on a particularly simple and efficient version of boosting discovered by Freund [18]. Each stage i of Freund's boosting algorithm explicitly defines a distribution D_i and calls on a weak learner to produce a weak approximator with respect to D_i. Distribution D_i is defined in terms of the performance of the weak hypotheses produced at the preceding boosting stages and in terms of the target distribution D. After a polynomial number of stages, a majority vote over the weak hypotheses gives a strong approximator with respect to D.

So in order to strongly learn DNF with respect to a distribution D, all that is needed is an efficient algorithm for weakly learning DNF with respect to the set of distributions $\{D_i\}$ defined by Freund's algorithm in the process of boosting with respect to D. While the question of whether or not such an algorithm exists for arbitrary D is an open problem, it is known that when boosting with respect to *uniform* a modified version of the KM algorithm can efficiently find a weakly approximating parity for any DNF f with respect to any of the distributions D_i defined by Freund's algorithm [22]. When learning with respect to such a distribution D_i, the modified KM algorithm must be given not only a membership oracle for f but also an oracle for the distribution D_i, that is, a function which given an instance x returns the weight that D_i places on x. Because Freund's booster explicitly defines the distribution D_i at each boosting stage i, an oracle for each D_i can be simulated and therefore the modified KM algorithm can be boosted by a modified version of Freund's algorithm that supplies D_i to the weak learner at each stage i. This then gives a strong learning algorithm that uses membership queries to learn DNF with respect to the uniform distribution.

4.2. Algorithmic details. The HS algorithm and its primary subroutine WDNF (the modified KM) are sketched in somewhat more detail in Figures 4.1 and 4.2. We will assume here and elsewhere that the number of terms s in the target function's representation as a DNF is known. This assumption can be relaxed by placing a standard guess-and-double loop around the body of the HS program (see, e.g., [24] for details); this increases the running time of the algorithm by at most a factor of $\log s$. The HS algorithm runs for $O(s^2 \log(1/\epsilon))$ stages. At each stage i, $r_i(x)$ (line 8) represents the number of weak hypotheses w_j among those hypotheses produced before stage i that are "right" on x. For uniform target distribution, the distributions D_i defined by Freund's booster are given by

$$D_i(x) = \frac{\alpha^i_{r_i(x)}}{\sum_y \alpha^i_{r_i(y)}}, \tag{4.1}$$

Invocation: $h \leftarrow \texttt{HS}(n, s, MEM(f), \epsilon, \delta)$
Input: n; s = size of DNF f; $MEM(f)$; $\epsilon > 0$; $\delta > 0$
Output: with probability at least $1-\delta$ (over random choices made by HS), HS returns h such that $\Pr[f = h] \geq 1 - \epsilon$

1. $\gamma \leftarrow 1/(8s+4)$
2. $k \leftarrow \frac{1}{2}\gamma^{-2}\ln(4/\epsilon)$
3. $w_0 \leftarrow \texttt{WDNF}(n, s, MEM(f), \mathcal{U}_n, \delta/2k)$
4. **for** $i \leftarrow 1, \ldots, k-1$ **do**
5. $\quad B(j; n, p) \equiv \binom{n}{j} p^j (1-p)^{n-j}$
6. $\quad \beta_r^i \equiv B(\lfloor k/2 \rfloor - r; k-i-1, 1/2+\gamma)$ if $i - k/2 < r \leq k/2$, $\beta_r^i \equiv 0$ otherwise
7. $\quad \alpha_r^i \equiv \beta_r^i / \max_{r=0,\ldots,i-1}\{\beta_r^i\}$.
8. $\quad r_i(x) \equiv |\{0 \leq j < i \mid w_j(x) = f(x)\}|$
9. $\quad \Theta \equiv c_2 \epsilon^3$
10. $\quad E_\alpha \leftarrow \textbf{Est}(\mathbf{E}_x[\alpha^i_{r_i(x)}], EX(f), \Theta/3, \delta/2k)$
11. $\quad$ **if** $E_\alpha \leq 2\Theta/3$ **then**
12. $\quad\quad k \leftarrow i$
13. $\quad\quad$ **break do**
14. $\quad$ **endif**
15. $\quad \tilde{D}_i(x) \equiv \alpha^i_{r_i(x)} / 2^n E_\alpha$
16. $\quad w_i \leftarrow \texttt{WDNF}(n, s, MEM(f), \tilde{D}_i(x), \delta/2k)$
17. **enddo**
18. $h(x) \equiv MAJ(w_0(x), w_1(x), \ldots, w_{k-1}(x))$
19. **return** h

FIG. 4.1. *Harmonic sieve* (HS) *algorithm for learning DNF from a membership oracle.* **Est**$(E, EX(f), \epsilon, \delta)$ *uses random sampling from* $EX(f)$ *to efficiently estimate the value of* E*, producing a value within an additive factor of* ϵ *of the true value with probability at least* $1-\delta$*.* c_2 *represents a fixed constant (*$1/57$ *is sufficient).*

Invocation: $w_i \leftarrow \texttt{WDNF}(n, s, MEM(f), cD, \delta)$
Input: n; s = size of DNF f; $MEM(f)$; cD, an oracle that given x returns $c \cdot D(x)$, where c is a constant in $[1/2, 3/2]$ and D is a probability distribution on $\{0,1\}^n$; $\delta > 0$
Output: with probability at least $1-\delta$ (over random choices made by WDNF), WDNF returns h such that $\Pr_D[f = h] \geq 1/2 + 1/(8s+4)$

1. $g(x) \equiv 2^n f(x) c D(x)$
2. find (using membership queries and with probability at least $1-\delta$) χ_a such that $|\mathbf{E}[g\chi_a]| \geq c/(4s+2)$
3. $h(x) \equiv \text{sign}(\mathbf{E}[g\chi_a]) \cdot \chi_a(x)$
4. **return** h

FIG. 4.2. WDNF *subroutine called by* HS.

where $\alpha_{r_i(x)}$ is defined in Figure 4.1. Note that while D_i is explicitly defined, it is not computationally feasible to compute D_i exactly because of the sum over an exponential number of terms in the denominator of (4.1). However, by a Chernoff bound argument this sum, and therefore D_i, can be closely approximated in time polynomial in the standard parameters. The function $\tilde{D}_i$ is HS's approximation to D_i. Note that because of the bound on the variable E_α and the accuracy with which E_α estimates $\mathbf{E}_x[\alpha^i_{r_i(x)}]$, with probability at least $1 - \delta/2k$, $\mathbf{E}_x[\alpha^i_{r_i(x)}] = c_3 E_\alpha$ for fixed $c_3 \in [1/2, 3/2]$. With the same probability, then, $\tilde{D}_i(x) = c_3 D_i(x)$ for all x.

Therefore, while we show WDNF in Figure 4.1 being called with an argument $\tilde{D}_i(x)$, the corresponding parameter in Figure 4.2 is cD, an oracle representing the product of a constant and a probability distribution. This oracle, along with the membership oracle for the target f, is used by WDNF to simulate an oracle g. We have omitted the details of line 2 of WDNF because this is the main point at which our quantum algorithm will differ from HS. Rather than using membership queries to locate the required parity χ_a, the new algorithm will use a quantum example oracle. Both algorithms depend on a key fact about DNF expressions [22]: for every DNF f with s terms and for every distribution D there exists a parity χ_a such that

$$|\mathbf{E}_D[f\chi_a]| \geq \frac{1}{2s+1}.$$

It follows from the definition of expectation that for either $h = \chi_a$ or $h = -\chi_a$

$$\mathbf{Pr}_D[f = h] \geq \frac{1}{2} + \frac{1}{4s+2}.$$

The WDNF algorithm can only guarantee to find a parity which is nearly optimally correlated with the target, which is why another factor of two is given up by the algorithm. Finally, WDNF also relies on the easily verified fact that for $g(x) = 2^n f(x)cD(x)$, $\mathbf{E}[g\chi_a] = c\mathbf{E}_D[f\chi_a]$. In essence, by combining the target f and distribution D we create a function g with the property that the large Fourier coefficients of g correspond to parities that are weak approximators to f with respect to D. This is why the hypothesis returned by WDNF is appropriate.

The version of WDNF modified to utilize a quantum example oracle is based on a similar idea of learning with respect to uniform in order to find a weak hypothesis with respect to a nonuniform distribution, but the implementation of the idea is quite different. We develop the modified algorithm in detail in the next section.

5. Learning DNF from a quantum example oracle. In this section we show how to modify the harmonic sieve in order to uniform-learn DNF using a quantum example oracle rather than a membership oracle. First, consider the call to WDNF at line 16 of HS for a fixed i, and for notational convenience let $D \equiv D_i$ and $\alpha(x) \equiv \alpha^i_{r_i(x)}$. Then, given the discussion concerning $\tilde{D}_i$ above and the definition of $\tilde{D}_i$ at line 15 in Figure 4.1, with high probability there is a $c_3 \in [1/2, 3/2]$ such that for all χ_a

$$\sum_x f(x)\chi_a(x)\tilde{D}_i(x) = c_3\mathbf{E}_D[f\chi_a] = \mathbf{E}[\alpha f\chi_a]/E_\alpha.$$

Also, again with high probability, the call to **Est** at line 10 is successful in estimating $\mathbf{E}[\alpha]$ to within the desired accuracy and therefore $E_\alpha \geq 2\Theta/3$. Applying this observation to the equation above and invoking the key DNF fact cited earlier gives that, with high probability, each time WDNF is called there exists some χ_a such that

$$|\mathbf{E}[\alpha f\chi_a]| \geq \frac{\Theta}{3(2s+1)} = \frac{c_2\epsilon^3}{3(2s+1)}, \tag{5.1}$$

and this χ_a (or its inverse) is a $(1/(4s+2))$-approximator to f. Our goal will be to find such a χ_a, or at least one that is nearly as good an approximator, using only a quantum example oracle for f.

Conceptually, to find such a χ_a we will repeatedly run a quantum subprogram that randomly selects one χ_a each time the subprogram runs. On any given run each

χ_a is selected with probability proportional to $\mathbf{E}^2[\alpha f \chi_a]$. The technique we use to perform this random sampling from the set of χ_a's is similar to a quantum algorithm of Bernstein and Vazirani that samples the χ_a's with probability $\hat{f}^2(a) = \mathbf{E}^2[f\chi_a]$. However, there are two difficulties with using their technique directly. First, their algorithm uses calls to the function f (membership queries), and we want an algorithm that uses only quantum example queries. Second, their technique works for Boolean ($\{-1,+1\}$-valued) functions, but the pairwise product αf, viewed as representing a single function, is clearly not Boolean in general. We address each of these difficulties in turn below.

5.1. Randomly selecting parities using a quantum example oracle. Our first step in modifying `WDNF` to learn from a quantum example oracle for Boolean f—$QEX(f)$—rather than from a membership oracle is to show that we can randomly select parity functions with probability proportional to $\hat{f}^2$, using only $QEX(f)$. The proof of the following lemma presents the required algorithm `QSAMP`.

LEMMA 1. *There is a quantum program* `QSAMP` *that, given any quantum example oracle $QEX(f)$ for $f : \{0,1\}^n \to \{-1,+1\}$, returns χ_a with probability $\hat{f}^2(a)/2$.*

Proof. `QSAMP` begins by calling $QEX(f)$ on a blank tape to get the superposition

$$\frac{1}{2^{n/2}} \sum_x |x, f(x)\rangle.$$

`QSAMP` next replaces $f(x)$ with $(1-f(x))/2$ (call this $f'(x)$); note that $(-1)^{f'(x)} = f(x)$. Then we will apply a Fourier operator F to the entire tape contents. We define F as

$$F(|a\rangle) \equiv \frac{1}{2^{n/2}} \sum_y (-1)^{a \cdot y} |y\rangle,$$

where $|a| = |y| = n$. This operation can be performed in n steps by a quantum Turing machine [6]. Also recall that $(-1)^{a \cdot y} \equiv \chi_a(y) \equiv \chi_y(a)$. Thus applying F gives us

$$\begin{aligned} F\left(\frac{1}{2^{n/2}} \sum_x |x, f'(x)\rangle\right) &= \frac{1}{2^{n/2}} \sum_x F(|x, f'(x)\rangle) \\ &= \frac{1}{2^{n+1/2}} \sum_{x,y,z} (-1)^{x \cdot y} (-1)^{f'(x)z} |y, z\rangle \\ &= \frac{1}{\sqrt{2}} \sum_y \mathbf{E}_x[\chi_y(x) f(x)] |y, 1\rangle \\ &\quad + \frac{1}{\sqrt{2}} \sum_y \mathbf{E}_x[\chi_y(x)] |y, 0\rangle \\ &= \frac{1}{\sqrt{2}} \sum_y \hat{f}(y) |y, 1\rangle + \frac{1}{\sqrt{2}} |\bar{0}, 0\rangle, \end{aligned}$$

where $|y| = |x| = n$ and $|z| = 1$ and the final line follows by orthonormality of the parity basis. An *Obs* operation at this point produces $|y, 1\rangle$ with probability $\hat{f}^2(y)/2$, as desired. ∎

5.2. Sampling parity according to coefficients of non-Boolean functions. While algorithm `QSAMP` is a good first step toward relaxing `HS`'s requirement for a

Invocation: $h \leftarrow$ HS'$(n, s, QEX(f), \epsilon, \delta)$
Input: n; s = size of DNF f; quantum example oracle $QEX(f)$; $\epsilon > 0$; $\delta > 0$
Output: with probability at least $1 - \delta$ (over random choices made by HS$'$), HS$'$ returns h such that $\Pr[f = h] \geq 1 - \epsilon$

1. γ, k, α_r^i, $r_i(x)$, Θ, $\tilde{D}_i(x)$ are defined as in HS
2. $w_0 \leftarrow$ WDNF'$(n, s, MEM(f), \mathcal{U}_n, \delta/2k)$
3. **for** $i \leftarrow 1, \ldots, k-1$ **do**
4. $\quad EX(f) \equiv Obs(QEX(f))$
5. $\quad E_\alpha \leftarrow \mathbf{Est}(\mathbf{E}_x[\alpha^i_{r_i(x)}], EX(f), \Theta/3, \delta/2k)$
6. $\quad$ **if** $E_\alpha \leq 2\Theta/3$ **then**
7. $\quad\quad k \leftarrow i$
8. $\quad\quad$ **break do**
9. $\quad$ **endif**
10. $\quad w_i \leftarrow$ WDNF'$(n, s, QEX(f), \tilde{D}_i(x), \delta/2k)$
11. **enddo**
12. $h(x) \equiv MAJ(w_0(x), w_1(x), \ldots, w_{k-1}(x))$
13. **return** h

FIG. 5.1. *Modified harmonic sieve* (HS$'$) *algorithm for learning DNF from a quantum example oracle* $QEX(f)$.

membership oracle, it is not enough. As noted above, we need to sample the parity functions according to the coefficients of the non-Boolean function αf. We will do this indirectly by sampling over individual bits of the function. First, note that we can limit the accuracy of α and still compute an adequate approximation to $\mathbf{E}[\alpha f \chi_a]$. Let T denote the quantity on the right-hand side of (5.1). Also let $d = \lceil \log(3/T) \rceil = O(\log(s/\epsilon^3))$. Then, since $0 \leq \alpha(x) \leq 1$ for all x, for $\theta(x) = \lfloor 2^d \alpha(x) \rfloor 2^{-d}$, we have

$$|\mathbf{E}[\theta f \chi_a]| \geq |\mathbf{E}[\alpha f \chi_a]| - \frac{T}{3}$$

for all χ_a. Furthermore, note that any value θ taken on by $\theta(x)$ can be written as

$$\theta = \theta_1 2^{-1} + \theta_2 2^{-2} + \cdots + \theta_d 2^{-d} + k 2^{-d},$$

where for each $j \in [1, d]$, $\theta_j \in \{-1, +1\}$ and $k \in \{-1, 0, 1\}$. Thus

$$|\mathbf{E}[\theta f \chi_a]| \leq \max_j |\mathbf{E}[\theta_j f \chi_a]| + \frac{T}{3}.$$

By (5.1), with high probability there exists χ_a such that $|\mathbf{E}[\alpha f \chi_a]| \geq T$. Therefore, for any such χ_a there is a fixed polynomial p_1 and an index j such that $|\mathbf{E}[\theta_j f \chi_a]| \geq 1/p_1(s, 1/\epsilon)$. Furthermore, for each j, the number of χ_a's such that $|\mathbf{E}[\theta_j f \chi_a]| \geq 1/p_1(s, 1/\epsilon)$ is at most $p_1^2(s, 1/\epsilon)$ by Parseval's identity. This suggests that to find a weak approximator to f with respect to D defined as in (4.1), we define θ as above and then apply quantum sampling using each of the d Boolean functions $\theta_j \cdot f$. We formalize this idea in the next section.

5.3. Modified HS algorithm. Combining the above observations, by modifying HS as shown in Figures 5.1 through 5.3, we obtain a quantum algorithm that uses a quantum example oracle to learn DNF with respect to the uniform distribution . The first difference between the new algorithm and the original is that **Est** will now be

Invocation: $w_i \leftarrow$ WDNF'$(n, s, QEX(f), \alpha, \delta)$
Input: n; $s =$ size of DNF f; $QEX(f)$; α, an oracle that given x returns $\alpha(x) \in [0,1]$; $\delta > 0$
Output: with probability at least $1-\delta$ (over random choices made by WDNF'), WDNF' returns h such that $\Pr_D[f = h] \geq 1/2 + 1/(8s+4)$

1. $T \equiv c_2\epsilon^3/(3(2s+1))$
2. $d \equiv \lceil \log(3/T) \rceil$
3. **for** $j \leftarrow 1, \ldots, n$ **do**
4. **for** $\ell \leftarrow 1, \ldots, 2^{2d+1}\ln(2n/\delta)$ **do**
5. $h =$ QSAMP'$(QEX(f), \theta_j)$
6. $EX(f) \equiv Obs(QEX(f))$
7. $E_c \leftarrow$ **Est-a**$(h, EX(f), \alpha, 1/(8s+4), \delta/(n2^{2d+2}\ln(2n/\delta)))$
8. **if** $|E_c| \geq 3/(8s+4)$ **then**
9. **return** $\mathrm{sign}(E_c) \cdot h$
10. **endif**
11. **enddo**
12. **enddo**
13. **return** 1

FIG. 5.2. WDNF' *subroutine called by* HS'. *Procedure* **Est-a**$(h, EX(f), \alpha, \epsilon, \delta)$, *described in the text, estimates* $\mathbf{E}_D[fh]$ *within accuracy* ϵ *with probability at least* $1-\delta$, *where* $D(x) = \alpha(x)/\sum_y \alpha(y)$. θ_j *represents the* j*th bit of* α.

Invocation: $h \leftarrow$ QSAMP'$(QEX(f), \theta_j)$
Input: Quantum example oracle $QEX(f)$; Boolean function $\theta_j(x)$.
Output: For $y \neq \bar{0}$, returns χ_y with probability $\hat{f}^2(y)/2$. Returns $\chi_{\bar{0}}$ with probability $1/2 + \hat{f}^2(\bar{0})/2$.

1. Call $QEX(f)$ on blank tape
2. In superposition, replace (reversibly) $f(x)$ with $(1 - f(x)\theta_j(x))/2$.
3. In superposition, apply Fourier operator F to the $n+1$ bits on the tape.
4. Perform an *Obs* operation.
5. Return χ_y, where y represents the first n bits on the tape.

FIG. 5.3. QSAMP' *subroutine called by* WDNF'.

given a simulated example oracle $EX(f)$—simulated using $QEX(f)$ as explained in section 3—in order to estimate expected values.

The second, more important, difference is that in order to find a weak approximator (line 2 of WDNF) we will use the quantum approach outlined above. That is, for each value of $j \in [1, d]$ we will sample the χ_a's in such a way that the probability of seeing each χ_a is exactly $\mathbf{E}^2[\theta_j f \chi_a]/2$. We do this by running a modified QSAMP that, after calling $QEX(f)$, replaces $f(x)$ with $\theta_j(x) \cdot f(x)$ expressed as a $\{0,1\}$-valued function. This is a reversible operation because x is still on the tape and $\theta_j^2(x) = 1$ for all x; therefore, this operation can be performed by a QTM. As shown in the preceding section, there exists some χ_a and some j such that χ_a is a weak approximator to f and $|\mathbf{E}[\theta_j f \chi_a]| \geq T/3 \geq 2^{-d}$. Therefore, if we run the modified QSAMP' $2^{2d+1}\ln(2n/\delta)$ times for each value of j, then—with probability at least $1 - \delta/2$—at least one of the χ_a's returned by the quantum sampler will be this weak approximator.

Finally, we will again simulate $EX(f)$—this time within WDNF′—in order to test whether or not a given h returned by QSAMP′ is a weak approximator. In order to perform this test, $\mathbf{E}_D[fh]$ is estimated (where D is the distribution defined by α as in (4.1)) by procedure **Est-a**. This procedure, given the uniform-distribution example oracle $EX(f)$ and the function $\alpha(x)$, simulates the example oracle $EX(f, D)$ using the same method as boost-by-filtering algorithms such as Freund's. Specifically, it queries $EX(f)$ and receives the pair $\langle x, f(x)\rangle$. It then flips a biased coin which produces heads with probability $\alpha(x)$. If the coin comes up heads, then the algorithm uses this pair in subsequent processing. Otherwise, it discards the pair, queries $EX(f)$ again, and repeats the coin-flip test. It can be shown that this process efficiently simulates $EX(f, D)$ for the α's produced by our algorithm (see, e.g., [18]). Finally, given $EX(f, D)$, by a standard Chernoff argument we can estimate $\mathbf{E}_D[fh]$ with the required accuracy and confidence with a number of samples polynomial in the appropriate parameters. Overall, WDNF′ allocates half of the confidence δ to estimating the $\mathbf{E}_D[fh]$'s. This gives us

THEOREM 2. *DNF is quantum learnable with respect to uniform.*

Also, $\widehat{PT}_1$, the class of functions expressible as a threshold of a polynomial number of parity functions, has the property that for every $f \in \widehat{PT}_1$ and every distribution D there exists a parity function χ_a that weakly approximates f with respect to D [22]. This was the only property of DNF that we used in the above arguments. Therefore, the following theorem holds.

THEOREM 3. *$\widehat{PT}_1$ is quantum learnable with respect to uniform.*

We now briefly examine the asymptotic time bound of this algorithm. First, Chernoff bounds tell us that to estimate the expected value of a Boolean variable to within an additive tolerance λ with confidence δ requires a sample (and time) $\tilde{O}(\lambda^2)$ (the notation $\tilde{O}(\cdot)$ is the same as standard big-O notation with log factors suppressed; in this case, δ contributes only a log factor and therefore does not appear in the bound). Using this fact and the earlier description of the algorithm, we can straightforwardly show that the algorithm as given has a time bound of $\tilde{O}(ns^6/\epsilon^{12})$. This compares with a bound on the harmonic sieve of $\tilde{O}(ns^8/\epsilon^{18})$. Furthermore, as noted in [24], the ϵ^3 factor that appears in the harmonic sieve can be brought arbitrarily close to ϵ^2; with this improvement the bounds improve to approximately $\tilde{O}(ns^6/\epsilon^8)$ and $(\tilde{O}(ns^8/\epsilon^{12})$, respectively. While neither bound is a "small" polynomial, the quantum algorithm is somewhat of an improvement. It is also reasonable to suspect that future improvements in the running time of the original algorithm would lead to similar improvements in the quantum algorithm as well.

6. Membership oracle versus quantum example oracle. In practice, it is not clear how a quantum example oracle could be constructed without using a membership oracle. Furthermore, because a QTM uses interference over an entire superposition to perform its computations, it might seem that perhaps there is some way to simulate a membership oracle given only a quantum example oracle by choosing a clever interference pattern. In this section we show that this is not the case.

DEFINITION 4. *We say that* membership queries can be quantum-example simulated for function class $\mathcal{F}$ *if there exists a BQP algorithm $\mathcal{A}$ and a distribution D such that for all $f \in \mathcal{F}$ and all x, running $\mathcal{A}$ on input x with quantum example oracle $QEX(f, D)$ produces $f(x)$.*

THEOREM 5. *Membership queries cannot be quantum-example simulated for DNF.*

Before proving this theorem, we develop some intuition. Consider two functions f_0 and f_1 that differ in exactly one input x. Then the superpositions returned by $QEX(f_0, D)$ and $QEX(f_1, D)$ are very similar for "almost all" D. In particular, if we think of superpositions as vectors in an inner product space of dimension 2^n, then there is in general an exponentially small angle between the superpositions generated by these two oracles. This angle will not be changed by unitary transformations. Thus, in general, an observation will be unable to detect a difference between the superpositions produced by $QEX(f_0, D)$ and $QEX(f_1, D)$. Therefore, a BQP algorithm with only a quantum example oracle $QEX(f_i, D)$, $i \in \{0, 1\}$, will be unable to correctly answer a membership query on x for both f_0 and f_1.

We now present two lemmas that will help us to formalize this intuition.

LEMMA 6. *Let $\mathcal{A}$ be a quantum algorithm that makes at most t calls to $QEX(f, D)$ for $f : \{0,1\}^n \to \{0,1\}$. Then there is an equivalent quantum program (modulo a slowdown polynomial in n and t) that makes all t calls at the beginning of the program.*

Proof. Let R_M be the unitary matrix representing the transitions of the QTM M. Let the configurations of M be encoded by bit strings in any reasonable way. Then suppose M is initially in a superposition

$$S_0 = \sum_y a_y |y\rangle,$$

where the sum is over all possible configurations y of M. After a single transition μ, M will be in the superposition

$$S_1 = \sum_{y,z} a_y R_M[z, y] |z\rangle.$$

Now assume that all of the configurations z with nonzero amplitude in S_1 cause $QEX(f, D)$ to be called. (Recall that by definition, either all of the configurations in a superposition cause this to happen or none of them do.) The resulting superposition will be

$$S_2 = \sum_{x,y,z} \sqrt{D(x)} a_y R_M[z, y] |z, x, f(x)\rangle$$

(by $z, x, f(x)$ we mean the configuration that results when $(x, f(x))$ is appended to the tape contents specified by the configuration z). But notice that there is a machine M' that, beginning with S_0, first calls $QEX(f, D)$, producing

$$S_1' = \sum_{x,y} \sqrt{D(x)} a_y |y, x, f(x)\rangle,$$

and then simulates the transition μ. A technical detail of this simulation is that a transition that corresponds to writing in a blank cell of y's tape necessitates shifting x and $f(x)$ to the right one cell first. Thus M' takes at most polynomially (in n) many steps and produces S_2 given S_0. A simple inductive argument completes the proof. ∎

Before presenting the next lemma, we need several definitions.

DEFINITION 7. *Define Obs over* any *linear combination of configurations (i.e., we no longer require that the sum of squared amplitudes be* 1*) as*

$$Obs\left(\sum_x u_x |x\rangle\right) = \sum_{x : x_1 = 1} |u_x|^2.$$

Define the length *of a linear combination of configurations* $S = \sum_x u_x|x\rangle$ *to be* $\|S\| = \sqrt{\sum_x |u_x|^2}$. *For any linear combination of configurations* S *we define for* $i \in \{0,1\}$

$$S^{(i)} = \sum_{x:x_1=i} u_x|x\rangle.$$

LEMMA 8. *Let* S_1 *and* S_2 *be superpositions and let* S *be any linear combination of configurations. Let* W *be any sequence of valid quantum operations. Then*

1. $Obs(S) \le \|S\|^2$;
2. $\|WS\| = \|S\|$;
3. $|\sqrt{Obs(S_1)} - \sqrt{Obs(S_2)}| \le \sqrt{Obs(S_1 - S_2)}$.

Proof. For part 1 we have

$$Obs(S) = \|S^{(1)}\|^2 \le \|S^{(0)}\|^2 + \|S^{(1)}\|^2 = \|S\|^2.$$

Part 2 follows from the fact that W is a unitary operation that preserves length.

To prove part 3 we have

$$\begin{aligned} |\sqrt{Obs(S_1)} - \sqrt{Obs(S_2)}| &= \left| \|S_1^{(1)}\| - \|S_2^{(1)}\| \right| \\ &\le \|S_1^{(1)} - S_2^{(1)}\| \\ &= \sqrt{Obs(S_1 - S_2)}. \end{aligned}$$ □

Proof of Theorem 5. By way of contradiction, assume that M is a QTM that can quantum-simulate membership queries for any DNF h, using calls to $QEX(h, D)$. By Lemma 6, we can assume without loss of generality that all of the calls to $QEX(h, D)$ occur at the beginning of M's program. Take $f(x) = 0$ and $g(x) = x_1^{c_1} \wedge \cdots \wedge x_n^{c_n}$, where $x^d = 1$ if and only if $x = d$. The second function is zero for all assignments to x except $c = (c_1, \ldots, c_n)$. We want to use the simulator M to find $h(c)$ for $h \in \{f, g\}$. The simulator will first make t calls to $QEX(h, D)$, giving

$$S_h = \sum_{z_1,\ldots,z_t} \sqrt{D(z_1)\cdots D(z_t)}|z_1, h(z_1), \ldots, z_t, h(z_t)\rangle.$$

After that, the computation for both f and g is the same in the sense that both computations consist of a series of applications of R_M to S_h. The superpositions S_f and S_g differ only in configurations

$$|z_1, h(z_1), \ldots, z_t, h(z_t)\rangle,$$

where one of the z_i is c.

Therefore,

$$S_f = S_g + E_{f,g},$$

where

$$\begin{aligned} E_{f,g} = &\sum_{(\exists i) z_i = c} \sqrt{D(z_1)\cdots D(z_t)}|z_1, f(z_1), \ldots, z_t, f(z_t)\rangle \\ &- \sum_{(\exists i) z_i = c} \sqrt{D(z_1)\cdots D(z_t)}|z_1, g(z_1), \ldots, z_t, g(z_t)\rangle. \end{aligned}$$

If W represents the sequence of transitions after the initial calls to $QEX(f, D)$, then at the end of the computation by M we observe $Obs(WS_f)$ and $Obs(WS_g)$ for f and g, respectively. By Lemma 8

$$\begin{aligned}
|\sqrt{Obs(WS_f)} - \sqrt{Obs(WS_g)}|^2 &\leq Obs(WE_{f,g}) \\
&\leq \|WE_{f,g}\|^2 \\
&= \|E_{f,g}\|^2 \\
&= 2 \sum_{(\exists i) z_i = c} \left(\sqrt{D(z_1) \cdots D(z_t)}\right)^2 \\
&= 2(1 - (1 - D(c))^t) \\
&\leq 2tD(c).
\end{aligned}$$

For any fixed D, any fixed polynomial p_1, and large enough n, almost all choices of c are such that $D(c) < 1/p_1(n)$. For all such c and appropriately chosen p_1 the observations $Obs(WS_f)$ and $Obs(WS_g)$ are indistinguishable. □

While it is not possible to simulate membership queries in polynomial time given only a quantum example oracle, it is a simple matter to simulate a uniform quantum example oracle with membership queries.

LEMMA 9. *For every Boolean function f, $QEX(f,\mathcal{U})$ can be simulated by a QTM making a single call to $MEM(f)$.*

Proof. $QEX(f,\mathcal{U})$ can be simulated by applying the Fourier transform F to the tape $|\bar{0}\rangle$ and then calling $MEM(f)$. □

Thus a membership oracle for f is strictly more powerful than a uniform quantum example oracle for f.

7. Concluding remarks. We have defined the notion of a quantum example oracle and argued that it is a natural quantum extension of the standard PAC example oracle. We then showed the learnability of DNF (and $\widehat{PT}_1$) with respect to uniform, given access to such an oracle. Such an oracle is also shown to be weaker (with respect to uniform) than a membership oracle.

While we believe that these results are interesting theoretically, like many oracle results in complexity theory the practical significance of our results is not clear. Our algorithm at the very least offers some potential speed-up over an implementation of the unmodified harmonic sieve for learning DNF from a membership oracle on a quantum computer. Also, while we do not currently see how to implement a quantum example oracle without recourse to a membership oracle, it is conceivable that there may be some way to build a quantum example oracle from something (much) less than a full membership oracle, which could add substantially to our algorithm's relevance to practical machine-learning problems. Finally, we hope that these results may provide stepping stones toward answering the larger question of whether DNF can be learned by a QTM from a standard PAC example oracle. A positive answer to this question has potentially great practical significance.

An earlier version of this paper [14] claimed that our quantum algorithm would learn DNF in a generalized persistent noise model. This propagated an erroneous claim by Jackson that the harmonic sieve was noise-tolerant [22]. While a modified version of the harmonic sieve has subsequently been shown to tolerate persistent classification noise [23], we do not consider the quantum extension of that algorithm in this paper, leaving this problem open for further study.

Acknowledgment. N. H. Bshouty thanks Richard Cleve for an enlightening seminar on quantum computation.

REFERENCES

[1] H. Aizenstein, L. Hellerstein, and L. Pitt, *Read-thrice DNF is hard to learn with membership and equivalence queries*, in Proceedings 33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, PA, 1992, pp. 523–532.

[2] H. Aizenstein and L. Pitt, *Exact learning of read-twice DNF formulas*, in Proceedings 32nd Annual Symp. on Foundations of Computer Science, San Juan, Puerto Rico, 1991, pp. 170–179.

[3] H. Aizenstein and L. Pitt, *Exact learning of read-k disjoint DNF and not-so-disjoint DNF*, in Proceedings 5th Annual Workshop on Computational Learning Theory, Pittsburgh, PA, 1992, pp. 71–76.

[4] D. Angluin, M. Frazier, and L. Pitt, *Learning conjunctions of Horn clauses*, Machine Learning, 9 (1992), pp. 147–164.

[5] D. Angluin and M. Kharitonov, *When won't membership queries help?*, J. Comput. System Sci., 50 (1995), pp. 336–355.

[6] E. Bernstein and U. Vazirani, *Quantum complexity theory*, in Proceedings 25th Annual ACM Symp. on Theory of Computing, San Diego, CA, 1993, pp. 11–20.

[7] A. Berthiaume and G. Brassard, *Oracle quantum computing*, in Proceedings Workshop on the Physics of Computation, Dallas, TX, IEEE Press, Piscataway, NJ, 1992, pp. 195–199.

[8] A. Berthiaume and G. Brassard, *The quantum challenge to structural complexity theory*, in Proceedings 7th IEEE Conference on Structure in Complexity Theory, Boston, MA, 1992, pp. 132–137.

[9] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich, *Weakly learning DNF and characterizing statistical query learning using Fourier analysis*, in Proceedings 26th Annual ACM Symp. on Theory of Computing, Montreal, Canada, 1994, pp. 253–262.

[10] A. Blum, R. Khardon, E. Kushilevitz, L. Pitt, and D. Roth, *On learning read-k-satisfy-j DNF*, in Proceedings 7th ACM Workshop on Comput. Learning Theory, ACM Press, New York, 1994, pp. 110–117.

[11] A. Blum and S. Rudich, *Fast learning of k-term DNF formulas with queries*, J. Comput. System Sci., 51 (1995), pp. 367–373.

[12] J. Bruck, *Harmonic analysis of polynomial threshold functions*, SIAM J. Discrete Math., 3 (1990), pp. 168–177.

[13] N. H. Bshouty, *Exact learning via the monotone theory*, in Proceedings 34th Annual Symp. on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 302–311.

[14] N. H. Bshouty and J. C. Jackson, *Learning DNF over the uniform distribution using a quantum example oracle*, in Proceedings 8th Annual Workshop on Computational Learning Theory, Santa Cruz, CA, 1995, pp. 118–127.

[15] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proceedings Roy. Soc. London Ser. A, 400, 1985, pp. 97–117.

[16] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, Proceedings Roy. Soc. London Ser. A, 439, 1992, pp. 553–558.

[17] Y. Freund, *An improved boosting algorithm and its implications on learning complexity*, in Proceedings 5th Annual Workshop on Computational Learning Theory, Pittsburgh, PA, 1992, pp. 391–398.

[18] Y. Freund, *Boosting a weak learning algorithm by majority*, Inform. and Comput., 121 (1995), pp. 256–285.

[19] Y. Freund and R. E. Schapire, *A decision-theoretic generalization of on-line learning and an application to boosting*, in Proceedings 2nd Annual European Conf. on Computational Learning Theory, Barcelona, Spain, 1995.

[20] O. Goldreich and L. A. Levin, *A hard-core predicate for all one-way functions*, in Proceedings 21st Annual ACM Symp. on Theory of Computing, Seattle, WA, 1989, pp. 25–32.

[21] T. R. Hancock, *Learning 2μDNF formulas and $k\mu$ decision trees*, in Proceedings 4th Annual Workshop on Computational Learning Theory, 1991, pp. 199–209.

[22] J. Jackson, *An efficient membership-query algorithm for learning DNF with respect to the uniform distribution*, in Proceedings 35th Annual Symp. on Foundations of Computer Science, 1994, pp. 42–53.

[23] J. Jackson, E. Shamir, and C. Shwartzman, *Learning with queries corrupted by classification noise*, in Proceedings 5th Israel Symp. on the Theory of Computing and Systems, 1997.

[24] J. C. Jackson, *The Harmonic Sieve: A Novel Application of Fourier Analysis to Machine Learning Theory and Practice*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, 1995. Available as Technical report CMU-CS-95-183.

[25] M. Kearns, M. Li, L. Pitt, and L. Valiant, *On the learnability of Boolean formulae*, in Proceedings 19th Annual ACM Symp. on Theory of Computing, New York, NY, 1987, pp. 285–295.

[26] E. Kushilevitz and Y. Mansour, *Learning decision trees using the Fourier spectrum*, SIAM J. Comput., 22 (1993), pp. 1331–1348.

[27] E. Kushilevitz and D. Roth, *On learning visual concepts and DNF formulae*, in Proceedings 6th Annual Workshop on Computational Learning Theory, Santa Cruz, CA, 1993, pp. 317–326.

[28] N. Linial, Y. Mansour, and N. Nisan, *Constant depth circuits, Fourier transform, and learnability*, J. ACM, 40 (1993), pp. 607–620.

[29] R. E. Schapire, *The strength of weak learnability*, Machine Learning, 5 (1990), pp. 197–227.

[30] P. W. Shor, *Algorithms for quantum computation: Discrete logarithms and factoring*, in Proceedings 35th Annual Symp. on Foundations of Computer Science, Santa Fe, NM, 1994, pp. 124–134.

[31] D. R. Simon, *On the power of quantum computation*, in Proceedings 35th Annual Symp. on Foundations of Computer Science, 1994, pp. 116–123.

[32] L. G. Valiant, *A theory of the learnable*, Comm. ACM, 27 (1984), pp. 1134–1142.

[33] A. C.-C. Yao, *Quantum circuit complexity*, in Proceedings 34th Annual Symp. on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 352–361.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1155–1166

APPROXIMABILITY AND NONAPPROXIMABILITY RESULTS FOR MINIMIZING TOTAL FLOW TIME ON A SINGLE MACHINE*

HANS KELLERER†, THOMAS TAUTENHAHN‡, AND GERHARD J. WOEGINGER§

Dedicated to the memory of Gene Lawler

Abstract. We consider the problem of scheduling n jobs that are released over time on a single machine in order to minimize the total flow time. This problem is well known to be NP-complete, and the best polynomial-time approximation algorithms constructed so far had (more or less trivial) worst-case performance guarantees of $O(n)$.

In this paper, we present one positive and one negative result on polynomial-time approximations for the minimum total flow time problem: The positive result is the first approximation algorithm with a *sublinear* worst-case performance guarantee of $O(\sqrt{n})$. This algorithm is based on resolving the preemptions of the corresponding optimum preemptive schedule. The performance guarantee of our approximation algorithm is not far from best possible, as our second, negative result demonstrates: Unless $P = NP$, no polynomial-time approximation algorithm for minimum total flow time can have a worst-case performance guarantee of $O(n^{1/2-\varepsilon})$ for any $\varepsilon > 0$.

Key words. scheduling, approximation algorithm, worst-case analysis, total flow time, release time, single machine

AMS subject classifications. 08C85, 68Q20, 05C38, 68R10, 90C35

PII. S0097539796305778

1. Introduction.

Scheduling independent jobs on a single machine has been studied extensively under various objective functions. We consider one of the basic problems of this kind, which from a worst-case point of view also appeared to be among the most intractable ones: There are given n independent jobs $J_1, \ldots, J_n$ which have to be scheduled nonpreemptively on a single machine. Each job J_i has a processing time p_i and becomes available for execution at its release time r_i, where p_i and r_i are nonnegative real numbers. All job data are known in advance. Without loss of generality we assume that the smallest release time is equal to zero. In a certain schedule for these jobs, let C_i be the *completion time* of job J_i, let S_i be its *starting time* (i.e., $S_i + p_i = C_i$), and let $F_i = C_i - r_i$ denote its *flow time*, respectively. The objective is to determine a schedule that minimizes the total flow time $\sum F_i$. This problem is commonly denoted by $1|r_i|\sum F_i$.

In case all job release times are identical, the problem can be solved in $O(n \log n)$ time by applying the well-known *shortest processing time* (SPT) rule; see Smith [19]. For arbitrary release times, the problem becomes NP-complete (Lenstra, Rinnooy Kan, and Brucker [15]). Several authors (Chandra [4], Chu [6], Deogun [7], and

*Received by the editors June 12, 1996; accepted for publication (in revised form) May 2, 1997; published electronically March 19, 1999. A preliminary version of this paper appeared in *Proc. 28th Annual ACM Symp. on the Theory of Computing*, 1997.

http://www.siam.org/journals/sicomp/28-4/30577.html

†Institut für Statistik, Ökonometrie und Operations Research, Universitätsstrasse 15, Universität Graz, A–8010 Graz, Austria (hans.kellerer@kfunigraz.ac.at).

‡Fakultät für Mathematik, Otto-von-Guericke Universität Magdeburg, D–39016 Magdeburg, Germany (on.tau@zib-berlin.de). This research was supported by DAAD (German Academic Exchange Service).

§TU Graz, Institut für Mathematik B, Steyrergasse 30, A-8010 Graz, Austria (gwoegi@opt.math.tu-graz.ac.at). The research of this author was supported by a research fellowship of the Euler Institute for Discrete Mathematics and Its Applications and by START project Y43-MAT of the Austrian Ministry of Science.

Dessouky and Deogun [8]) developed branch-and-bound algorithms for $1|r_i|\sum F_i$. Other papers (Bianco and Ricciardelli [3], Dyer and Wolsey [10], Hariri and Potts [13], and Posner [18]) gave branch-and-bound algorithms for $1|r_i|\sum w_iF_i$, the problem of minimizing the total *weighted* flow time. Gazmuri [12] designed asymptotically optimal algorithms for minimizing total flow time under very general probability distributions of the job data.

The preemptive version $1|\text{pmtn}, r_i|\sum F_i$ of the problem can be solved in polynomial time by the *shortest remaining processing time* (SRPT) rule (see, e.g., Baker [2]). The objective value of the optimum preemptive schedule clearly is a lower bound on the objective value of the nonpreemptive problem. Ahmadi and Bagchi [1] have shown that it dominates six other lower bounds for $1|r_i|\sum F_i$ that had been used in the scheduling literature.

Approximation algorithms. For an instance I of $1|r_i|\sum F_i$, let $F^H(I)$ denote the total flow time obtained when algorithm H is applied to I, and let $F^*(I)$ be the objective value of an optimum solution for I. We usually write F^H and F^* instead of $F^H(I)$ and $F^*(I)$, respectively, if the instance I is clear from the context. Now let $\rho : \mathbb{R}^+ \to \mathbb{R}^+$ and let H be an approximation algorithm. We say that algorithm H has *worst-case performance guarantee* ρ if

$$\sup\{F^H(I)/F^*(I) \mid I \text{ is an instance with } n \text{ jobs } \} \;\le\; \rho(n)$$

holds for all integers $n \ge 1$.

Chu [5] and Mao and Rifkin [17] investigated several "reasonable" algorithms for $1|r_i|\sum F_i$, but none of them yielded a sublinear worst-case performance guarantee. For instance, a straightforward procedure for $1|r_i|\sum F_i$ is the *earliest start time* (EST) algorithm: "Whenever the machine becomes free for assignment, take the shortest of the (at that time) available jobs and assign it to the machine." The following example shows that the total flow time in an EST schedule can be a factor of n away from the optimum: Let ε be a very small positive real number. Then choose $p_1 = 1$, $p_2 = \cdots = p_n = \varepsilon$, $r_1 = 0$, $r_2 = \cdots = r_n = \varepsilon$. The EST schedule processes the jobs in the ordering $(1, 2, \ldots, n)$ with total flow time $\Omega(n)$, whereas an optimum schedule processes the jobs in the ordering $(2, 3, \ldots, n-1, 1)$ with total flow time $O(1)$.

Another direct approach is the *earliest completion time* (ECT) rule. An ECT schedule is formed by scheduling jobs without any unnecessary idle time, where the next job to be scheduled is the one with the earliest possible completion time of all unscheduled jobs. Again, let ε be a very small positive real number. Then the job data $p_1 = 1 + 2\varepsilon$, $p_2 = \cdots = p_n = \varepsilon$ and $r_i = i - 1$ for all $i = 1, \ldots, n$ yields an ECT schedule of $(2, 3, \ldots, n-1, 1)$ with total flow time $\Omega(n)$, whereas the optimum schedule is $(1, 2, \ldots, n)$ with total flow time $O(1)$. Again, we get a worst-case ratio of n.

New results. The results of this paper are as follows. First we present a polynomial-time approximation algorithm with worst-case performance guarantee of $O(\sqrt{n})$. The algorithm starts from an optimum solution of the corresponding *preemptive* instance. Then the preemptive schedule is transformed step by step into a nonpreemptive one by successively resolving the preemptions. Our proof also demonstrates that for any instance the minimum total flow time of an optimum nonpreemptive schedule is at most a factor of $\sqrt{n}$ larger than the corresponding minimum total flow time of the optimum preemptive schedule.

In the second part of the paper, we prove that polynomial-time approximation algorithms for minimum total flow time on a single machine cannot have a worst-case performance guarantee of $O(n^{1/2-\varepsilon})$ for any $\varepsilon > 0$. This result is derived by an appropriate reduction from the NP-complete numerical three-dimensional matching problem.

Organization of the paper. Section 2 describes the approximation algorithm and proves the claimed worst-case performance guarantee of $O(\sqrt{n})$. Section 3 gives the nonapproximability result, and section 4 contains the discussion.

2. The approximation algorithm. In this section, we design a polynomial-time approximation algorithm for the minimum total flow time problem that has worst-case performance guarantee $O(\sqrt{n})$. The main idea is to start with an optimum solution of the corresponding problem, where preemption is allowed, and then to get rid of the preemptions while increasing the total flow time by only some "moderate" amount.

As already mentioned in the introduction, an optimum preemptive schedule can be obtained by applying the SRPT rule, which is defined as follows: The remaining processing time $P_i(t)$ of job J_i is the amount of processing time of J_i which has not been scheduled before time t. At any time t, the available job J_i with SRPT $P_i(t)$ is processed until it is either completed or until another job J_j with $p_j < P_i(r_j)$ becomes available. In the second case, J_i is preempted and J_j is processed.

We denote the total flow time in an optimum preemptive schedule by F^*_{pmtn}. With every job J_i we associate the interval $\mathcal{I}_i = [S_i, C_i]$ in the preemptive schedule. Note that due to possible preemptions, the length of $\mathcal{I}_i$ in general need not be equal to p_i. Let us observe some simple properties of the SRPT schedule: If a job J_i has a preemption at time t, then the processing of some job J_j with $p_j < P_i(t)$ starts at this time. Consequently, job J_i cannot return to the machine until job J_j is completed.

OBSERVATION 2.1. *In the SRPT schedule, for any two jobs J_i and J_j either the corresponding intervals $\mathcal{I}_i$ and $\mathcal{I}_j$ are disjoint or one of them contains the other one. Furthermore, there is no machine idle time during $\mathcal{I}_i$ and $\mathcal{I}_j$.* □

With the preemptive schedule we associate in the following way a directed ordered forest that describes the containment relations of the intervals $\mathcal{I}_i$. The jobs $J_1, \ldots, J_n$ form the vertices of the forest. We introduce a directed edge going from job J_i to J_j if and only if $\mathcal{I}_j \subseteq \mathcal{I}_i$ and there does not exist a job J_k with $\mathcal{I}_j \subseteq \mathcal{I}_k \subseteq \mathcal{I}_i$. For every vertex J_i, its sons are ordered from left to right according to the ordering of their corresponding intervals $\mathcal{I}_j$. This yields a collection of ordered directed out-trees. Out-trees that consist of a single vertex are called *trivial* out-trees. To complete the definition of the forest, we also order the roots of these out-trees from left to right according to the ordering of their corresponding intervals. By $T(i)$ we denote the maximal subtree of this forest rooted at J_i (hence, $T(i)$ exactly contains those jobs that are processed during $[S_i, C_i]$). A *leaf* is a job with out-degree zero. (Hence, it does not have any preemptions.)

Since our approximation algorithm transforms the preemptive schedule into a nonpreemptive one, the forest associated with the final schedule will be a collection of n trivial out-trees. In sections 2.1, 2.2, and 2.3, we describe three procedures for removing preemptions and simplifying the forest structure. Every procedure acts on a certain subtree $T(i)$ and removes all preemptions of the job J_i. Section 2.4 explains how the approximation algorithm applies and combines these three procedures.

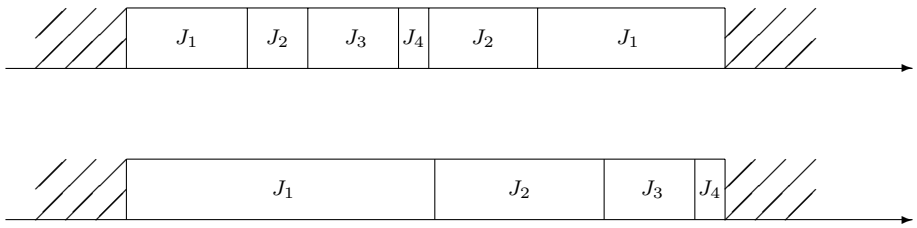


FIG. 1. *Illustration for Procedure SmallSubtree(i) with* $i = 1$.

2.1. How to handle small subtrees. A subtree $T(i)$ is called *small* if it contains at most $\sqrt{n}$ jobs. In this case we use the following procedure to resolve the preemptions of job J_i and of all jobs contained in $T(i)$.

PROCEDURE SmallSubtree(i).
Let $S_i = S_{i_0} < S_{i_1} < \cdots < S_{i_k}$ denote the starting times of the jobs in $T(i)$ in the current preemptive schedule. Remove all jobs and reinsert them without preemptions in the ordering $J_i, J_{i_1}, J_{i_2}, \ldots, J_{i_k}$.

For an illustration, see Figure 1. It is easy to see that SmallSubtree(i) does not decrease the starting time of any job. Hence, the resulting schedule still obeys all release times. Completion times of jobs outside of $T(i)$ are not changed, and the completion times of jobs in $T(i)$ are all increased by less than $C_i - r_i$. Since there are at most $\sqrt{n}$ jobs in $T(i)$,

$$\Delta_{small}(i) \;=\; \sqrt{n} F_i \tag{1}$$

is an upper bound on the increase of the objective function caused by SmallSubtree(i).

2.2. How to handle the last root. Now let J_i be the root of the rightmost nontrivial out-tree. (J_i is preempted, but all jobs that are processed after C_i are without preemptions.) Such a root J_i is called the *last root* and may be handled according to the following procedure.

PROCEDURE LastRoot(i).
Compute $\lceil \sqrt{n} \rceil + 1$ time points $t_0 = C_i, t_1, \ldots, t_{\lceil \sqrt{n} \rceil}$ such that there are exactly hp_i units of idle time between C_i and t_h in the current schedule for all $h \in \{0, \ldots, \sqrt{n}\}$. Determine $0 \le k \le \lceil \sqrt{n} \rceil - 1$ so as to minimize the value of $k + |\{j : \; t_k \le S_j \le t_{k+1}\}|$.
In case there is a job processed during $[t_k, t_k + p_i]$, shift it to the right until its processing is disjoint from $[t_k, t_k + p_i]$. If necessary shift also some later jobs to the right, but without changing their relative orderings and without introducing any unnecessary idle time. Then remove J_i and reschedule it in the interval $[t_k, t_k + p_i]$.

For an illustration, see Figure 2. Since the job starting times are not decreased, the resulting schedule is again feasible. Moreover, by definition of the numbers t_h, there are exactly p_i units of idle time between t_k and t_{k+1}. Hence, the shifting of jobs takes place only within the interval $[t_k, t_{k+1}]$, and no jobs outside of this interval are affected by LastRoot(i).

LEMMA 2.2. *The value of* k *determined by Procedure LastRoot(i) satisfies*

$$k + |\{j : \; t_k \le S_j \le t_{k+1}\}| \;\le\; \frac{3}{2}\sqrt{n}.$$

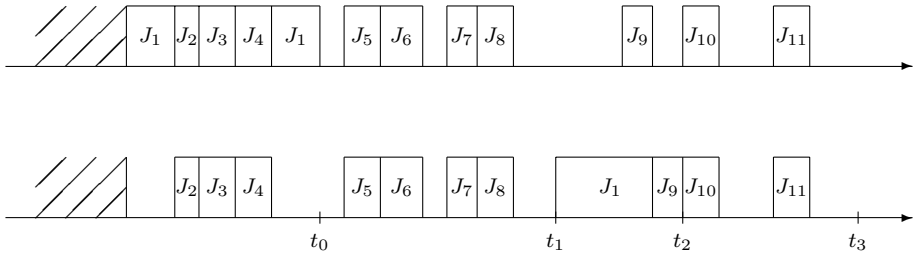


FIG. 2. *Illustration for Procedure LastRoot(i) with* $i = 1$.

Proof. Define $n_h = |\{j : \ t_h \le S_j \le t_{h+1}\}|$. Then

$$\sum_{h=0}^{\lceil\sqrt{n}\,\rceil-1} (h + n_h) \ \le \ \frac{1}{2}\lceil\sqrt{n}\,\rceil(\lceil\sqrt{n}\,\rceil - 1) + n \ \le \ \frac{3}{2}\sqrt{n}\lceil\sqrt{n}\,\rceil.$$

Since k was chosen to minimize $k + n_k$, the value $k + n_k$ is bounded from above by the average of these $\lceil\sqrt{n}\,\rceil$ numbers. □

Procedure LastRoot(i) increases the flow time of job J_i by at most $kp_i + \sum_{j=1}^{n} p_j$. Furthermore, the flow times of at most $n_k = |\{j : t_k \le S_j \le t_{k+1}\}|$ other jobs are increased by at most p_i. By applying Lemma 2.2 one gets that LastRoot(i) increases the value of the objective function by at most

$$\Delta_{last}(i) \ = \ kp_i + \sum_{j=1}^{n} p_j + n_k p_i \ \le \ \frac{3}{2}\sqrt{n}\, p_i + \sum_{j=1}^{n} p_j. \tag{2}$$

Finally, observe that in the new schedule J_i is processed entirely *after* its completion time in the old schedule.

2.3. How to handle fathers and sons. Procedure FatherSon(i, j) described below may be applied only if the jobs J_i and J_j fulfill the following four conditions:

(C1) J_j is a son of J_i.
(C2) All sons of J_i in $T(i)$ that lie to the right of J_j are leaves (i.e., jobs without preemption).
(C3) There are less than $\sqrt{n}$ jobs that lie to the right of J_j in $T(i)$.
(C4) Just before executing FatherSon(i, j), job J_i is removed and rescheduled entirely *after* its old completion time.

Condition (C4) may be fulfilled if J_i is rescheduled by LastRoot(i) or (as we will see) if it is rescheduled by FatherSon(k, i), where k was the father of i at that time. Condition (C4) has the following consequences: At the moment as SRPT decided to start processing J_j instead of J_i, it did so because the remaining processing time of J_i was larger than p_j. Hence, the total time that J_i is processed during $[C_j, C_i]$ is at least p_j, and *in case* J_i *is moved to some place after* C_i*, there is sufficient empty space in* $[C_j, C_i]$ *to schedule all of* J_j.

PROCEDURE FatherSon(i, j).

All jobs that are processed during $[C_j, C_j + p_j]$ are shifted to the right until their processing is disjoint from $[C_j, C_j + p_j]$. If necessary, also, some later jobs are shifted to the right but without changing

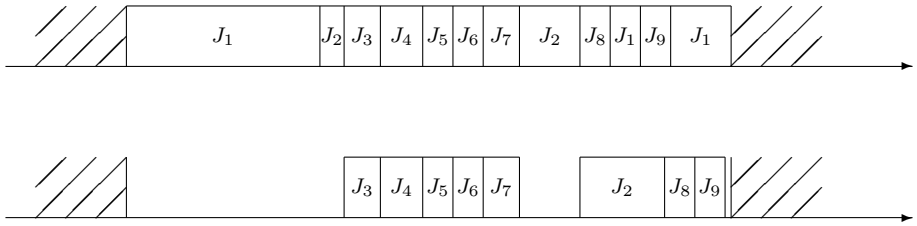


FIG. 3. *Illustration for Procedure FatherSon(i, j) with* $i = 1$ *and* $j = 2$.

their relative orderings and without introducing any unnecessary idle time.

Then remove J_j and reschedule it to the interval $[C_j, C_j + p_j]$.

For an illustration, see Figure 3. Note that in the new schedule J_j is processed entirely *after* its completion time in the old schedule. Furthermore note that Procedure FatherSon(i, j) transforms all sons of J_i in $T(i)$ that lie to the right of J_j into *trivial* out-trees.

Procedure FatherSon(i, j) produces another feasible schedule where job J_j no longer has preemption. Only jobs in the interval $[C_j, C_i]$ are shifted to the right, and no other jobs are affected. All the shifted jobs are contained in $T(i)$, they lie to the right of J_j, and, by condition (C3), their number is less than $\sqrt{n}$. The completion time of every moved job (including job J_j) increases by at most p_j. Hence, the total increase in the value of the objective function that FatherSon(i, j) produces can be bounded by

$$\Delta_{fson}(j) \;=\; \sqrt{n}\, p_j. \tag{3}$$

2.4. Putting everything together. Finally, we describe our approximation algorithm in detail. The algorithm starts with the optimum schedule for the preemptive problem and determines the corresponding directed ordered forest.

In the first phase, Procedure SmallSubtree(i) is applied to all jobs J_i for which $2 \le |T(i)| \le \sqrt{n}$ holds. Afterward, every subtree $T(i)$ either fulfills $|T(i)| = 1$ (and J_i has no preemption) or $|T(i)| > \sqrt{n}$ holds.

In the second phase, the algorithm goes through a number of steps and repeatedly modifies the rightmost nontrivial out-tree. Every such step increases the number of trivial out-trees by at least $\sqrt{n}$. Every step consists of one call to LastRoot, possibly followed by several calls to FatherSon. For a nonleaf vertex J_i in the forest, let RMNLS(i) denote the index of its *rightmost nonleaf son* and let LEAF(i) denote the number of sons of J_i lying to the right of RMNLS(i). (Obviously, all these LEAF(i) sons must be leaves.) Every step of the second phase is performed as follows. Let J_i be the root of the rightmost nontrivial out-tree.

LastRoot(i);
While RMNLS(i) exists and LEAF$(i) < \sqrt{n}$ do
    FatherSon$(i$,RMNLS$(i))$
    $i :=$RMNLS(i)
EndWhile

Let J_{i^*} denote the last job whose preemptions are resolved in such a step. Then either J_{i^*} has more than $\sqrt{n}$ sons (that all are leaves) to the right of RMNLS(i^*) or job

RMNLS(i^*) does not exist. In the latter case, all sons of J_i^* are leaves and, according to the result of the first phase, their number must be greater than $\sqrt{n}$. In either case, there are at least $\sqrt{n}$ leaves that formerly were preempting J_{i^*}, the father of J_{i^*}, its grandfather, and so on—the whole chain of ancestors to the formerly "last root." Since the execution of the step removes all preemptions from this chain of ancestors, it turns all these at least $\sqrt{n}$ leaves into trivial out-trees.

In the second phase of the algorithm the step described above is repeated over and over until all preemption have been removed. Since every step produces at least $\sqrt{n}$ new trivial out-trees, the second phase terminates after at most $\sqrt{n}$ steps. This completes the description of our approximation algorithm.

THEOREM 2.3. *The approximation algorithm described for the problem* $1|r_i|\sum F_i$ *has a worst-case performance guarantee of* $O(\sqrt{n})$*, and it can be implemented to run in* $O(n^{3/2})$ *time. Moreover, there exist instances for which the algorithm yields a schedule whose objective value is* $\Omega(\sqrt{n})$ *from the optimum.*

Proof. To analyze the worst-case behavior of the algorithm, we define job classes $\mathcal{S}$, $\mathcal{L}$, and $\mathcal{F}$. Class $\mathcal{S}$ contains the jobs J_i for which SmallSubtree(i) is executed, $\mathcal{L}$ the jobs J_i for which LastRoot(i) is executed, and $\mathcal{F}$ the jobs J_j for which FatherSon(i, j) is executed. Since every procedure removes all preemptions of the corresponding job, every job is contained in at most one of the classes $\mathcal{S}$, $\mathcal{L}$, and $\mathcal{F}$. Moreover, $|\mathcal{L}| \le \sqrt{n}$ holds, since LastRoot(i) is executed exactly once per step in the second phase, and there are at most $\sqrt{n}$ steps.

The constructed nonpreemptive schedule has an objective value F^H which satisfies

$$\begin{aligned} F^H &\le F^*_{pmtn} + \sum_{i\in\mathcal{S}} \Delta_{small}(i) + \sum_{i\in\mathcal{L}} \Delta_{last}(i) + \sum_{i\in\mathcal{F}} \Delta_{fson}(i) \\ &\le F^*_{pmtn} + \sum_{i\in\mathcal{S}} \sqrt{n} F_i + \sum_{i\in\mathcal{L}} \left(\frac{3}{2}\sqrt{n}\, p_i + \sum_{j=1}^{n} p_j \right) + \sum_{i\in\mathcal{F}} \sqrt{n}\, p_i. \end{aligned}$$

Here we used (1), (2), and (3). If we also apply the straightforward relations $p_i \le F_i$, for $1 \le i \le n$ and $\sum_{i=1}^{n} F_i = F^*_{pmtn}$, together with $|\mathcal{L}| \le \sqrt{n}$, then we determine that

$$F^H \;\le\; F^*_{pmtn} + \frac{3}{2}\sqrt{n} \sum_{i\in\mathcal{S}\cup\mathcal{L}\cup\mathcal{F}} F_i + |\mathcal{L}| \sum_{j=1}^{n} p_j \;\le\; \left(1 + \frac{5}{2}\sqrt{n}\right) F^*_{pmtn} \tag{4}$$

holds. Since $F^*_{pmtn} \le F^*$, this proves that the algorithm has a worst-case performance guarantee of $O(\sqrt{n})$.

What about the time complexity? The SRPT schedule can be computed in $O(n \log n)$ time. Procedure SmallSubtree needs to be run only on the vertices which have at most $\sqrt{n}$ descendants and whose father is either nonexistent or has more than $\sqrt{n}$ descendants. All these vertices can be located in a single $O(n)$ time traversal of the forest. Hence, procedures SmallSubtree and FatherSon are both executed only $O(n)$ times and always consider only $O(\sqrt{n})$ of the jobs. Procedure LastRoot may have to deal with up to $O(n)$ jobs, but it is executed only $O(\sqrt{n})$ times. Summarizing, this yields a time complexity of

$$|\mathcal{S}| \cdot O(\sqrt{n}) + |\mathcal{L}| \cdot O(n) + |\mathcal{F}| \cdot O(\sqrt{n}) \;\le\; O(n^{3/2}).$$

To see that the algorithm can be a factor of $\Omega(\sqrt{n})$ from the optimum, consider the instance of $n = x^2$ jobs: The jobs J_i with $1 \le i \le x - 1$ all have processing

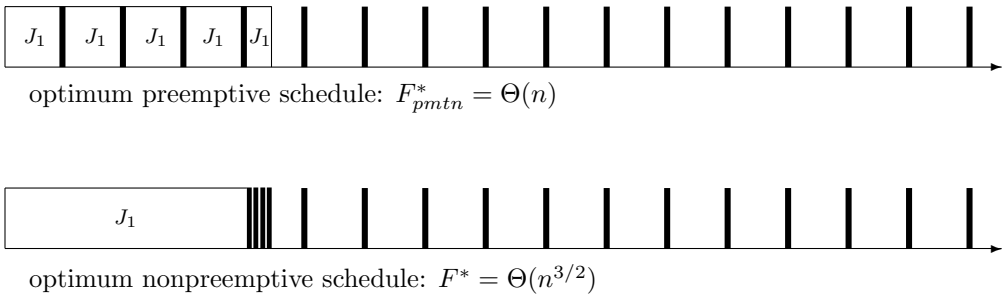


FIG. 4. *Illustration for the lower bound example in Corollary* 2.4.

time $1/(x-1)$ and release times $1+(i-1)/(x-1)$. Job J_x has processing time x^2 and release time 0. The remaining x^2-x jobs are dummy jobs, all with processing times 0 and release times 0. The forest corresponding to the optimum preemptive schedule has a single nontrivial tree with root J_x and $J_1, \ldots, J_{x-1}$ as leaves. Hence, our algorithm calls SmallSubtree(x) and produces a schedule with $F^H = x^3 - x + 2$, whereas $F^* = x^2 + 3$ holds. As x tends to infinity, the ratio F^H/F^* grows like $\Theta(x) = \Theta(\sqrt{n})$. □

COROLLARY 2.4. *For any instance of n jobs with release times on a single machine, the inequality $F^*/F^*_{pmtn} \le \frac{5}{2}\sqrt{n} + 1$ holds. Moreover, for every sufficiently large n, there exists an instance I_n such that $F^*(I_n)/F^*_{pmtn}(I_n) \approx \sqrt{n}$.*

Proof. The upper bound follows from (4) and $F^* \le F^H$. The lower bound follows from the instance with n jobs where $r_1 = 0$, $p_1 = n$, $r_i = (i-1)\sqrt{n}$, and $p_i = \varepsilon = 1/n$ for $i \ge 2$. Then $F^*_{pmtn} \approx n$ and $F^* \approx n^{3/2}$ holds. (See Figure 4.) □

3. The nonapproximability result. In this section, we will prove that no polynomial-time approximation algorithm for minimizing the total flow time may have worst-case performance guarantee $O(n^{1/2-\varepsilon})$ for any $\varepsilon > 0$. The proof is done by a reduction from the following version of the NP-complete numerical three-dimensional matching problem (see Garey and Johnson [11]).

Problem. Numerical three-dimensional matching (N3DM).

Instance. Positive integers a_i, b_i, and c_i, $1 \le i \le k$, with $\sum_{i=1}^{k}(a_i + b_i + c_i) = kD$.

Question. Do there exist permutations π, ψ such that $a_i + b_{\pi(i)} + c_{\psi(i)} = D$ holds for all i?

This problem remains NP-complete even if the numbers a_i, b_i, and c_i are encoded in unary and the total size of the input is $\Theta(kD)$. In the following discussion, we will make use of this unary encoding. Consider an arbitrary instance of N3DM and let $0 < \varepsilon < \frac{1}{2}$ be some real number. Define numbers

$$n = \lceil (20k)^{4/\varepsilon} D^{2/\varepsilon} \rceil, \qquad r = \lceil 2Dn^{(1-\varepsilon)/2} \rceil, \qquad g = 100rk^2.$$

Next we will construct from the N3DM instance and from the number ε a corresponding scheduling instance with n jobs. For every number a_i in the N3DM instance, we introduce a corresponding job with processing time $2r + a_i$, for every b_i we introduce a job with processing time $4r + b_i$, and for every c_i we introduce a job with processing time $8r + c_i$. These $3k$ jobs are called the *big* jobs and they are all released at time 0.

Moreover, there will be a number of so-called *tiny* jobs. Tiny jobs occur only in groups denoted by $G(t;\ell)$, where t and ℓ are positive integers. A group $G(t;\ell)$ consists

of ℓ tiny jobs with processing time $1/\ell$. They are released at the times $t + i/\ell$ for $i = 0, \ldots, \ell - 1$. Note that it is possible to process all jobs in $G(t; \ell)$ in a feasible way during the time interval $[t, t + 1]$ with a total flow time of 1. We introduce the following groups of tiny jobs.

(T1) For $1 \le i \le k$, we introduce the group $G((14r + D + 1)i - 1; rg)$.

(T2) For $1 \le i \le g$, we introduce the group $G((14r + D + 1)k + ri - 1; g)$.

In a pictorial setting, the groups of type (T1) occur at regular intervals from time $14r + D$ to time $(14r + D + 1)k$. They leave k holes, where each hole has length $14r + D$. Similarly, the groups of type (T2) occur in a regular pattern after time $(14r + D + 1)k$. They leave holes of size $r - 1$.

So far we have introduced $3k$ big jobs and $k + 100rk^2$ groups of tiny jobs, which amounts to an overall number of

$$3k + krg + 100rk^2 g = 3k + 100r^2k^3 + 10,000r^2k^4 < 11,000r^2k^4 < n$$

jobs. In order to simplify calculations, we introduce a number of artificial jobs all with processing time zero and release time zero such that the total number of jobs becomes equal to n. This completes the construction of the scheduling instance.

LEMMA 3.1. *If the N3DM instance has a solution, then for the constructed scheduling instance $F^* \le 200rk^2$ holds.*

Proof. Consider the following feasible schedule. All tiny jobs are processed immediately at their release times. Hence, their total flow time equals $k + 100rk^2$. For every triple $(a_i, b_{\pi(i)}, c_{\psi(i)})$ with sum D in the solution of the N3DM instance, we pack the corresponding three jobs together into one of the holes of length $14r + D$ that are left free by the groups of type (T1). The job corresponding to a_i is processed first, the job corresponding to $b_{\pi(i)}$ is processed second, and the job corresponding to $c_{\psi(i)}$ is processed last in this hole. It is easy to see that this yields a total flow time of

(5)
$$k + 100rk^2 + 3\sum_{i=1}^{k}(a_i + 2r) + 2\sum_{i=1}^{k}(b_i + 4r) + \sum_{i=1}^{k}(c_i + 8r) + \frac{3}{2}(14r + D + 1)k(k - 1).$$

Since $\sum_{i=1}^{k}(a_i + b_i + c_i) = kD$ and since $D \le r$, one easily gets an upper bound of

$$k + 100rk^2 + 25rk + 24rk(k - 1) \;<\; 200rk^2$$

for the expression in (5). □

LEMMA 3.2. *If the N3DM instance does not have a solution, then for the constructed scheduling instance $F^* \ge 100r^2k^2$ holds.*

Proof. Consider an optimum schedule S^* and suppose that its total flow time is strictly less than $100r^2k^2$. Our first claim then is that in S^* all tiny jobs in groups of type (T1) are processed as soon as they are released: Since all these tiny jobs have identical processing times, they are processed in S^* in order of increasing release times. It is easy to see that there is no use in splitting one of the groups of type (T1) by processing some larger jobs in between (since this would contradict the SPT rule). Since all big jobs have integer processing times and since the total processing time in every group is equal to 1, we may further assume that the starting time of every big job and of every group $G(t; \ell)$ in S^* is an integer. When the processing of some group $G(t; rg)$ starts at time $t + x$, for x an integer, the total flow time of the rg tiny jobs in $G(t; rg)$ is $rgx + 1 = 100r^2k^2x + 1$. This implies $x = 0$ and proves the claim.

Our second claim is that in S^* none of the big jobs are processed during the time interval that starts with release of the last group of type (T1) and ends with release of the last group of type (T2). Suppose otherwise. The groups of type (T2) are released at regular intervals of length r. Since big jobs all have processing times of at least $2r$, scheduling a big job somewhere between these groups would shift the jobs of at least one group by at least r time units away from their release time. This would yield a total flow time of at least $gr = 100r^2k^2$ and proves the second claim.

Our third claim is that in S^* one of the big jobs is processed after the last group of tiny jobs. Suppose otherwise. There are two types of holes that are left free by the tiny jobs for processing the big jobs: k holes of length $14r + D$ and $100rk^2$ holes of size $r - 1$. Since big jobs all have processing times of at least $2r$, they must be packed into the holes of size $14r + D \leq 15r$. Since two jobs corresponding to the numbers c_i and c_j have total processing time at least $16r$, every such hole must contain exactly one job corresponding to some c_i. By analogous arguments, we determine that every hole of size $14r+D$ must contain exactly one job corresponding to some a_i, b_j, and c_h, respectively. This implies that the corresponding three numbers fulfill $a_i+b_j+c_h \leq D$, which in turn yields $a_i + b_j + c_h = D$ (since the total sum of these $3k$ numbers is kD and the total size of the holes is kD). Hence, the N3DM instance would have a solution; this contradiction shows that our third claim holds true.

Finally, observe that the last group of tiny jobs is processed at time $t = (14r + D + 1)k + 100r^2k^2 - 1$. Hence, the big job that is processed after this last group has a completion time of at least $100r^2k^2$, and since its release time is zero, its flow time is at least $100r^2k^2$. This final contradiction completes the proof of the lemma. □

THEOREM 3.3. *For all $0 < \varepsilon < \frac{1}{2}$ and $0 < \alpha < 1$, there does not exist a polynomial-time approximation algorithm for the minimum total flow time problem with worst-case approximation guarantee $\alpha n^{1/2-\varepsilon}$ unless $P = NP$.*

Proof. Suppose such an approximation algorithm H would exist for some fixed $0 < \varepsilon < \frac{1}{2}$ and $0 < \alpha < 1$. Take an instance of N3DM that is encoded in unary, and perform the above construction. Since the instance is encoded in unary, the size of the resulting scheduling instance is polynomial in the size of the N3DM instance.

Then apply algorithm H to the scheduling instance. In case the N3DM instance has a solution, Lemma 3.1 and the worst-case performance guarantee of H imply that

$$F^H \leq \alpha n^{1/2-\varepsilon} \cdot F^* < \frac{r}{2} \cdot 200rk^2 = 100r^2k^2.$$

In case the N3DM instance does not have a solution, Lemma 3.2 implies that

$$F^H \geq F^* \geq 100r^2k^2$$

holds. Hence, with the help of algorithm H we could distinguish between these two possibilities in polynomial time, which would imply $P = NP$. □

4. Concluding remarks. In this paper we investigated the problem of minimizing total flow time on a single machine. For this NP-complete problem, only approximation algorithms with worst-case bounds of $\Omega(n)$ were known up to now. We presented the first approximation algorithm with sublinear worst-case performance guarantee. The algorithm has a tight worst-case of $O(\sqrt{n})$. It is based on a resolution of the interrupted jobs in the corresponding optimum preemptive schedule. Moreover, we derived a lower bound on the worst-case performance guarantee of polynomial-time approximation algorithms for this problem. We proved that no

polynomial-time approximation algorithm can have a worst-case performance guarantee of $O(n^{1/2-\varepsilon})$ with $\varepsilon > 0$. This also shows that our approximation algorithm is almost "best possible."

We finish the paper with some remarks and open problems.

(1) The open main problem concerns closing the gap between our upper bound and our lower bound. For example, there might exist an approximation algorithm with worst-case performance guarantee of $O(\sqrt{n}/\log n)$.

(2) Subsequent to and inspired by our work, Leonardi and Raz [16] investigated the problem $Pm|r_i|\sum F_i$ with parallel machines. Since the preemptive problem $Pm|\text{pmtn}, r_i|\sum F_i$ is NP-complete for $m \geq 2$ machines [9], Leonardi and Raz [16] do not have the optimum preemptive schedule as a starting point for their approximation algorithm. Instead, they start with an approximative solution for this relaxation and then resolve the preemptions.

(3) Another open problem consists in designing approximation algorithms for the total weighted flow time problem on a single machine. Also in this case, the corresponding preemptive problem $1|\text{pmtn}, r_i|\sum w_i F_i$ is NP-complete (Labetoulle et al. [14]).

(4) The *waiting* time W_i of some job J_i in a schedule is defined by $W_i = S_i - r_i$; i.e., it is the time that the job spends in the system while waiting for being processed. We are not aware of any positive results on approximating the total waiting time.

Acknowledgment. Gerhard Woeginger acknowledges helpful discussions with Amos Fiat and Stefano Leonardi.

REFERENCES

[1] R.H. Ahmadi and U. Bagchi, *Lower bounds for single-machine scheduling problems*, Naval Res. Logist., 37 (1990), pp. 967–979.

[2] K.R. Baker, *Introduction to Sequencing and Scheduling*, John Wiley, New York, 1974.

[3] L. Bianco and S. Ricciardelli, *Scheduling of a single machine to minimize total weighted completion time subject to release dates*, Naval Res. Logist., 29 (1982), pp. 151–167.

[4] R. Chandra, *On $n|1|\overline{F}$ dynamic deterministic problems*, Naval Res. Logist., 26 (1979), pp. 537–544.

[5] C. Chu, *Efficient heuristics to minimize total flow time with release dates*, Oper. Res. Lett., 12 (1992), pp. 321–330.

[6] C. Chu, *A branch-and-bound algorithm to minimize total flow time with unequal release dates*, Naval Res. Logist., 39 (1992), pp. 859–875.

[7] J.S. Deogun, *On scheduling with ready times to minimize mean flow time*, Comput. J., 26 (1983), pp. 320–328.

[8] M.I. Dessouky and J.S. Deogun, *Sequencing jobs with unequal ready times to minimize mean flow time*, SIAM J. Comput., 10 (1981), pp. 192–202.

[9] J. Du, J.Y.T. Leung, and G.H. Young, *Minimizing mean flow time with release time constraint*, Theoret. Comput. Sci., 75 (1990), pp. 347–355.

[10] M.E. Dyer and L.A. Wolsey, *Formulating the single machine sequencing problem with release dates as a mixed integer program*, Discrete Appl. Math., 26 (1990), pp. 255–270.

[11] M.R. Garey and D.S. Johnson, *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.

[12] P.G. Gazmuri, *Probabilistic analysis of a machine scheduling problem*, Math. Oper. Res., 10 (1985), pp. 328–339.

[13] A.M.A. Hariri and C.N. Potts, *An algorithm for single machine sequencing with release times to minimize total weighted completion time*, Discrete Appl. Math., 5 (1983), pp. 99–109.

[14] J. Labetoulle, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, *Preemptive scheduling of uniform machines subject to release dates*, in Progress in Combinatorial Optimization, Academic Press, New York, 1984, pp. 245–261.

[15] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker, *Complexity of machine scheduling problems*, Ann. Discrete Math., 1 (1977), pp. 343–362.

[16] S. Leonardi and D. Raz, *Approximating total flow time on parallel machines*, in Proc. 29th Annual ACM Symposium on the Theory of Computing, El Paso, TX, 1997.

[17] W. Mao and A. Rifkin, *On-line algorithms for a single machine scheduling problem,* in The Impact of Emerging Technologies on Computer Science and Operations Research, S. Nash and A. Sofer, eds., Kluwer Academic Publishers, Norwell, MA, 1995, pp. 157–173.

[18] M.E. Posner, *Minimizing weighted completion times with deadlines*, Oper. Res., 33 (1985), pp. 562–574.

[19] W.E. Smith, *Various optimizers for single-state production*, Naval Res. Logist. Quart., 3 (1956), pp. 56–66.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1167–1181

FAST ESTIMATION OF DIAMETER AND SHORTEST PATHS (WITHOUT MATRIX MULTIPLICATION)*

D. AINGWORTH†, C. CHEKURI†, P. INDYK†, AND R. MOTWANI†

Abstract. In the recent past, there has been considerable progress in devising algorithms for the all-pairs shortest paths (APSP) problem running in time significantly smaller than the obvious time bound of $O(n^3)$. Unfortunately, all the new algorithms are based on fast matrix multiplication algorithms that are notoriously impractical. Our work is motivated by the goal of devising purely combinatorial algorithms that match these improved running times. Our results come close to achieving this goal, in that we present algorithms with a small additive error in the length of the paths obtained. Our algorithms are easy to implement, have the desired property of being combinatorial in nature, and the hidden constants in the running time bound are fairly small.

Our main result is an algorithm which solves the APSP problem in unweighted, undirected graphs with an *additive* error of 2 in time $O(n^{2.5}\sqrt{\log n})$. This algorithm returns actual paths and not just the distances. In addition, we give more efficient algorithms with running time $O(n^{1.5}\sqrt{k \log n} + n^2 \log^2 n)$ for the case where we are only required to determine shortest paths between k *specified* pairs of vertices rather than all pairs of vertices. The starting point for all our results is an $O(m\sqrt{n \log n})$ algorithm for distinguishing between graphs of diameter 2 and 4, and this is later extended to obtaining a ratio 2/3 approximation to the diameter in time $O(m\sqrt{n \log n} + n^2 \log n)$. Unlike in the case of APSP, our results for approximate diameter computation can be extended to the case of *directed* graphs with *arbitrary* positive real weights on the edges.

Key words. diameter, shortest paths, matrix multiplication

AMS subject classifications. 05C12, 05C50, 05C85, 68Q20

PII. S0097539796303421

1. Introduction.

Consider the problem of computing APSP in an unweighted, undirected graph G with n vertices and m edges. The recent work of Alon, Galil, and Margalit [2], Alon, Galil, Margalit, and Naor [3], and Seidel [16] has led to dramatic progress in devising fast algorithms for this problem. These algorithms are based on formulating the problem in terms of matrices with *small integer* entries and using fast matrix multiplications. They achieve a time bound of $\tilde{O}(n^\omega)$[1] where ω denotes the exponent in the running time of the matrix multiplication algorithm used. The current best matrix multiplication algorithm is due to Coppersmith and Winograd [9] and has $\omega = 2.376$. In contrast, the naive algorithm for APSP performs breadth-first searches (BFSs) from each vertex and requires time $\Theta(nm)$.

Given the fundamental nature of this problem, it is important to consider the desirability of implementing the algorithms in practice. Unfortunately, fast matrix multiplication algorithms are far from being practical and suffer from large hidden constants in the running time bound. Consequently, we adopt the view of treating

*Received by the editors May 13, 1996; accepted for publication (in revised form) June 16, 1997; published electronically March 22, 1999.

http://www.siam.org/journals/sicomp/28-4/30342.html

†Department of Computer Science, Stanford University, Stanford, CA 94305-9045 (donald@cs.stanford.edu, chekuri@cs.stanford.edu, indyk@cs.stanford.edu, rajeev@cs.stanford.edu). The first author was supported by an NSF Graduate Fellowship and NSF grant CCR-9357849. The second and third authors were supported by an OTL grant and NSF grant CCR-9357849. The fourth author was supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Development Award, an OTL grant, and NSF Young Investigator award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

[1]The notation $\tilde{O}(f(n))$ denotes $O(f(n)\,\text{polylog}\,(n))$.

these results primarily as indicators of the existence of efficient algorithms and consider the question of devising a purely *combinatorial algorithm* for APSP that runs in time $O(n^{3-\epsilon})$. The (admittedly vague) term "combinatorial algorithm" is intended to contrast with the more algebraic flavor of algorithms based on fast matrix multiplication. To understand this distinction, the reader may find it instructive to try and interpret the "algebraic" algorithms in purely graph-theoretic terms even with the use of the simpler matrix multiplication algorithm of Strassen [17]. The best known combinatorial algorithm, due to Feder and Motwani [12], runs in $O(n^3/\log n)$ time, yielding only a marginal improvement over the naive algorithm.

We take a step in the direction of realizing the goals outlined above by presenting an algorithm which solves the APSP problem with an *additive* error of 2 in time $O(n^{2.5}\sqrt{\log n})$. This algorithm returns actual paths and not just the distances. Note that the running time is better than the $\tilde{O}(n^{2.81})$ time bound of the more practical matrix multiplication algorithm of Strassen [17]. Further, as explained below, we also give slightly more efficient algorithms (for *sparse* graphs) for approximating the diameter. Our algorithms are easy to implement, have the desired property of being combinatorial in nature, and the hidden constants in the running time bound are fairly small. Our additive approximations are presented only for the case of unweighted, undirected graphs, but they can be easily generalized to the case of undirected graphs with small integer edge weights. In addition, we give a more efficient algorithm with running time $O(n^{1.5}\sqrt{k \log n} + n^2 \log^2 n)$ for the case where we are only required to determine shortest paths between k specified pairs of vertices rather than all pairs of vertices.

A crucial step in the development of our result was the shift of focus to the problem of computing the diameter of a graph. This is the maximum over all pairs of vertices of the shortest path distance between the vertices. The diameter can be determined by computing APSP distances in the graph, and it appears that this is the only known way to solve the diameter problem. In fact, Fan Chung [6] had earlier posed the question of whether there is an $O(n^{3-\epsilon})$ algorithm for finding the diameter without resorting to fast matrix multiplication. The situation with regard to combinatorial algorithms for diameter is only marginally better than in the case of APSP. Basch, Khanna, and Motwani [4] presented a combinatorial algorithm that verifies whether a graph has diameter 2 in time $O\left(n^3/\log^2 n\right)$. A slight adaptation of this algorithm yields a Boolean matrix multiplication algorithm which runs in the same time bound, thereby allowing us to verify that the diameter of a graph is d, for any constant d, in $O\left(n^3/\log^2 n\right)$ time.

Consider the problem of devising a fast algorithm for *approximating* the diameter. It is easy to estimate the diameter within a ratio 1/2 in $O(m)$ time: perform a BFS from any vertex v and let d be the depth of the BFS tree obtained; clearly, the diameter of G lies between d and $2d$. No better approximation algorithm was known for this problem; in fact, it was not even known how to distinguish between graphs of diameter 2 and 4. Our first result is an $O(m\sqrt{n \log n})$ algorithm for distinguishing between graphs of diameter 2 and 4, and this is later extended to obtaining a ratio 2/3 approximation to the diameter in time $O(m\sqrt{n \log n} + n^2 \log n)$. It should be noted that, unlike in the case of APSP, our results for approximate diameter computation can be extended to the case of *directed* graphs with *arbitrary* positive real weights on the edges.

The problem of computing approximate shortest paths has been considered earlier in the literature but purely from the point of view of multiplicative errors in the

approximation. Awerbuch, Berger, Cowen, and Peleg [1] and Cohen [7] have presented efficient algorithms for computing t-stretch paths for $t \geq 4$, where a path is said to have stretch t if its length is at most t times the length of the shortest path between its endpoints. Cohen [8] gave an algorithm that approximates paths from s sources to all other nodes in a weighted graph in time $O((m + sn)n^{\epsilon})$ for any $\epsilon > 0$. This algorithm outputs paths of length $(1 + O(1/\operatorname{polylog} n))d(u, v) + O(w_{max} \operatorname{polylog} n)$, where $d(u, v)$ denotes the distance between vertices u and v, and w_{max} is the largest edge weight in the graph. Her algorithm may be specialized to the unweighted case to compute paths of length $(1 + \delta)d(u, v) + c$ for any $\delta > 0$ within the same time, where the constant c depends on ϵ and δ.

The rest of this paper is organized as follows. We begin in section 2 by presenting some definitions and an algorithm for a version of the dominating set problem that underlies all our algorithms. In section 3, we describe the algorithms for distinguishing between graphs of diameter 2 and 4, and the extension to obtaining a ratio 2/3 approximation to the diameter. As we remarked earlier, these results can be applied to directed, weighted graphs. Then, in section 4, we apply the ideas developed in estimating the diameter to obtain the promised algorithm for an additive-error approximation for APSP. These ideas are extended in section 5 to obtain a more efficient algorithm for additive-error approximations to the k-pairs shortest paths problem. Finally, in section 6 we present an empirical study of the performance of our algorithm for APSP.

2. Preliminaries and a basic algorithm. We present some notation and a result concerning dominating sets in graphs. Initially, all definitions are with respect to some undirected, unweighted, connected graph $G(V, E)$ with n vertices and m edges. Later, we will point out the extension to directed and weighted graphs.

DEFINITION 2.1. *The* distance*, $d(u, v)$, between two vertices u and v is the length of the shortest path between them.*

DEFINITION 2.2. *The* diameter*, Δ, of a graph G is defined to be* $\max_{u,v \in G} d(u, v)$.

DEFINITION 2.3. *The k-neighborhood, $N_k(v)$, of a vertex v is the set of all vertices other than v that are at distance at most k from v, i.e.,*

$$N_k(v) = \{u \in V \mid 0 < d(v, u) \leq k\}.$$

The degree *of a vertex v is denoted by $d_v = |N_1(v)|$. Finally, we will use the notation $N(v) = N_1(v) \cup \{v\}$ to denote the set of vertices at distance at most 1 from v.*

It is important to keep in mind that the set $N(v)$ contains not just the neighbors of v but also includes v itself.

DEFINITION 2.4. *For any vertex $v \in V$, we denote by* $\textsc{b}(v)$ *the depth of a BFS tree in G rooted at the vertex v.*

Throughout this paper, we will be working with a parameter s to be chosen later that will serve as the threshold for classifying vertices as being of *low* degree or *high* degree. This threshold is implicit in the following definition.

DEFINITION 2.5. *Let $L(V) = \{u \in V \mid d_u < s\}$ and $H(V) = V \setminus L(V) = \{u \in V \mid d_u \geq s\}$.*

The following is a generalization of the standard notion of a dominating set.

DEFINITION 2.6. *Given a set $A \subseteq V$, a set $D \subseteq V$ is a* dominating set for *A if and only if for each vertex $v \in A$, $N(v) \cap D \neq \emptyset$. That is, for each vertex in $A \setminus D$, one of its neighbors is in D.*

The following theorem underlies all our algorithms.

THEOREM 2.7. *There exists a dominating set for $H(V)$ of size $O(s^{-1}n\log n)$, and such a dominating set can be found in $O(m+ns)$ time.*

REMARK 2.8. *It is easy to see that choosing a set of $\Theta(s^{-1}n\log n)$ vertices uniformly at random gives the desired dominating set for $H(V)$ with high probability. This construction in the proof of this theorem is in effect a derandomization of this randomized algorithm.*

Proof. Suppose, to begin with, that $H(V) = V$; then we are interested in the standard dominating set for the graph G. The problem of computing a minimum dominating set for G can be reformulated as a set cover problem as follows: for every vertex v create a set $S_v = N(v)$. This gives an instance of the set cover problem $\mathcal{S} = \{S_v \mid v \in V\}$, where the goal is to find a minimum cardinality collection of sets whose union is V. Given any set cover solution $\mathcal{C} \subseteq \mathcal{S}$, the set of vertices corresponding to the subsets in $\mathcal{C}$ forms a dominating set for G of the same size as $\mathcal{C}$. This is because each vertex v occurs in one of the sets $S_w \in \mathcal{C}$ and thus is either in the dominating set itself or has a neighbor therein. Similarly, any dominating set for G corresponds to a set cover for $\mathcal{S}$ of the same cardinality.

The greedy set cover algorithm repeatedly chooses the set that covers the most uncovered elements, and it is known to provide a set cover of size within a factor $\log n$ of the *optimal fractional solution* [13, 15]. Since every vertex has degree at least s and therefore the corresponding set S_v has cardinality at least s, assigning a weight of $1/s$ to every set in $\mathcal{S}$ gives a fractional set cover of total weight (fractional size) equal to $s^{-1}n$. Thus, the optimal *fractional* set cover size is $O(n/s)$, and the greedy set cover algorithm must then deliver a solution of size $O(s^{-1}n\log n)$. This gives a dominating set for G of the same size. If we implement the greedy set cover algorithm by keeping the sets in buckets sorted by the number of uncovered vertices, the algorithm can be shown to run in time $O(m)$.

Consider now the case where $H(V) \neq V$. Construct a graph $G' = (V', E')$, adding a set of dummy vertices $X = \{x_i \mid 1 \leq i \leq s\}$, as follows: define $V' = V \cup X$ and $E' = E \cup \{(x_i, x_j) \mid 1 \leq i < j \leq s\} \cup \{(u, x_i) \mid u \in L(V)\}$. Every vertex in this new graph has degree s or higher, so by the preceding argument we can construct a dominating set for G' of size $O(s^{-1}(n+s)\log(n+s)) = O(s^{-1}n\log n)$. Since none of the new vertices in X are connected to the vertices in $H(V)$, the restriction of this dominating set to V will give a dominating set for $H(V)$ of size $O(s^{-1}n\log n)$. Finally, the running time is increased by the addition of the new vertices and edges, but since the total number of edges added is at most $ns + s^2 = O(ns)$, we get the desired time bound. ∎

2.1. Extension to directed graphs. We briefly indicate the extension of the preceding definitions, notation, and observations to directed graphs. Given a directed graph $G(V, E)$, we will denote by $\overleftarrow{G}$ the graph obtained from G by *reversing* the direction of all the edges of G.

We use $\rightarrow$ and $\leftarrow$ to overline quantities defined with respect to G and $\overleftarrow{G}$, respectively. We will use the term *degree* to refer to the *out-degree* of a vertex, and for $v \in V$ we will denote its degree by $\overrightarrow{d}_v$. The definitions of distance, diameter, neighborhoods, BFS tree, and dominating set given earlier extend naturally to directed graphs as described below. We give the definitions only for G, and definitions for $\overleftarrow{G}$ can be obtained similarly.

DEFINITION 2.9. *For any two vertices $u, v \in V$, we define $d(u, v)$ as the length of the shortest path from u to v. If no such path exists, we assume $d(u, v) = \infty$.*

Algorithm 2-vs-4

1. **if** $\vec{L}(V) \neq \emptyset$ **then**
   (a) **choose** $v \in \vec{L}(V)$
   (b) **compute** a $\overrightarrow{BFS}$ tree from v and a $\overleftarrow{BFS}$ tree from each of the vertices in $\vec{N}(v)$
2. **else**
   (a) **compute** an out-dominating set D for $\vec{H}(V) = V$
   (b) **compute** a $\overrightarrow{BFS}$ tree from each of the vertices in D
3. **endif**
4. **if** all BFS trees have depth 2 **then return** 2
   else return 4.

FIG. 3.1. *Algorithm* 2-*vs*-4.

Note that $d(u, v)$ is not symmetric in general.

DEFINITION 2.10. *The* diameter Δ *of a graph* G *is defined to be* $\max_{u,v \in G} d(u, v)$.

DEFINITION 2.11. *Let* $\vec{N}_k(v) = \{u \in V \mid 0 < d(v, u) \leq k\}$. *Further,* $\vec{N}(v) = \vec{N}_1(v) \cup \{v\}$ *denotes the set of vertices at distance at most* 1 *from* v.

DEFINITION 2.12. $\overrightarrow{BFS}$ *is a BFS tree in the directed graph* G. *For any vertex* $v \in V$, *we denote by* $\vec{\mathrm{B}}(v)$ *the depth of a* $\overrightarrow{BFS}$ *tree in* G *rooted at the vertex* v.

We define $\vec{H}(V)$, $\vec{L}(V)$, and dominating set for directed graphs with respect to out-going edges incident at the vertices.

DEFINITION 2.13. *For some* s, *let* $\vec{L}(V) = \{u \in V \mid \vec{d}_u < s\}$ *and* $\vec{H}(V) = V \setminus \vec{L}(V) = \{u \in V \mid \vec{d}_u \geq s\}$.

DEFINITION 2.14. *Given a set* $A \subseteq V$, *a set* $D \subseteq V$ *is an* out-dominating set for A *if and only if for each vertex* $v \in A$, $\vec{N}(v) \cap D \neq \emptyset$.

The following is an easy consequence of Theorem 2.7.

COROLLARY 2.15. *Given a directed graph* $G(V, E)$, *there exists an out-dominating set for* $\vec{H}(V)$ *of size* $O(s^{-1} \log n)$, *and such a dominating set can be found in* $O(m+ns)$ *time.*

3. Estimating the diameter. In this section we will develop an algorithm to find an estimate E such that $\frac{2}{3}\Delta \leq E \leq \Delta$. We first present an algorithm for distinguishing between graphs of diameter 2 and 4. It is then shown that this algorithm generalizes to the promised approximation algorithm.

3.1. Distinguishing diameter 2 from 4. The basic idea behind the algorithm is rooted in the following lemma whose proof is straightforward.

LEMMA 3.1. *Suppose that* G *has a pair of vertices* a *and* b *with* $d(a, b) \geq 4$. *Then, any* $\overrightarrow{BFS}$ *tree rooted at a vertex* $v \in \vec{N}(a)$ *and any* $\overleftarrow{BFS}$ *tree rooted at a vertex* $v \in \overleftarrow{N}(b)$ *will have depth at least* 3.

The algorithm shown in Figure 3.1, called Algorithm 2-vs-4, computes BFS trees from a small set of vertices that is guaranteed to contain such a vertex, and so one of these BFS trees will certify that the diameter is more than 2.

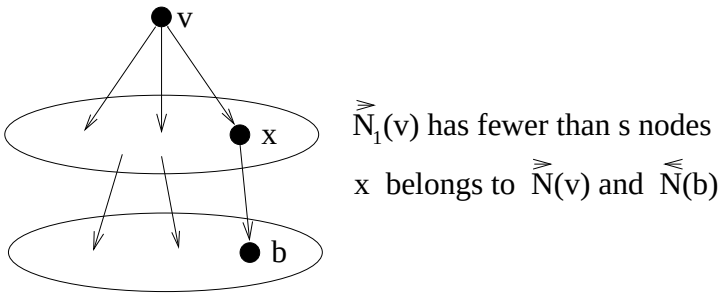


FIG. 3.2. *Case* 1 *in Algorithm* 2-*vs*-4.

We are assuming here that the sets $\overrightarrow{L}(V)$ and $D(V)$ are provided as a part of the input; otherwise, they can be computed in $O(m + ns)$ time.

THEOREM 3.2. *Algorithm* 2-*vs*-4 *distinguishes graphs of diameter* 2 *and* 4, *and it has running time* $O(ms^{-1}n \log n + ms)$.

Proof. It is clear that the algorithm outputs 2 for graphs of diameter 2 since in such graphs no BFS tree can have depth exceeding 2. Assume then that G has diameter 4 and fix any pair of vertices $a, b \in V$ such that $d(a, b) \geq 4$. We will show that the algorithm performs a properly directed BFS from a vertex $v \in \overrightarrow{N}(a) \cup \overleftarrow{N}(b)$. Since, by Lemma 3.1, the depth of the BFS tree rooted at v is at least 3, the algorithm will output 4.

We consider the two cases that can arise in the algorithm.

Case 1. $[\overrightarrow{L}(V) \neq \emptyset]$.

If b belongs to $\overrightarrow{N}(v)$, then there is nothing to prove. If $\overrightarrow{\mathrm{B}}(v) > 2$, then again we have nothing to prove. Note that if $\overrightarrow{\mathrm{B}}(v) = 1$, then $\overrightarrow{d}_v = n - 1$ and our choice of $s = o(n)$ would imply that $v \notin \overrightarrow{L}(V)$ which would be a contradiction. Therefore, the only case that remains is when $\overrightarrow{\mathrm{B}}(v) = 2$ and $d(v, b) = 2$ (see Figure 3.2). These assumptions imply that $\overrightarrow{N}(v) \cap \overleftarrow{N}(b) \neq \emptyset$, and Lemma 3.1 completes the proof. The size of $\overrightarrow{N}(v)$ is at most s; therefore, the time to compute the BFS trees is bounded by $O(ms)$.

Case 2. $[\overrightarrow{L}(V) = \emptyset]$.

Since D is an out-dominating set for V, it follows immediately that $D \cap \overrightarrow{N}(a) \neq \emptyset$, establishing the proof of correctness. From Theorem 2.7, we have $|D| = O(s^{-1}n \log n)$, and this implies a bound of $O(ms^{-1}n \log n)$ on the cost of computing the BFS trees in this case. □

Choosing $s = \sqrt{n \log n}$, we obtain the following corollary.

COROLLARY 3.3. *Graphs of diameter* 2 *and* 4 *can be distinguished in* $O(m\sqrt{n \log n})$ *time.*

3.2. Approximating the diameter. The ideas used in Algorithm 2-vs-4 can be generalized to estimate the diameter for all directed graphs: fix any two vertices a and b for which $d(a, b) = \Delta$, where Δ is the diameter of the graph. Suppose we can a find a vertex v in $\overrightarrow{N}_{\Delta/3}(a)$ or v' in $\overleftarrow{N}_{\Delta/3}(b)$; then it is clear that $\overrightarrow{\mathrm{B}}(v) \geq \frac{2}{3}\Delta$ or $\overleftarrow{\mathrm{B}}(v') \geq \frac{2}{3}\Delta$, and we can use $\overrightarrow{\mathrm{B}}(v)$ or $\overleftarrow{\mathrm{B}}(v')$ as our estimate. As before, we will find a small set of vertices which is guaranteed to have a vertex in $\overrightarrow{N}_{\Delta/3}(a) \cup \overleftarrow{N}_{\Delta/3}(b)$. Then

Algorithm Approx-Diameter

1. **compute** an s-partial-$\overrightarrow{BFS}$ tree from each vertex in V
2. **let** w be the vertex with the maximum depth ($\overrightarrow{\text{PB}}(w)$) partial-$\overrightarrow{BFS}$ tree
3. **compute** a $\overrightarrow{BFS}$ tree from w and a $\overleftarrow{BFS}$ tree from each vertex in $P\overrightarrow{BFS}_s(w)$
4. **compute** a new graph $\widehat{G}$ from G by adding all edges of the form (v, u) where $u \in P\overrightarrow{BFS}_s(v)$
5. **compute** an out-dominating set D in $\widehat{G}$
6. **compute** a $\overrightarrow{BFS}$ tree from each vertex in D
7. **return** estimate E equal to the maximum depth of all BFS trees from Steps 3 and 6.

FIG. 3.3. *Algorithm approx-diameter.*

we can compute the BFS tree from each of these vertices and use the maximum of the depths of these trees as our estimate E. The reason for choosing the fraction $1/3$ will become apparent in the analysis of the algorithm. In what follows, it will simplify notation to assume that $\Delta/3$ is an integer; in general, though, our analysis needs to be modified to use $\lfloor \Delta/3 \rfloor$. Also, we assume that $\Delta \geq 3$, and it is easy to see that the case $\Delta \leq 2$ can be handled separately.

A key tool in the rest of our algorithms will be the notion of a *partial-BFS* defined in terms of a parameter k.

DEFINITION 3.4. *A k-partial-BFS tree is obtained by performing a BFS up to the point where exactly k vertices (not including the root) have been visited.*

LEMMA 3.5. *A k-partial-BFS tree can be computed in time $O(k^2)$.*

Proof. The number of edges examined for each vertex visited is bounded by k since the k-partial-BFS process is terminated when k distinct vertices have been examined. This implies that the total number of edges examined is $O(k^2)$ and that dominates the running time. □

Note that a k-partial-BFS tree contains the k vertices closest to the root but that this set is not uniquely defined due to the need to break ties, which is done arbitrarily. Typically, k will be clear from the context and we will not specify it explicitly.

DEFINITION 3.6. *Let $P\overrightarrow{BFS}_k(v)$ be the set of vertices visited by a k-partial-$\overrightarrow{BFS}$ from v. Denote by $\overrightarrow{\text{PB}}(v)$ the depth of the tree constructed in this fashion. $P\overleftarrow{BFS}$ and $\overleftarrow{\text{PB}}(v)$ are defined similarly.*

Consider now the formal description of the approximation algorithm for diameter, algorithm approx-diameter, as shown in Figure 3.3.

The following lemmas constitute the analysis of this algorithm.

LEMMA 3.7. *The dominating set D found in step 5 is of size $O(s^{-1}n \log n)$.*

Proof. In $\widehat{G}$, each vertex $v \in V$ is adjacent to all vertices in $P\overrightarrow{BFS}_s(v)$ with respect to the graph G. Since $|P\overrightarrow{BFS}_s(v)| = s$ for every vertex v, the out-degree of each vertex in $\widehat{G}$ is at least s. From Theorem 2.7, it follows that we can find a dominating set of size $O(s^{-1}n \log n)$. □

LEMMA 3.8. *If $|\vec{N}_{\Delta/3}(v)| \geq s$ for all $v \in V$, then $D \cap (\vec{N}_{\Delta/3}(v) \cup \{v\}) \neq \emptyset$ for each vertex $v \in V$.*

Proof. Consider any particular vertex $v \in V$. If v is in D, then there is nothing

to prove. Otherwise, since D is a dominating set in $\widehat{G}$, there is a vertex $u \in D$ such that (v, u) is an edge in $\widehat{G}$. If (v, u) is in G, then again we are done since $u \in \vec{N}(v) \subset \vec{N}_{\Delta/3}(v)$. The other possibility is that u is not a neighbor of v in G, but then it must be the case that $u \in \overrightarrow{PBFS}_s(v)$. The condition $|\vec{N}_{\Delta/3}(v)| \geq s$ implies that $\overrightarrow{PBFS}_s(v) \subset \vec{N}_{\Delta/3}(v)$, which in turn implies that $u \in \vec{N}_{\Delta/3}(v)$, and hence $u \in D \cap \vec{N}_{\Delta/3}(v)$. □

The reader should notice the similarity between the preceding lemma and Case 2 in Theorem 3.2. Lemma 3.8 follows from the more general set cover ideas used in the proof of Theorem 2.7 and as such it holds even if we replace $\Delta/3$ by some other fraction of Δ. The more crucial lemma is given below.

LEMMA 3.9. *Let S be the set of vertices v such that $|\vec{N}_{\Delta/3}(v)| < s$. If $S \neq \emptyset$ then the vertex w found in step 2 belongs to S. In addition if $\vec{\mathrm{B}}(w) < \frac{2}{3}\Delta$, then for every vertex v,*

$$\overrightarrow{PBFS}_s(w) \cap \overleftarrow{N}_{\Delta/3}(v) \neq \emptyset.$$

Proof. It can be verified that for any vertex $u \in S$, $\overrightarrow{\mathrm{PB}}(u) > \Delta/3$; conversely, for any vertex v in $V \setminus S$, $\overrightarrow{\mathrm{PB}}(v) \leq \Delta/3$. From this we can conclude that if S is nonempty, then the vertex of largest depth belongs to S.

Also, for each vertex $u \in S$, we must have $\vec{N}_{\Delta/3}(u) \subset \overrightarrow{PBFS}_s(u)$. If $\vec{\mathrm{B}}(w) < \frac{2}{3}\Delta$, then every vertex is within a distance $\frac{2}{3}\Delta$ from w. From this and the fact that $\vec{N}_{\Delta/3}(w) \subset \overrightarrow{PBFS}_s(w)$, it follows that $\overrightarrow{PBFS}_s(w) \cap \overleftarrow{N}_{\Delta/3}(v) \neq \emptyset$. □

The proof of the above lemma makes clear the reason why our estimate is only within $\frac{2}{3}$ of the diameter. Essentially, we need to ensure that the Δ/k neighborhood of w intersects the Δ/k neighborhood of every other vertex. This can happen only if $\vec{\mathrm{B}}(w)$ is sufficiently small. If it is not small enough, we want $\vec{\mathrm{B}}(w)$ itself to be a good estimate. Balancing these conditions gives us $k = 3$ and the ratio $2/3$.

THEOREM 3.10. *Algorithm approx-diameter gives an estimate E such that $\frac{2}{3}\Delta \leq E \leq \Delta$ in time $O(ms + ms^{-1}n \log n + ns^2)$. Choosing $s = \sqrt{n \log n}$ gives a running time of $O(m\sqrt{n \log n} + n^2 \log n)$.*

Proof. The analysis is partitioned into two cases. Let a and b be two vertices such that $d(a, b) = \Delta$.

Case 1. (*For all vertices v, $|\vec{N}_{\Delta/3}(v)| \geq s$.*)

If either a or b is in D, we are done. Otherwise from the proof of Lemma 3.8, the set D has a vertex $v \in \vec{N}_{\Delta/3}(a)$. Since in step 6 we compute $\overrightarrow{BFS}$ trees from each vertex in D, one of these is v and $\vec{\mathrm{B}}(v)$ is the desired estimate.

Case 2. (*There exists a vertex $v \in V$ such that $|\vec{N}_{\Delta/3}(v)| < s$.*)

Let w be the vertex in step 2. If $\vec{\mathrm{B}}(w) \geq \frac{2}{3}\Delta$, $\vec{\mathrm{B}}(w)$ is our estimate and we are done. Otherwise from Lemma 3.9, $\overrightarrow{PBFS}_s(w)$ has a vertex $v \in \overleftarrow{N}_{\Delta/3}(b)$. Since in step 3 we compute $\overleftarrow{BFS}$ trees from each vertex in $\overrightarrow{PBFS}_s(w)$, one of these is v and $\overleftarrow{\mathrm{B}}(v)$ is the desired estimate.

The running time is easy to analyze. Each partial-BFS in step 1 takes at most $O(s^2)$ time by Lemma 3.5; thus, the total time spent on step 1 is $O(ns^2)$. Step 2 can be implemented in $O(n)$ time. In step 3, we compute BFS trees from s vertices, which

requires a total of $O(ms)$ time. The time required in step 4 is dominated by the time required to compute the partial-BFS trees in step 1. Theorem 2.7 implies that step 5 requires only $O(n^2 + ns)$ time (note that the graph $\widehat{G}$ could have many more edges than m). By Lemma 3.7, step 6 takes $O(ms^{-1}n \log n)$ time. Finally, the cost of step 7 is dominated by the cost of computing the various BFS trees in steps 3 and 6. The running time is dominated by the cost of steps 1, 3, and 6, and adding the bounds for these gives the desired result. □

3.3. Extension to weighted graphs. The algorithm for estimating the diameter extends to the case of weighted graphs as well, provided all edge weights are positive. This requires some minor modifications to algorithm approx-diameter that are listed below.

- The BFS is replaced by Dijkstra's algorithm [10] for shortest paths, and the depth of the tree now refers to the distance to the farthest vertex found so far.
- In forming the new graph $\widehat{G}$ in step 4 we need to remove all the original edges of G before we add the new edges. Note that $\widehat{G}$ is an unweighted graph.

The last modification is necessary because in a weighted graph it is not necessarily the case that a neighbor of a vertex v belongs to $N_{\Delta/3}(v)$. The running time remains the same because the time required by Dijkstra's algorithm (implemented with Fibonacci heaps [10]) is $O(m)$ when $m = \Omega(n \log n)$.

We obtain the following theorem.

THEOREM 3.11. *Given a directed graph with positive edge weights, there is an algorithm that gives an estimate E such that $\frac{2}{3}\Delta \leq E \leq \Delta$ in time $O(m\sqrt{n \log n} + n^2 \log n)$.*

4. Estimating APSP. We now turn to the problem of approximate APSP computations. We restrict ourselves to undirected and unweighted graphs for the rest of the paper, although it should be noted that there is an obvious extension of the results below to the case of undirected graphs with edge weights that are small integers.

It is possible to determine not only the diameter but the APSP distances to within an additive error of 2. The basic idea is that a dominating set, since it contains a neighbor of every vertex in the graph, must contain a vertex that is within distance 1 of any shortest path. Since we can only find a small dominating set for vertices in $H(V)$, we have to treat $L(V)$ vertices differently, but their low degree allows us to manage with only a partial-BFS, which we can combine with the information we have gleaned from the dominating set.

We give a detailed description of the approximate APSP algorithm, algorithm approx-APSP, in Figure 4.1. In Figure 4.2 we illustrate the main ideas behind this algorithm.

THEOREM 4.1. *In algorithm approx-APSP, for all vertices $u, v \in V$, the distances returned in $\widehat{d}$ satisfy the inequality*

$$0 \leq \widehat{d}(u, v) - d(u, v) \leq 2.$$

Further, the algorithm can be modified to produce paths of length $\widehat{d}$ rather than merely returning the approximate distances. This algorithm runs in time $O(n^2 s + n^3 s^{-1} \log n)$; choosing $s = \sqrt{n \log n}$ gives a running time of $O(n^{2.5}\sqrt{\log n})$.

Proof. We first show that the algorithm can be easily modified to return actual paths rather than only the distances. To achieve this, in steps 3 and 4 we can associate with each updated entry in the matrix the path from the BFS tree used for the update. In step 5, we merely concatenate the two paths from step 3 that determine the minimum value of $\widehat{d}$.

Algorithm Approx-APSP

Comment: Define $G[L(V)]$ to be the subgraph of G induced by $L(V)$.

1. **initialize** all entries in the distance matrix $\widehat{d}$ to ∞
2. **compute** a dominating set D for $H(V)$ of size $s^{-1}n\log n$
3. **compute** a BFS tree from each vertex $v \in D$, and update $\widehat{d}$ with the shortest path lengths for v so obtained
4. **compute** a BFS tree in $G[L(V)]$ for each vertex $v \in L(V)$, and update $\widehat{d}$ with the shortest path lengths for v so obtained
5. **for all** $u, v \in V \setminus D$ **do**

$$\widehat{d}(u,v) \leftarrow \min\{\widehat{d}(u,v), \min_{w\in D}\{\widehat{d}(w,u) + \widehat{d}(w,v)\}\}$$

6. **return** $\widehat{d}$ as the APSP matrix, and its largest entry as the diameter.

FIG. 4.1. *Algorithm approx-APSP.*

For a vertex u, it is clear that the shortest path distance to any vertex $v \in V$ that is returned cannot be smaller than the correct values, since they correspond to actual paths. To see that they differ by no more than 2, we need to consider three cases.

Case 1. $(u \in D)$.

In this case, the BFS tree from v is computed in step 3 and so clearly the distances returned are correct.

Case 2. $(u \in H(V) \setminus D)$.

By the definition of D, it must be the case that u has a neighbor w in D. Clearly, the distances from u and w to any other vertex cannot differ by more than 1, and the distances from w are always correct as per Case 1. The assignment in step 6 guarantees $\widehat{d}(u,v) \le \widehat{d}(w,u) + \widehat{d}(w,v) = d(w,v) + 1 \le d(u,v) + 2$.

Case 3. $(u \in L(V))$.

Fix any shortest path from u to v. Suppose that the path from u to v is entirely contained in $L(V)$; then $\widehat{d}(u,v)$ is set correctly in step 4. Otherwise, the path must contain a vertex $w \in H(V)$. If w is contained in D, then the correct distance is computed as per Case 1. Finally, if $w \in H(V) \setminus D$, then D contains a neighbor x of w. Clearly, in step 6, one of the possibilities considered will involve a path from u to x and a path from x to v. Since the distances involving x are correctly computed in step 3, this means that $\widehat{d}(u,v) \le d(x,u) + d(x,v) \le d(w,u) + d(w,v) + 2 = d(u,v) + 2$.

Finally, we analyze the running time of this algorithm. Step 1 requires only $O(n^2)$ time, and Theorem 2.7 implies that we can perform step 2 in the stated time bound. Step 3 requires $ms^{-1}n\log n$ for computing the BFS trees. Step 4 may compute as many as $\Omega(n)$ BFS trees, but $G[L(V)]$ only has $O(ns)$ edges and so this requires only $O(n^2 s)$ time. Finally, step 5 takes all n^2 vertex pairs and compares them with the $s^{-1}n\log n$ vertices in D. This implies the desired time bound. ∎

Although the error in this algorithm is 2, it can be improved for the special case of distinguishing diameter 2 from 4 based on the following two observations.

FACT 4.2. *If $u \in H(V)$ is at distance Δ from some vertex v, then $\widehat{d}(u,v) \le \Delta + 1$.*

Proof. Consider w, the vertex that dominates u. If the algorithm were to have set $\widehat{d}(u,v) > \Delta + 1$, then step 5 of the algorithm would imply $\widehat{d}(w,v) > \Delta$. Since $\widehat{d}$ is exact for vertices in D, this is not possible. ∎

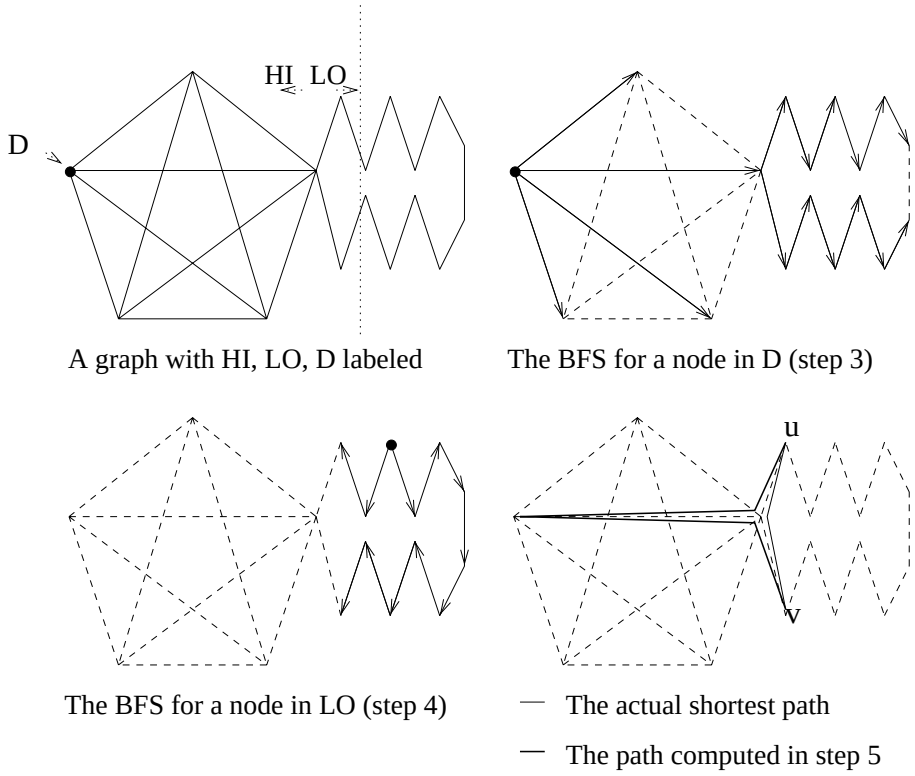


FIG. 4.2. *Illustration of algorithm approx-APSP.*

FACT 4.3. *If the algorithm reports for some* $u \in L(V)$ *that* $\text{B}(u) > 2$*, we can verify this in time* $O(ns)$ *per vertex.*

Thus, by performing a verification for each of the $L(V)$ vertices that report distance over 2, we can improve algorithm approx-APSP so that it always performs as well as the diameter approximation algorithms of the previous section. The first fact also appears to be useful in bringing the diameter error down to 1, but, unfortunately, the vertices in $L(V)$ cannot be handled as easily for larger diameters.

5. Estimating *k*-pairs shortest paths. In this section we consider the problem where we only seek to determine the distances between a given set of distinguished pairs of vertices denoted by P. We show that the algorithm approx-APSP can be generalized to handle this problem with the same error bounds. The generalized algorithm, called approx-kPSP, works in time $O(n^{1.5}\sqrt{k \log n} + n^2 \log^2 n)$, where k is the cardinality of the set P. When P contains all pairs of vertices, the behavior of approx-kPSP is identical to that of approx-APSP. However, for small k, the algorithm is significantly faster than approx-APSP.

The main idea behind the speed-up is the observation that the choice of $s = \sqrt{n \log n}$ is not optimal when we do not need to find the distances between all pairs. Step 5 of approx-APSP (the concatenation of paths) requires at most $O(kns^{-1} \log n)$ time (instead of $O(n^3 s^{-1} \log n)$), so the total time taken is $O(mns^{-1} \log n + n^2 s + kns^{-1} \log n)$. Now, the first term of the sum is not necessarily dominated by the last one. We have two cases: if $k \geq m$, the last term dominates the first but we get the

Algorithm approx-kPSP(i)

Comments:
define $s_i = \frac{n}{2^i}$ and $t = \log\left(\frac{n^{1.5}}{\sqrt{k \log n}}\right)$,
all $\widehat{d}(u,v)$ are initially equal to ∞ and $G_1 = (V_1, E_1)$ is set to G.

1. **if** $i = t$ **then** compute a BFS tree from each vertex $v \in V_i$ in G_i and update $\widehat{d}$ with the shortest path lengths obtained
2. **else let**
   $U_i = \{v \in V_i |$ degree of v in G_i is at least $s_i\}$,
   $V_{i+1} = V_i - U_i$, and
   G_{i+1} be the subgraph of G_i induced by V_{i+1}
   (a) **call** Approx-kPSP(i+1)
   (b) **compute** a dominating set D_i for U_i in G_i
   (c) **compute** a BFS tree from each $v \in D_i$ and update $\widehat{d}$ with the shortest path lengths obtained
   (d) **for each** $\{u,v\} \in P$ **do**

$$\widehat{d}(u,v) \leftarrow \min\{\widehat{d}(u,v), \min_{w \in D_i} \widehat{d}(w,u) + \widehat{d}(w,v)\}$$

3. **return**

FIG. 5.1. *Algorithm approx-kPSP.*

desired running time that depends on k when we balance the second and third terms; conversely, when $k < m$, we observe that the first term dominates the last. Note that the first term is the cost of performing BFS from all the dominating set vertices.

The intuition for the improvement comes from the following example: suppose that all vertices have degree d. Then we could take $s = d$, so $m = O(nd)$ and $|D| = O(\frac{n}{d} \log n)$, and the first term would be equal to $m|D| = O(n^2 \log n)$. This example shows that performing BFS from all the dominating set vertices is not expensive if the degrees are more or less uniform. Of course, in general, such an assumption is not true. However, we can exploit this observation by partitioning the vertex set into $O(\log n)$ classes, such that the ith class consists of vertices of degree between $\frac{n}{2^i}$ and $\frac{n}{2^{i-1}}$, and computing the dominating sets for each class separately. This effectively reduces the first term to $O(n^2 \operatorname{polylog}(n))$, and now we can balance the second and third terms as in the other case.

This algorithm, called approx-kPSP, is described in Figure 5.1. The algorithm is recursive and at the top level it is invoked with parameter value $i = 1$, assuming $G_1 = G$. The algorithms makes t recursive calls, where t is a parameter chosen so as to minimize the running time.

We begin the analysis by identifying the optimal choice of t.

LEMMA 5.1. *The parameter t can be chosen such that the running time of algorithm approx-kPSP is $O(n^{1.5}\sqrt{k \log n} + n^2 \log^2 n)$. This time is achieved for $t = \log(n^{1.5}/\sqrt{k \log n})$.*

Proof. Observe that for each i and $v \in V_i$, the degree of v in G_i is less than s_{i-1} (assume $s_0 = n$). This implies that $|E_i| = O(ns_{i-1})$. Let C_i denote the running time of the invocation of approx-kPSP with argument i. It is easy to see that $C_t = n|E_t|$ and that for $i < t$, C_i is dominated by the time required for steps 2(c) and 2(d) which

require $|D_i||E_i|$ and $k|D_i|$ time, respectively. The total time can now be estimated as follows:

$$\sum_{i=1}^{t} C_i = n|E_t| + \sum_{i=1}^{t-1} (|D_i||E_i| + k|D_i|) \leq \sum_{i=1}^{t-1} \frac{n}{s_i} n s_{i-1} \log n + k \sum_{i=1}^{t-1} \frac{n}{s_i} \log n + n^2 s_t$$
$$\leq 2n^2 \log^2 n + \frac{n}{s_t} k \log n + n^2 s_t.$$

Letting $t = \log(n^{1.5}/\sqrt{k \log n})$ we get $s_t = \Theta(\sqrt{(k \log n)/n}\,)$ which gives the desired bound on the running time of the algorithm. □

We can now complete the analysis of the algorithm.

THEOREM 5.2. *For all pairs $(u, v) \in P$, the distances returned in $\widehat{d}$ by approx-kPSP satisfy the inequalities:*

$$0 \leq \widehat{d}(u, v) - d(u, v) \leq 2.$$

The algorithm runs in time $O(n^2 \log^2 n + n^{1.5}\sqrt{k \log n})$.

Proof. Let $d_i(u, v)$ denote the distance between u and v in G_i. Clearly, it is sufficient to show by induction on i that $0 \leq \widehat{d}(u, v) - d_i(u, v) \leq 2$ after finishing approx-kPSP(i). The base case (for $i = t$) holds trivially since we compute the exact shortest paths. The proof of the inductive step is similar to the proof Theorem 4.1; hence we omit the details. The time bound follows from Lemma 5.1. □

5.1. Application: Randomized approximation scheme for diameter. Algorithm approx-kPSP can be used to obtain a randomized approximation scheme for the diameter of a graph. Let $u, v \in V$ be such that $d(u, v) = \Delta$. If we choose a vertex w uniformly at random from V, the probability of $d(u, w) \leq \frac{\epsilon}{2}\Delta$ is $\Theta(\frac{\epsilon\Delta}{n})$. This guarantees that a set P of $O(\frac{n^2}{\epsilon^2\Delta^2} \log n)$ vertices chosen uniformly at random contains vertices x, y such that $d(u, x) \leq \frac{\epsilon}{2}\Delta$ and $d(v, y) \leq \frac{\epsilon}{2}\Delta$; hence $d(x, y) \geq (1-\epsilon)\Delta$ with high probability. We can use algorithm approx-kPSP to approximate distances between all pairs of vertices in P in $O(n^{1.5}\sqrt{|P| \log n} + n^2 \log^2 n) = O(\frac{n^{2.5}}{\epsilon\Delta} \log n + n^2 \log^2 n)$ time. For large Δ, say, $\Delta = \Omega(n^\delta)$ for some $\delta > 0$, the improvement is significant. We obtain the following theorem.

THEOREM 5.3. *For any $0 < \epsilon < 1$ there exists a Monte Carlo algorithm which finds an estimate E such that $(1-\epsilon)\Delta \leq E \leq \Delta + 2$ in time $O(\frac{n^{2.5}}{\epsilon\Delta} \log n + n^2 \log^2 n)$.*

Note that while this randomized algorithm assumes knowledge of Δ, it is sufficient to provide it with a constant-factor approximation to Δ. The depth of a BFS tree rooted at an arbitrary vertex of G is a 2-approximation for Δ and can be used for this purpose.

6. Experimental results. To evaluate the usefulness of our algorithm, we ran it on two families of graphs and compared the results with a carefully coded algorithm based on BFS. The algorithm approx-APSP was tweaked with the following heuristic improvement to step 5 that avoids many needless iterations: when a node has a neighbor in D, we copy the distances of its neighbor (since they can differ by at most 1). This algorithm (called fast approx-APSP) occasionally has a higher fraction of incorrect entries but seems to be a faster way to solve the APSP problem.

The first family of graphs were random graphs from the $G_{n,m}$ model [5], which are graphs chosen uniformly at random from those with n vertices and m edges. In our experiments, we chose random graphs with n ranging from 10 to 1000, and $2m/n^2$

TABLE 6.1

Summary of experimental results. The speed-up numbers indicate the ratio of the execution time of the carefully coded BFS algorithm to that of the algorithms. The accuracy refers to the ratio of the total number of exact *entries in the distance matrix to the total number of entries in the matrix.*

		Approx-APSP		Fast approx-APSP	
		speed-up	accuracy	speed-up	accuracy
Random graphs	Median	0.52	0.39	5.30	0.51
	Average	0.63	0.39	4.75	0.55
	Std Dev	0.23	0.14	1.70	0.12
GraphBase	Median	0.59	0.69	3.95	0.53
	Average	2.44	0.72	10.18	0.47
	Std Dev	0.24	0.16	1.73	0.13

ranging from 0.03 to 0.90. On these graphs, fast approx-APSP runs about five times faster than the BFS implementation, and about half of the distances are off by one.

The second family of graphs comes from the Stanford GraphBase [14]. We tested all of the connected, undirected graphs from Appendix C in Knuth [14] (ignoring edge weights). This is a very heterogeneous family of graphs, including graphs representing highway connections for American cities, athletic schedules, five-letter English words, and expander graphs, as well as more combinatorial graphs. Thus the results here are quite indicative of practical performance. Although the BFS-based algorithm runs faster for certain subfamilies of the GraphBase, fast approx-APSP outperformed the other algorithms overall.

The results are summarized in Table 6.1. The speed-up numbers indicate the inverse of the ratio of the execution time of the algorithms to that of the carefully coded BFS algorithm. The accuracy refers to the ratio of the total number of *exact* entries in the distance matrix to the total number of entries in the matrix. In both of these families, the accuracy of approx-APSP could be improved by subtracting 1 in step 5. This did not seem necessary given that the BFS approach performed about as fast as approx-APSP, and that fast approx-APSP performed faster with roughly 50% accuracy. The numbers indicate that for general graphs where an additive factor error is acceptable, fast approx-APSP is the algorithm of choice, and for more specific families of graphs, the parameters can be adjusted for even better performance.

7. Conclusions and further work. Our work suggests several interesting directions for future work, the most elementary being the following: is there a combinatorial algorithm running in time $O(n^{3-\epsilon})$ for distinguishing between graphs of diameter 2 and 3? It is our belief that the problem of efficiently computing the diameter can be solved given such a decision algorithm, and our work provides some evidence in support of this belief. Subsequent to our work, Dor, Halperin, and Zwick [11] have shown that the problem of computing APSP with an additive error of at most 1 is as hard as Boolean matrix multiplication. This result still does not resolve the question of whether computing the diameter is easier and raises the interesting question of whether there is some strong equivalence between the diameter and APSP problems, e.g., that their complexity is the same within poly-logarithmic factors. Dor, Halperin, and Zwick have also extended our techniques to obtain tradeoffs between the additive error and the running time and to obtain approximate distances with multiplicative factors instead of additive factors. Finally, of course, removing the additive error from our results remains a major open problem.

Acknowledgments. We are grateful to Noga Alon for his comments and suggestions, and to Nati Linial for helpful discussions. We are also indebted to Edith Cohen for comments that helped us extend some of our results. Thanks also to Michael Goldwasser, David Karger, Sanjeev Khanna, and Eric Torng for their comments.

REFERENCES

[1] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg, *Near-linear cost sequential and distributed constructions of sparse neighborhood covers*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 638–647.

[2] N. Alon, Z. Galil, and O. Margalit, *On the exponent of the all pairs shortest path problem*, in Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, San Juan, PR, 1991, pp. 569–575.

[3] N. Alon, Z. Galil, O. Margalit, and M. Naor, *Witnesses for Boolean matrix multiplication and for shortest paths*, in Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, 1992, pp. 417–426.

[4] J. Basch, S. Khanna, and R. Motwani, *On Diameter Verification and Boolean Matrix Multiplication,* Report STAN-CS-95-1544, Department of Computer Science, Stanford University, Stanford, CA, 1995.

[5] B. Bollobás, *Random Graphs*. Academic Press, New York, 1985.

[6] Fan R. K. Chung, *Diameters of graphs: Old problems and new results*, Congr. Numer., 60 (1987), pp. 295–317.

[7] E. Cohen, *Fast algorithms for t-spanners and stretch-t paths*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, 1993, pp. 648–658.

[8] E. Cohen, *Polylog-time and near-linear work approximation scheme for undirected shortest paths*, Proceedings of 26th Annual ACM Symposium on Theory of Computing, 1994, pp. 16–26.

[9] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[10] T. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press and McGraw–Hill, New York, 1990.

[11] D. Dor, S. Halperin, and U. Zwick, *All pairs almost shortest paths*, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, Burlington, VT, 1996, pp. 452–461.

[12] T. Feder and R. Motwani, *Clique partitions, graph compression and speeding-up algorithms*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, 1991, pp. 123–133.

[13] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.

[14] D. E. Knuth, *The Stanford GraphBase: A platform for combinatorial computing*, Addison–Wesley, Reading, MA, 1993.

[15] L. Lovász, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.

[16] R. G. Seidel, *On the all-pairs-shortest-path problem*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, Victoria, BC, 1992, pp. 745–749.

[17] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1182–1197

CONSTRUCTING APPROXIMATE SHORTEST PATH MAPS IN THREE DIMENSIONS*

SARIEL HAR-PELED†

Abstract. We present a new technique for constructing a data structure that approximates shortest path maps in $\mathbb{R}^d$. By applying this technique, we get the following two results on approximate shortest path maps in $\mathbb{R}^3$.

(i) Given a polyhedral surface or a convex polytope $\mathcal{P}$ with n edges in $\mathbb{R}^3$, a source point s on $\mathcal{P}$, and a real parameter $0 < \varepsilon \leq 1$, we present an algorithm that computes a subdivision of $\mathcal{P}$ of size $O((n/\varepsilon)\log(1/\varepsilon))$ which can be used to answer efficiently approximate shortest path queries. Namely, given any point t on $\mathcal{P}$, one can compute, in $O(\log(n/\varepsilon))$ time, a distance $\Delta_{\mathcal{P},s}(t)$, such that $d_{\mathcal{P},s}(t) \leq \Delta_{\mathcal{P},s}(t) \leq (1+\varepsilon)d_{\mathcal{P},s}(t)$, where $d_{\mathcal{P},s}(t)$ is the length of a shortest path between s and t on $\mathcal{P}$.

The map can be computed in $O(n^2 \log n + (n/\varepsilon)\log(1/\varepsilon)\log(n/\varepsilon))$ time, for the case of a polyhedral surface, and in $O((n/\varepsilon^3)\log(1/\varepsilon) + (n/\varepsilon^{1.5})\log(1/\varepsilon)\log n)$ time if $\mathcal{P}$ is a convex polytope.

(ii) Given a set of polyhedral obstacles $\mathcal{O}$ with a total of n edges in $\mathbb{R}^3$, a source point s in $\mathbb{R}^3 \setminus \operatorname{int}\cup_{O\in\mathcal{O}} O$, and a real parameter $0 < \varepsilon \leq 1$, we present an algorithm that computes a subdivision of $\mathbb{R}^3$, which can be used to answer efficiently approximate shortest path queries. That is, for any point $t \in \mathbb{R}^3$, one can compute, in $O(\log(n/\varepsilon))$ time, a distance $\Delta_{\mathcal{O},s}(t)$ that ε-approximates the length of a shortest path from s to t that avoids the interiors of the obstacles. This subdivision can be computed in roughly $O(n^4/\varepsilon^6)$ time.

Key words. approximation algorithms, Euclidean shortest paths, Voronoi diagrams

AMS subject classifications. 68U05, 65D99, 52B05, 52B10

PII. S0097539797325223

1. Introduction.

The *three-dimensional Euclidean shortest path problem* is defined as follows: Given a set of pairwise-disjoint polyhedral objects in $\mathbb{R}^3$ and two points s and t, compute the shortest path between s and t which avoids the interiors of the given polyhedral "obstacles." This problem has received considerable attention in computational geometry. It was shown to be NP-hard by Canny and Reif [3], and the fastest available algorithms for this problem run in time that is exponential in the total number of obstacle vertices (which we denote by n) [20, 21]. The apparent intractability of the problem has motivated researchers to develop polynomial-time algorithms for computing approximate shortest paths and for computing shortest paths in special cases.

In the *approximate three-dimensional Euclidean shortest path* problem, we are given an additional parameter $\varepsilon > 0$, and the goal is to compute a path between s and t that avoids the interiors of the obstacles and whose length is at most $(1+\varepsilon)$ times the length of the shortest obstacle-avoiding path (we call such a path an *ε-approximate path*). Approximation algorithms for the three-dimensional shortest path problem were first studied by Papadimitriou [19], who gave an $O(n^4(L+\log(n/\varepsilon))^2/\varepsilon^2)$-time algorithm for computing an ε-approximate shortest path, where L is the number of bits used in each computation. A rigorous analysis of Papadimitriou's algorithm

*Received by the editors July 29, 1997; accepted for publication (in revised form) March 3, 1998; published electronically March 22, 1999. This work was supported by a grant from the U.S.–Israeli Binational Science Foundation. This work is part of the author's Ph.D. thesis, prepared at Tel Aviv University, Tel Aviv, Israel.

http://www.siam.org/journals/sicomp/28-4/32522.html

†School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel (sariel@math.tau.ac.il).

was recently given by Choi, Sellen, and Yap [6]. A different approach was taken by Clarkson [7], resulting in an algorithm with roughly $O\left(n^2/\varepsilon^4\right)$ running time (the precise result is stated in Theorem 4.2).

The problem of computing a shortest path between two points along the surface of a single convex polytope is an interesting special case of the three-dimensional Euclidean shortest path problem. Sharir and Schorr [22] gave an $O(n^3 \log n)$ algorithm for this problem, exploiting the property that a shortest path on a polyhedron *unfolds* into a straight line. Mitchell, Mount, and Papadimitriou [16] improved the running time to $O(n^2 \log n)$; their algorithm works for nonconvex polyhedra (or polyhedral surfaces) as well. Chen and Han [5] gave another algorithm with an improved running time of $O(n^2)$. It is a rather long-standing and intriguing open problem whether the shortest path on a convex polytope can be computed in subquadratic time. This has motivated the problem of finding near-linear algorithms that produce only an approximation of the shortest path. The first result in this direction is by Hershberger and Suri [12]. They present a simple algorithm that runs in $O(n)$ time and computes a path whose length is at most $2d_P(s,t)$. Using the algorithm of [12], Agarwal et al. [1] present a relatively simple algorithm that computes an ε-approximate shortest path (i.e., a path on ∂P between two points $s,t \in \partial P$ whose length is at most $(1+\varepsilon)d_P(s,t)$) for any prescribed $0 < \varepsilon \leq 1$, where the running time of the algorithm is $O(n \log(1/\varepsilon)+1/\varepsilon^3)$. In a companion paper [11], we present an improved algorithm, with $O(n)$ preprocessing time that answers two-point ε-approximate shortest path queries in $O((\log n)/\varepsilon^{1.5} + 1/\varepsilon^3)$ time for *any* pair of points $s,t \in \partial P$. Recently, Varadarajan and Agarwal [24] gave a subquadratic time algorithm that computes a constant approximation to the shortest path on a polyhedral terrain. Other recent works by Mata and Mitchell [14] and also by Lanthier, Maheshwari, and Sack [13] implement various heuristics for computing approximate shortest paths on weighted terrains (i.e., each face f is being assigned a weight w_f, such that the distance between any two points $a,b \in f$ is $w_f \cdot |ab|$). Those programs give satisfactory results in practice, which are within an order of magnitude better than their worst case analysis.

In this paper, extending our work in [11], we present a new general technique for constructing a data structure that one can use to answer ε-approximate shortest path queries, for a source point s and approximation factor $\varepsilon > 0$ fixed in advance. Using this technique, we solve two problems involving approximate shortest path maps in $\mathbb{R}^3$.

Approximate shortest path maps. The exact algorithms of [16, 22] receive as input a convex polytope or a polyhedral surface $\mathcal{P}$ and a fixed source point s on $\mathcal{P}$, and they compute a map (i.e., a subdivision of $\mathcal{P}$) of complexity $\Theta(n^2)$ that can be used to answer (exact) shortest path queries from s to any point on $\mathcal{P}$ (along $\mathcal{P}$) in $O(\log n)$ time (such a query reports the *length* of the shortest path; reporting the path itself might require more time). This shortest path map can be stored in linear space, for the case of a convex polytope, by using a persistent data structure; see [17]. However, the time required to compute this compact representation of the shortest path map is quadratic in the worst case. This raises the problem of computing a map of near-linear size for approximate shortest path queries from s. We show in section 3 that this is indeed possible: Given a polyhedral surface $\mathcal{P}$ with n edges in $\mathbb{R}^3$, a source point $s \in \mathcal{P}$, and a prescribed $0 < \varepsilon \leq 1$, there exists a map (a subdivision of $\mathcal{P}$) of complexity $O((n/\varepsilon)\log(1/\varepsilon))$, such that for any $t \in \mathcal{P}$, one can compute the length of an ε-approximate shortest path between s and t on $\mathcal{P}$ in $O(\log(n/\varepsilon))$ time by locating t in the map.

We present an algorithm that constructs such an approximation map in $O(n^2 \log n + (n/\varepsilon) \log(1/\varepsilon) \log(n/\varepsilon))$ time for the case of a polyhedral surface, and in $O((n/\varepsilon^3) \log(1/\varepsilon) + (n/\varepsilon^{1.5}) \log(1/\varepsilon) \log n)$ time for the case of a convex polytope. Note that if $\mathcal{P}$ is a convex polytope, then our previous result [11] provides an alternative structure with similar properties. However, the dependence of the query time on ε is much better in the method we present here.

Approximate spatial shortest path maps. In section 4, we present a similar result for ε-approximate shortest paths among polyhedral obstacles in $\mathbb{R}^3$. Let $\mathcal{O}$ be a set of polyhedral obstacles in $\mathbb{R}^3$ with a total of n edges, s a source point in $\mathbb{R}^3$, and $0 < \varepsilon \leq 1$ a parameter. We show that there exists a spatial subdivision $\mathcal{M}$ of $\mathbb{R}^3$ such that for any $t \in \mathbb{R}^3$ one can compute in $O(\log(n/\varepsilon))$ time the length of an ε-approximate shortest path between s and t that avoids the interiors of the obstacles by performing a spatial point-location query with t in $\mathcal{M}$. The space needed to compute and preprocess $\mathcal{M}$ for spatial point-location is $O(n^2/\varepsilon^{4+\delta})$, for any $\delta > 0$, and the preprocessing time is

$$O\left(\frac{n^4}{\varepsilon^2}\left(\frac{\beta(n)}{\varepsilon^4}\log\frac{n}{\varepsilon} + \log(n\rho)\log(n\log\rho)\right)\log\frac{1}{\varepsilon}\right),$$

where ρ is the ratio of the length of the longest edge in $\mathcal{O}$ to the Euclidean distance between s and t, $\beta(n) = \alpha(n)^{O(\alpha(n))^{O(1)}}$, and $\alpha(n)$ is the extremely slowly growing inverse of the Ackermann function. This algorithm uses the algorithm of Clarkson [7] that computes an ε-approximate shortest path between two given points in $O\left(\frac{n^2}{\varepsilon^4}\beta(n)\log\frac{n}{\varepsilon} + n^2\log(n\rho)\log(n\log\rho)\right)$ time.

The paper is organized as follows. Section 2 introduces the notion of a *distance function* and shows how to compute a "small-size" additive weighted Voronoi diagram that enables us to ε-approximate this function. We present two applications of this result. In section 3, we present the algorithm mentioned above for constructing a map for approximate shortest path queries from a fixed source on a convex polytope or a polyhedral surface in $\mathbb{R}^3$. In section 4 we present the algorithm mentioned above for constructing a spatial subdivision for approximate shortest path queries from a fixed source among polyhedral obstacles in $\mathbb{R}^3$. We conclude in section 5 by mentioning a few open problems.

2. Approximating a distance function by a weighted Voronoi diagram.

In this section, we introduce the notion of a distance function and show how to compute a "small-size" additive weighted Voronoi diagram that approximates the function up to a factor of $1 + \varepsilon$. We use this result in sections 3 and 4 to derive our two main results.

DEFINITION 2.1. *Let $\mathcal{I}$ be a subset of $\mathbb{R}^d$. A function $f : \mathcal{I} \to \mathbb{R}$ is a* distance function *on $\mathcal{I}$ if*

(i) $f(x) + f(y) \geq |xy|$ *for any $x, y \in \mathcal{I}$,*
(ii) $f(x) + |xy| \geq f(y)$ *for any straight segment $xy \subseteq \mathcal{I}$,*

where $|xy|$ denotes the Euclidean distance between x and y.

Thus, a distance function has to satisfy two types of triangle inequalities. Since these inequalities are satisfied by the Euclidean distance from any fixed point, a distance function can be regarded as a certain generalization of the Euclidean distance.

Example 2.1. Figure 2.1 illustrates some of the geometric restrictions imposed on a univariate distance function.

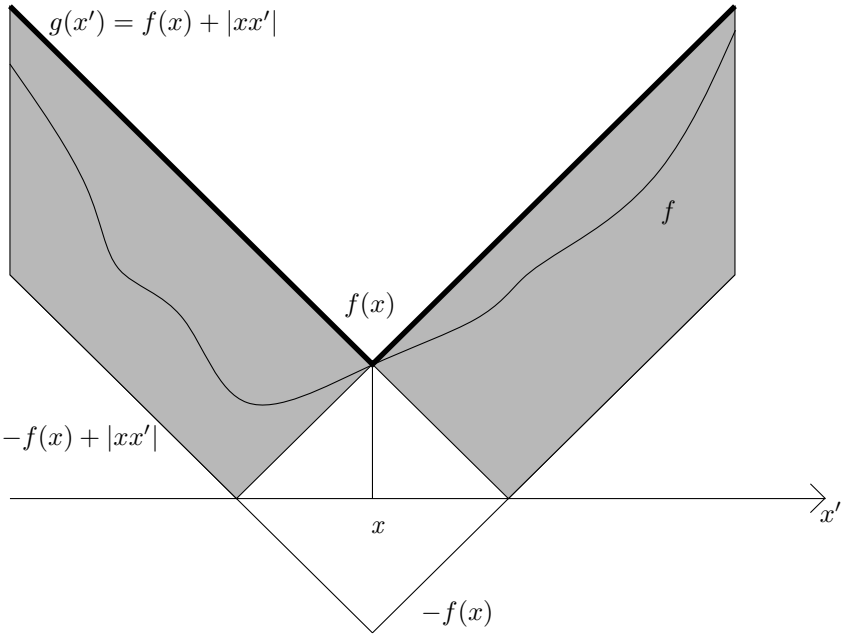


FIG. 2.1. *The graph of the distance function f must lie inside the gray area. This implies that the function* $g(x') = f(x) + |xx'|$ *approximates* $f(x')$ *"well" in a small neighborhood of* x *and for sufficiently large values of* x'.

Example 2.2. (i) Let $\mathcal{P}$ be a polyhedral surface in $\mathbb{R}^3$, and let s be a source point on $\mathcal{P}$. For any $t \in \mathcal{P}$, let $d_{\mathcal{P},s}(t)$ denote the length of a shortest path between s and t on $\mathcal{P}$. It is easy to verify that $d_{\mathcal{P},s}(t)$ is a distance function on $\mathcal{P}$.

(ii) Let $\mathcal{O}$ be a collection of pairwise-disjoint polyhedral obstacles in $\mathbb{R}^3$, and let s be a point in $FP(\mathcal{O}) = \mathbb{R}^3 \setminus \cup_{O \in \mathcal{O}} \text{int}O$. Let $FP(\mathcal{O}, s)$ denote the connected component of $FP(\mathcal{O})$ that contains s (i.e., the set of all the points in $\mathbb{R}^3$ that can be connected to s by a path that avoids the interiors of the obstacles of $\mathcal{O}$).

For any $t \in FP(\mathcal{O}, s)$, we denote by $d_{\mathcal{O},s}(t)$ the length of a shortest path between s and t that avoids the interior of the obstacles of $\mathcal{O}$. Clearly, $d_{\mathcal{O},s}$ is a distance function on $FP(\mathcal{O}, s)$.

DEFINITION 2.2. *A pair* $\mathcal{S} = (S, w)$ *is a* weighted set of points *if* $S = \{p_1, \ldots, p_m\}$ *is a finite set of points in* $\mathbb{R}^d$, *and* $w(\cdot)$ *is a function assigning nonnegative weights to the points of S. We define the distance of a point p from the point* p_i *to be* $V_{(p_i, w(p_i))}(p) = |pp_i| + w(p_i)$. *We define* $V_{\mathcal{S}}(p) = \min_{i=1}^{m} V_{(p_i, w(p_i))}(p)$. *The function* $V_{\mathcal{S}}(p)$ *induces a natural subdivision* $\mathcal{V}_{\mathcal{S}}$ *of* $\mathbb{R}^d$ *into cells, known as the (additive)* weighted Voronoi diagram *of* $\mathcal{S}$, *such that the ith cell is the locus of all points closest to* p_i *in this distance function. As is well known, in the planar case* $\mathcal{V}_{\mathcal{S}}$ *has complexity* $O(m)$, *and it can be computed in* $O(m \log m)$ *time (see* [10]).

Remark 2.1. For $d \geq 3$, the complexity of an additive weighted Voronoi diagram of m points in $\mathbb{R}^d$ is $O\left(m^{\lfloor d/2 \rfloor + 1}\right)$. This follows by reducing the computation of the diagram to the computation of a convex hull in $d + 2$ dimensions. Furthermore, we can compute the diagram in $O(m^{\lfloor d/2 \rfloor + 1})$ time (see [2]).

We next show how to approximate a distance function $f(\cdot)$ by a weighted Voronoi diagram. First, we compute a global minimum p_0 of f. As illustrated in Figure 2.1, the function $V_{(p_0, f(p_0))}(p)$ approximates $f(p)$ "well" for p sufficiently close to, or

sufficiently far from, p_0. In other words, f is well approximated in these regions by the weighted Voronoi diagram of the single site p_0 with weight $f(p_0)$. By adding extra sites to the diagram, we can make the distance induced by the resulting diagram an ε-approximation to f everywhere, as will be shown next.

DEFINITION 2.3. *Given a point $p \in \mathbb{R}^d$, and $r \geq 0$, let $B(p,r)$ denote the closed ball of radius r centered at p, and let $\overline{B}(p,r)$ denote the set $\mathbb{R}^d \setminus B(p,r)$. For $r' > r$, let $\mathcal{A}(p,r,r')$ denote the annulus (or shell) $B(p,r') \setminus B(p,r)$.*

The following sequence of technical lemmas provide the basis for approximating a given distance function by a weighted Voronoi diagram. The following lemmas are stated for arbitrary dimension d. We will apply them with $d = 1, 2$, or 3.

LEMMA 2.4. *Let $\mathcal{I}$ be a convex subset of $\mathbb{R}^d$, f a distance function defined over $\mathcal{I}$, and $\mathcal{S} = (S, w)$ a weighted set such that $S \subseteq \mathcal{I}$ and $f(x) \leq w(x)$ for all $x \in S$. Then $f(t) \leq V_{\mathcal{S}}(t)$ for all $t \in \mathcal{I}$.*

Proof. Let t be any point of $\mathcal{I}$, and let x denote the point of S realizing $V_{\mathcal{S}}(t)$. Then $V_{\mathcal{S}}(t) = w(x) + |tx| \geq f(x) + |tx| \geq f(t)$. ∎

Formalizing the intuition behind Figure 2.1, we have the following.

LEMMA 2.5. *Let $\mathcal{I}$ be a convex subset of $\mathbb{R}^d$, f a distance function defined over $\mathcal{I}$, $0 < \varepsilon \leq 1$ a parameter, p a point in $\mathcal{I}$, and w a real number such that $f(p) \leq w \leq (1+\varepsilon/8)f(p)$. Then $f(t) \leq V_{(p,w)}(t) \leq (1+\varepsilon)f(t)$ for all t in*

$$\mathcal{I} \cap \left(B\left(p, \frac{\varepsilon f(p)}{8}\right) \cup \overline{B}\left(p, \frac{6f(p)}{\varepsilon}\right)\right).$$

Proof. The first inequality $f(t) \leq V_{(p,w)}(t)$ follows immediately from Lemma 2.4.

As for the other inequality, for $t \in \mathcal{I} \cap B(p, \frac{\varepsilon f(p)}{8})$ we have

$$\frac{w}{(1+\varepsilon/8)} - \frac{\varepsilon w}{8} \leq f(p) - |pt| \leq f(t) \leq V_{(p,w)}(t) = w + |pt| \leq w + \frac{\varepsilon w}{8}.$$

However,

$$\frac{w + \varepsilon w/8}{\frac{w}{1+\varepsilon/8} - \varepsilon w/8} = \frac{1+\varepsilon/8}{\frac{1}{1+\varepsilon/8} - \varepsilon/8} \leq \frac{1+\varepsilon/8}{1-\varepsilon/8-\varepsilon/8} = \frac{1+\varepsilon/8}{1-\varepsilon/4} \leq 1+\varepsilon$$

since $\varepsilon \leq 1$. Thus, $V_{(p,w)}(t) \leq (1+\varepsilon)f(t)$ for all $t \in \mathcal{I} \cap B(p, \varepsilon f(p)/8)$.

For $t \in \mathcal{I} \cap \overline{B}(p, 6f(p)/\varepsilon)$, we have

$$|pt| - w \leq |pt| - f(p) \leq f(t) \leq V_{(p,w)}(t) = |pt| + w.$$

However,

$$\frac{|pt| + w}{|pt| - w} = 1 + \frac{2w}{|pt| - w} \leq 1 + \frac{2w}{3w/\varepsilon - w} \leq 1 + \frac{2w}{2w/\varepsilon} = 1 + \varepsilon$$

since $|pt| \geq 6f(p)/\varepsilon \geq 3w/\varepsilon$. Thus, $V_{(p,w)}(t) \leq (1+\varepsilon)f(t)$ for any such t. ∎

LEMMA 2.6. *Let $\mathcal{I}$ be a convex subset of $\mathbb{R}^d$, f a distance function defined over $\mathcal{I}$, $0 < \varepsilon \leq 1$ a parameter, and p a point in $\mathcal{I}$. Then for any $t \in \mathcal{I} \cap B(p, \varepsilon f(p)/9)$ and any number w_t such that $f(t) \leq w_t \leq (1+\varepsilon/8)f(t)$, we have $f(p) \leq V_{(t,w_t)}(p) \leq (1+\varepsilon)f(p)$.*

Proof. Since $|pt| \leq \varepsilon f(p)/9$, it follows that $f(t) \geq f(p) - |pt| \geq f(p)\,(1 - \varepsilon/9)$.

Thus,

$$\frac{\varepsilon f(t)}{8} \geq \frac{\varepsilon}{8} f(p) \left(1 - \frac{\varepsilon}{9}\right) \geq \frac{\varepsilon f(p)}{9} \geq |pt|,$$

implying that $p \in B(t, \varepsilon f(t)/8)$. By Lemma 2.5, we have $f(p) \leq V_{(t,w_t)}(p) \leq (1+\varepsilon)f(p)$. □

The preceding lemmas suggest the following strategy for constructing an approximation of a distance function f over (a convex portion of) $\mathbb{R}^d$: Pick a point p such that $f(p)$ is close to the global minimum of f. The Voronoi diagram $\mathcal{V}_{(p,w)}$ approximates f "well" near p and outside a larger ball centered at p, where w is an approximation of $f(p)$. We approximate f in the space between those two balls by partitioning it into concentric spherical shells whose radii form an increasing geometric progression and by covering each shell by a uniform grid (whose unit length increases with the radius of the shell). In this manner, the number of points needed is only a function of ε (the approximation factor) and d.

When approximating a distance function on a convex subset $\mathcal{I}$ of $\mathbb{R}^d$, we have to cope with the possibility that sites might be placed outside $\mathcal{I}$. We overcome this by projecting all such sites onto the boundary of $\mathcal{I}$.

DEFINITION 2.7. *Let $\mathcal{I}$ be a convex subset in $\mathbb{R}^d$, and let x be a point in $\mathbb{R}^d$. Let $\nu(x,\mathcal{I})$ denote the projection of x onto $\mathcal{I}$; that is, $\nu(x,\mathcal{I})$ is the closest point (in the Euclidean distance) in $\mathcal{I}$ to x. Clearly, if $x \in \mathcal{I}$, then $\nu(x,\mathcal{I}) = x$.*

When x is fixed, we call $\nu(x,\mathcal{I})$ the hook point *of $\mathcal{I}$.*

DEFINITION 2.8. *Let $r \geq r' > 0$ be real numbers, let p be a point in $\mathbb{R}^d$, and let $\mathcal{I}$ be a convex set in $\mathbb{R}^d$. We denote by $S(p,\mathcal{I},r,r')$ the set $\mathcal{I}_{r'} \cap B(p,r) \cap ((r'/\sqrt{d})\mathbb{Z}^d)$, where $\mathbb{Z}^d$ is the integer lattice and $\mathcal{I}_{r'} = \cup_{q\in\mathcal{I}} B(q,r')$ is the set of all points in $\mathbb{R}^d$ that are at distance at most r' from some point of $\mathcal{I}$. Clearly, $|S(p,\mathcal{I},r,r')| = O((r/r')^d)$ (with a constant of proportionality depending on d).*

The following technical lemma shows how to pick the sites of the additive weighted Voronoi diagram so that it ε-approximates a given distance function.

LEMMA 2.9. *Let $\mathcal{I}$ be a convex subset of $\mathbb{R}^d$, $f : \mathcal{I} \to \mathbb{R}$ a distance function, $0 < \varepsilon \leq 1$ a parameter, c a positive constant, and p a point in $\mathcal{I}$ such that $f(t) \geq f(p)/c$ for all $t \in \mathcal{I}$. Then one can compute a set S in $\mathcal{I}$ of size $O((1/\varepsilon)^d \log(1/\varepsilon))$ (the constant of proportionality depends on c) such that $p \in S$, and for any weight function w on S satisfying $f(x) \leq w(x) \leq (1+\varepsilon/8)f(x)$ for all $x \in S$, we have $f(t) \leq V_{\mathcal{S}}(t) \leq (1+\varepsilon)f(t)$ for all $t \in \mathcal{I}$, where $\mathcal{S} = (S,w)$.*

Proof. Let w_p be any number satisfying $f(p) \leq w_p \leq (1+\varepsilon/8)f(p)$.

Let $r_i = (2^i+1)w_p$, for $i = 1,\ldots,m$, where $m = \lceil \log_2(6/\varepsilon) \rceil$. Let $\mathcal{A}_1 = B(p,r_1)$, let $\mathcal{A}_i = \mathcal{A}(p,r_{i-1},r_i)$, for $i = 2,\ldots,m$, and let $\mathcal{A}_{m+1} = \overline{B}(p,r_m)$. Clearly, $\mathcal{I} = \cup_{i=1}^{m+1}(\mathcal{I} \cap \mathcal{A}_i)$.

Let $r'_1 = \varepsilon w_p/(18c)$ and let $r'_i = \varepsilon 2^{i-1} w_p/9$ for $i = 2,\ldots,m$. Let $S'_i = \mathcal{A}_i \cap S(p,\mathcal{I},r_i,r'_i)$ for $i = 1,\ldots,m$. Let $S = \{p\} \cup \bigcup_{i=1}^m S_i$, where $S_i = \{\nu(x,\mathcal{I}) | x \in S'_i\}$, for $i = 1,\ldots,m$. See Figure 2.2 for an illustration of the set S.

Let w be any weight function such that $f(x) \leq w(x) \leq (1+\varepsilon/8)f(x)$ for any $x \in S$, and let $\mathcal{S} = (S,w)$.

We claim that $\mathcal{S}$ is the required weighted set. Indeed, let $t \in \mathcal{I}$. If $t \in \mathcal{A}_{m+1}$, then $|pt| \geq (2^m+1)w_p \geq 6f(p)/\varepsilon$. Thus, $t \in \overline{B}(p, 6f(p)/\varepsilon)$, and by Lemmas 2.4 and 2.5, we have $f(t) \leq V_{\mathcal{S}}(t) \leq V_{(p,w_p)}(t) \leq (1+\varepsilon)f(t)$.

If $t \in B(p,r_m)$, let $\mathcal{A}_i$ be the shell containing t. Let x' be the closest point to t in S'_i. Let $x = \nu(x',\mathcal{I})$. By the definition of ν and S'_i and by the convexity of $\mathcal{I}$, we have $|tx| \leq |tx'| \leq r'_i$ (see Figure 2.3).

If $i = 1$, then the inequality $f(t) \geq f(p)/c \geq w_p/2c$ implies that $|tx| \leq r'_1 = \varepsilon w_p/(18c) \leq \varepsilon f(t)/9$. Thus, $x \in B(t, \varepsilon f(t)/9)$.

If $i > 1$, then we also have $|tx| \leq r'_i = \varepsilon 2^{i-1} w_p/9 \leq \varepsilon f(t)/9$, since $f(t) \geq$

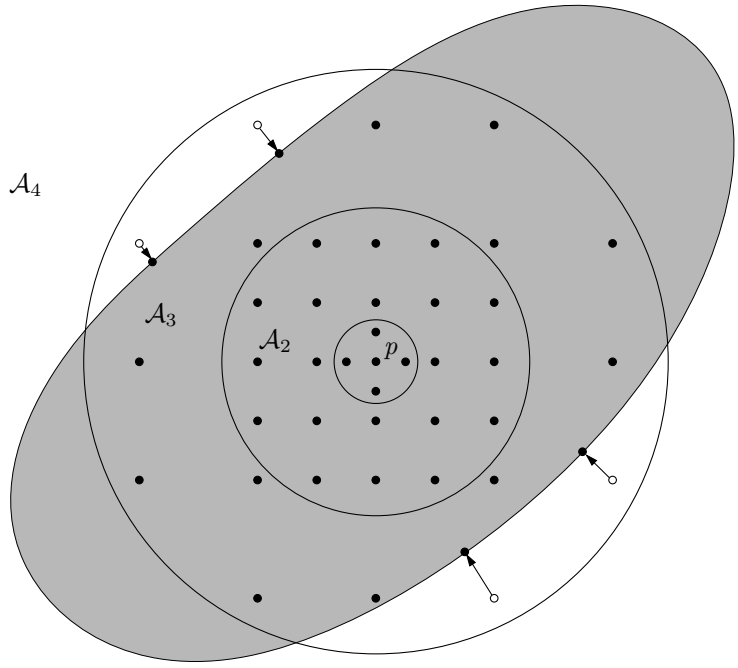


FIG. 2.2. *Illustration of the proof of Lemma* 2.9. *We pick our sites to be on a uniform grid inside each concentric shell around p.*

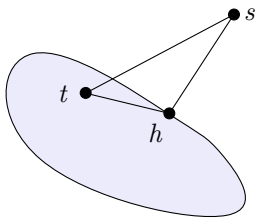


FIG. 2.3. $|st| \geq |ht|$ *for* $h = \nu(s, \mathcal{I})$.

$|pt| - f(p) \geq (2^{i-1}+1)w_p - f(p) \geq 2^{i-1}w_p$. Thus, $x \in B(t, \varepsilon f(t)/9)$.

By Lemmas 2.6 and 2.4, we have $f(t) \leq V_{\mathcal{S}}(t) \leq V_{(x,w(x))}(t) \leq (1+\varepsilon)f(t)$.

As for the size of S, we have

$$|S| = O\left(\sum_{i=1}^{m}\left(\frac{r_i}{r'_i}\right)^d\right) = O\left(\left(\frac{3w_p}{\varepsilon w_p/(18c)}\right)^d + \sum_{i=2}^{m}\left(\frac{(2^i+1)w_p}{\varepsilon 2^{i-1}w_p/9}\right)^d\right)$$
$$= O\left(\frac{1}{\varepsilon^d}\log\frac{1}{\varepsilon}\right). \qquad \square$$

To approximate a distance function $f(\cdot)$ using the constructive proof of Lemma 2.9, we need to find a point which is, within a constant factor, a global minimum of f over the given range. The following lemma shows that this can be easily done if f has a known zero point outside the given range.

LEMMA 2.10. *Let $\mathcal{I}'$ be a subset of $\mathbb{R}^d$, $f : \mathcal{I}' \to \mathbb{R}$ a distance function, $0 < \varepsilon \leq 1$ a parameter, $\mathcal{I}$ a convex subset of $\mathcal{I}'$, and s a point in $\mathcal{I}' \setminus \mathcal{I}$ such that $f(s) = 0$. Then $f(t) \geq f(h)/2$ for all $t \in \mathcal{I}$, where $h = \nu(s, \mathcal{I})$.*

Proof. Let t be any point in $\mathcal{I}$. Since h is the closest point in $\mathcal{I}$ to s and $\mathcal{I}$ is convex, it easily follows that $|st| \geq |ht|$ (see Figure 2.3). Since $f(t) = f(t) + f(s) \geq |st|$, we have $f(t) \geq |ht|$. Moreover, the segment ht is contained in $\mathcal{I}$, implying that $f(h) \leq f(t) + |ht| \leq 2f(t)$. □

Remark 2.2. The lemma also holds when $s \in \mathcal{I}$, but then it only yields the trivial bound $f(t) \geq 0$ for all $t \in \mathcal{I}$. Then, of course, s is the required global minimum.

Remark 2.3. Let $\mathcal{I}'$ be a set in $\mathbb{R}^d$, $\mathcal{I}$ a convex subset of $\mathcal{I}'$, $f : \mathcal{I}' \to \mathbb{R}$ a distance function, $0 < \varepsilon \leq 1$ a parameter, and s a point of $\mathcal{I}'$ such that $f(s) = 0$. Computing a weighted set $\mathcal{S} = (S, w)$ such that the weighted Voronoi diagram induced on $\mathcal{I}$ approximates f up to a factor of $(1 + \varepsilon)$ can be accomplished by following the proof of Lemma 2.9 in four stages:

(i) Compute the point $p = \nu(s, \mathcal{I})$. By Lemma 2.10, p is "almost" a minimum of f on $\mathcal{I}$.
(ii) Compute an $(\varepsilon/8)$-approximation w_p to $f(p)$ (as prescribed in the lemma).
(iii) Construct the set S of points in $\mathcal{I}$, as prescribed in the proof.
(iv) Approximate the distance function values of all the points of S, up to a factor of $1 + \varepsilon/8$, and use these values as the weights for the points of S.

Remark 2.4. The set of points S produced in the proof of Lemma 2.9 is made out of $O(\log(1/\varepsilon))$ subsets (i.e., $S_1, \ldots, S_m$) such that $f(x) \leq cf(t)$, for all $x, t \in S_i$, for $1 \leq i \leq m$, where c is an appropriate constant. This property enables us, in the case of shortest paths on a convex polytope, to approximate the value of the distance function to all the points of S_i simultaneously, yielding a more efficient algorithm. See Remark 3.1 for the details.

Remark 2.5. Let $\mathcal{I}'$ be a subset of $\mathbb{R}^d$, and let f, g be two distance functions defined over $\mathcal{I}'$. It is easy to verify that $h(x) = \max(f(x), g(x))$ is also a distance function. This implies that the distance function induced by any furthest neighbor Voronoi diagram of a finite set of points in $\mathbb{R}^d$ is a distance function in $\mathbb{R}^d$. Hence, by Lemma 2.9 we have the following.

COROLLARY 2.11. *Any furthest neighbor Voronoi diagram of points in $\mathbb{R}^d$ can be ε-approximated by a (nearest neighbor) weighted Voronoi diagram with $O((1/\varepsilon^d) \log(1/\varepsilon))$ sites.*

We can strengthen Lemma 2.9 by noticing that in the cases to which we are going to apply it, our distance function is the length of a shortest path from a fixed source point to the given point of $\mathcal{I}$. In such cases, if the source point lies outside $\mathcal{I}$, then a shortest path connecting any point in $\mathcal{I}$ to our source point must first pass through the boundary $\partial\mathcal{I}$. We next show that if we are able to ε-approximate the distance on the boundary of $\mathcal{I}$, then we can trivially ε-approximate the distance function to all the points of $\mathcal{I}$.

DEFINITION 2.12. *Let $\mathcal{I}$ be a convex polytope in $\mathbb{R}^d$. We call a function $f : \mathcal{I} \to \mathbb{R}$* boundary-induced *on $\mathcal{I}$, if f is a distance function, and for any $t \in \mathcal{I}$, there exists a point $x \in \partial\mathcal{I}$ such that $f(t) = f(x) + |tx|$.*

DEFINITION 2.13. *Given a convex polytope $\mathcal{I}$ in $\mathbb{R}^d$, we denote by $\phi(\mathcal{I})$ the set of all the facets ($(d-1)$-faces) of $\mathcal{I}$.*

LEMMA 2.14. *Let $\mathcal{I}$ be a convex polytope in $\mathbb{R}^d$, $f : \mathcal{I} \to \mathbb{R}$ a boundary-induced distance function, and $0 < \varepsilon \leq 1$ a parameter. For any facet F of $\mathcal{I}$, let $\mathcal{S}(F) = (S_F, w_F)$ be a weighted set of points in F such that $f(t) \leq V_{\mathcal{S}(F)}(t) \leq (1 + \varepsilon)f(t)$,*

for all $t \in F$. Then $f(t) \leq V_{\mathcal{S}}(t) \leq (1+\varepsilon)f(t)$ for all $t \in \mathcal{I}$, where $\mathcal{S} = (S, w) = \cup_{F \in \phi(\mathcal{I})} \mathcal{S}(F)$.

Proof. For any $t \in \mathcal{I}$, let x be the point in $\partial\mathcal{I}$ satisfying $f(t) = f(x) + |tx|$, and let F be a facet of $\mathcal{I}$ that contains x. By Lemma 2.4, we have

$$f(x) + |tx| = f(t) \leq V_{\mathcal{S}}(t) \leq V_{\mathcal{S}}(x) + |tx| \leq V_{\mathcal{S}(F)}(x) + |tx| \leq (1+\varepsilon)f(x) + |tx| \leq (1+\varepsilon)(f(x) + |tx|) = (1+\varepsilon)f(t). \qquad \square$$

Remark 2.6. Let $\mathcal{I}'$ be a set in $\mathbb{R}^d$, $\mathcal{I}$ a convex subset of $\mathcal{I}'$, $f : \mathcal{I}' \to \mathbb{R}$ a boundary-induced distance function, $0 < \varepsilon \leq 1$ a parameter, and s a point of $\mathcal{I}'$ such that $f(s) = 0$. Computing a weighted set $\mathcal{S} = (S, w)$ such that the weighted Voronoi diagram induced on $\mathcal{I}$ approximates f up to a factor of $(1+\varepsilon)$, can be done by applying the algorithm described in Remark 2.3 for each facet of $\mathcal{I}$. By Lemma 2.14, the union of all these weighted sets has the required properties.

3. Approximate shortest path map on a polyhedral surface in $\mathbb{R}^3$. Let $\mathcal{P}$ be a given polyhedral surface in $\mathbb{R}^3$ with n edges, let s be a source point on $\mathcal{P}$, and let $0 < \varepsilon \leq 1$ be a given parameter. In this section, we give an algorithm for constructing an approximation map on $\mathcal{P}$ of complexity $O((n/\varepsilon)\log(1/\varepsilon))$ such that given any $t \in \mathcal{P}$, one can compute in $O(\log(n/\varepsilon))$ time a distance $\Delta_{\mathcal{P}}(s,t)$ satisfying $d_{\mathcal{P}}(s,t) \leq \Delta_{\mathcal{P}}(s,t) \leq (1+\varepsilon)d_{\mathcal{P}}(s,t)$.

Although the following description is rather technical, one has to bear in mind that it is a straightforward implementation of the technique of Section 2. Namely, for each edge of our domain (polyhedral terrain, or a convex polytope) we compute a "small" set of points, the (geodesic) distance from the source point to all those points, and we construct the weighted additive Voronoi diagram that those points induce on each face of the domain.

DEFINITION 3.1. *A* polyhedral surface $\mathcal{P}$ *in* $\mathbb{R}^3$ *is the union of a collection of planar polygonal faces, with their edges and vertices, such that each edge is incident to at most two faces and any pair of faces intersect either at a common edge, a common vertex, or not at all. A face is a simple closed polygon (i.e., it contains its boundary), and an edge is a closed segment (i.e., it contains its endpoints). Without loss of generality we assume that all the faces are triangular (since simple polygons may be triangulated in linear time* [4] *and the number of new edges introduced by the triangulation is linear in the number of vertices). We also assume that $\mathcal{P}$ is connected.*

A polyhedral terrain *is a polyhedral surface that intersects every vertical line in at most a single point.*

DEFINITION 3.2. *Given a polyhedral surface $\mathcal{P}$ in $\mathbb{R}^3$ and any two points s, t on $\mathcal{P}$, we denote by $d_{\mathcal{P},s}(t)$ the length of a shortest path between s and t on $\mathcal{P}$.*

As noted in Example 2.2 (i), $d_{\mathcal{P},s}(\cdot)$ is a distance function on $\mathcal{P}$. Moreover, if F is a face of $\mathcal{P}$ and $s \notin F$, then $d_{\mathcal{P},s}$ is boundary induced on F. (If $s \in F$, then $d_{\mathcal{P},s}(t)$ is the Euclidean distance $|st|$.)

The following theorem is the main result of this section.

THEOREM 3.3. *Let $\mathcal{P}$ be a polyhedral surface in $\mathbb{R}^3$ with n edges, s a source point on $\mathcal{P}$, and $0 < \varepsilon \leq 1$ a real parameter. Then there exists a subdivision Π of $\mathcal{P}$ of complexity $O((n/\varepsilon)\log(1/\varepsilon))$, which facilitates ε-approximate shortest path queries from s on $\mathcal{P}$. That is, for any query point t on $\mathcal{P}$, one can compute, in $O(\log(n/\varepsilon))$ time, a distance $\Delta_{\mathcal{P}}(s,t)$ such that $d_{\mathcal{P},s}(t) \leq \Delta_{\mathcal{P}}(s,t) \leq (1+\varepsilon)\, d_{\mathcal{P},s}(t)$.*

The map can be computed in $O(n^2 \log n + (n/\varepsilon)\log(1/\varepsilon)\log(n/\varepsilon))$ time if $\mathcal{P}$ is an arbitrary polyhedral surface, and in $O((n/\varepsilon^3)\log(1/\varepsilon) + (n/\varepsilon^{1.5})\log(1/\varepsilon)\log n)$ time if $\mathcal{P}$ is a convex polytope.

The space used by the algorithm is $O((n/\varepsilon)\log(1/\varepsilon))$ if $\mathcal{P}$ is either a convex polytope or a polyhedral terrain, and $O(n^2 + (n/\varepsilon)\log(1/\varepsilon))$ otherwise.

Proof. For each face F of $\mathcal{P}$ that does not contain s, we construct a weighted Voronoi diagram that approximates $d_{\mathcal{P},s}$ on F. By Remark 2.6, this can be done by constructing a weighted Voronoi diagram on each edge of $\mathcal{P}$, as outlined in Remark 2.3.

For the general case, we compute the exact shortest path map of s on $\mathcal{P}$, using the algorithm of [16], in $O(n^2 \log n)$ time. The exact map enables us to compute the shortest distance from s to any point of $\mathcal{P}$ in $O(\log n)$ time. Thus, computing the distances from s to the n hooks of the edges of $\mathcal{P}$ takes additional $O(n \log n)$ time. The hook point of an edge is the closest point of the edge to s, and it can be computed in $O(1)$ time.

For the convex case, we approximate the distances on $\mathcal{P}$ to all the hooks on the edges of $\mathcal{P}$, up to a factor of $(1+\varepsilon/8)$. This takes $O(n/\varepsilon^3 + (n/\varepsilon^{1.5})\log n)$ time, using the algorithm of [1, section 6].

For each edge e of $\mathcal{P}$, we compute a set S_e of $O((1/\varepsilon)\log(1/\varepsilon))$ points on e, as specified in the proof of Lemma 2.9, taking the corresponding p to be the hook of e and w_p to be the approximated (exact in the nonconvex case) distance along $\mathcal{P}$ from s to p. Let $S = \cup_e S_e$, taken over all edges e of $\mathcal{P}$.

We now compute (or approximate) the distances from s to all the points in S. For the nonconvex case, this can be done in $O((n/\varepsilon)\log(1/\varepsilon)\log n)$ time, using the exact shortest path map.

For the convex case, we compute approximate distances from s to all points of S, up to a factor of $(1+\varepsilon/8)$. Using the observation of Remark 2.4, we partition S into $O(n\log(1/\varepsilon))$ sets, each of size $O(1/\varepsilon)$, such that the required distances to the points in each such set are within a fixed constant factor of each other (namely, for each edge e of $\mathcal{P}$, the set S_e is decomposed into $O(\log(1/\varepsilon))$ sets, as in the proof of Lemma 2.9). Using the algorithm described in Remark 3.1 below, we can compute the distances from s to all the points of S in $O((n/\varepsilon^3)\log(1/\varepsilon) + (n/\varepsilon^{1.5})\log(1/\varepsilon)\log n)$ time.

Next, we compute, for each face F of $\mathcal{P}$, the weighted Voronoi diagram induced by the weighted points of $\mathcal{S}$ that lie on ∂F. This takes $O((n/\varepsilon)\log^2(1/\varepsilon))$ overall time (see [10]), since each face contains $O((1/\varepsilon)\log(1/\varepsilon))$ points of $\mathcal{S}$. Let Π be the resulting map, consisting of the union of all those facial Voronoi diagrams.

By Lemmas 2.9 and 2.14, the map Π on $\mathcal{P}$ has the required properties. Moreover, we can preprocess each face F of $\mathcal{P}$ in $O((1/\varepsilon)\log^2(1/\varepsilon))$ time such that point location queries on F can be answered in $O(\log(1/\varepsilon))$ time (see [18]). Overall, this preprocessing takes $O((n/\varepsilon)\log^2(1/\varepsilon))$ time.

To answer an approximate shortest path query for a query point q, the algorithm must locate the face of $\mathcal{P}$ containing q. If $\mathcal{P}$ is a polyhedral terrain, we project the terrain into the xy-plane and preprocess it, in $O(n\log n)$ time, for planar point location. If $\mathcal{P}$ is a convex polytope, it can be preprocessed in linear time to answer point location queries in $O(\log n)$ time (see [9]). Otherwise, we preprocess $\mathcal{P}$ for spatial point-location in $O(n^2 \log n)$ time and $O(n^2)$ space with $O(\log n)$ query time, using the algorithm of [23].

Given any query point q on $\mathcal{P}$, the algorithm computes the face F of $\mathcal{P}$ that contains q in $O(\log n)$ time. Locating the face of the subdivision Π that contains q takes an additional $O(\log(1/\varepsilon))$ time. Thus, ε-approximate shortest path queries for $\mathcal{P}$ can be answered in $O(\log(n/\varepsilon))$ time. (If the face containing q is already known, the query time reduces to $O(\log(1/\varepsilon))$.) ∎

DEFINITION 3.4. *Let P be a convex body in $\mathbb{R}^3$. An* outer path *of P is a curve γ connecting two points on ∂P and disjoint from the interior of P.*

Remark 3.1. Let P be a convex polytope in $\mathbb{R}^3$, s a source point on P, T a set of points on P, and $0 < \varepsilon \leq 1$ a prescribed parameter. One can ε-approximate the length of the shortest path from s to all the points of T on P, in $O((n+|T|)/\varepsilon^3 + ((n+|T|)/\varepsilon^{1.5}) \log (n+|T|))$ time, by adding the points of T as vertices to P and by using the algorithm of [1, section 6].

The algorithm of [1] works by computing an approximation polytope for each point of T and by computing the exact distance from s to the point on this polytope.

Moreover, suppose that T can be partitioned into m sets $T_1, \ldots, T_m$ such that $d_{P,s}(t) \leq c \cdot d_{P,s}(t')$ for all $t, t' \in T_i$ and for each $i = 1, \ldots, m$, where c is a prescribed constant and all the points of T_i belong to the same edge of P, for any fixed $i = 1, \ldots, m$. Then it is possible to speed up the above algorithm, as follows. Instead of constructing an approximation polytope for each destination point separately, we construct an approximation polytope that can be used to approximate the distances from s to all the points of T_i, for $i = 1, \ldots, m$. This is done by ensuring that all the points of T_i lie on the boundary of the approximation polytope calculated by the algorithm, which can be enforced by intersecting it with a supporting plane of P passing through the edge containing the points of T_i (adding at most one new face to the approximation polytope). We also need to use a more refined approximation polytope, so as to achieve the claimed error bound, but since c is a constant this does not change the asymptotic complexity of the algorithm. See [1] for the technical details.

This improves the running time to

$$O\left(n + \frac{m}{\varepsilon^3} + \frac{m}{\varepsilon^{1.5}} \log n + \frac{|T|}{\varepsilon^{1.5}}\right)$$

by constructing an approximation polytope for the points of T_i (in $O(\frac{m}{\varepsilon^{1.5}} \log n)$ time), computing the exact distance map from the source point on the approximation polytope (in $O(1/\varepsilon^3)$ time), and extracting the shortest path to each point of T_i, repeating all this, for $T_1, \ldots, T_m$. Moreover, for each point $t \in T_i$, the algorithm computes a polygonal outer path of P, made up of $O(1/\varepsilon^{1.5})$ segments, that realizes the approximated distance.

Remark 3.2. The algorithm of [16] works for arbitrary polyhedral surfaces; in particular, it is not restricted to polyhedral terrains. Thus, the algorithm of Theorem 3.3 also works for general polyhedral surfaces.

Remark 3.3. For a convex polytope P with n edges in $\mathbb{R}^3$, one can compute an approximation map that can be used to compute an outer path that realizes the approximate distance. This is done by modifying the algorithm of Theorem 3.3 such that it stores an outer path from the source point to each of the constructed sites, where the outer path realizes its ε-approximate distance. Such a path is readily available from the procedure used to compute the approximate distance to the site, and the complexity of such a path is $O(1/\varepsilon^{1.5})$ (see [1]). The space needed to store the extended approximation map is $O(n/\varepsilon^{2.5} \log (1/\varepsilon))$, and the computation time remains $O((n/\varepsilon^3) \log(1/\varepsilon) + (n/\varepsilon^{1.5}) \log (1/\varepsilon) \log n)$.

The new map can be used to answer approximate shortest path queries, in $O(\log (n/\varepsilon))$ time, and also compute, in additional $O(1/\varepsilon^{1.5})$ time, an outer path of the convex polytope realizing this distance. Such an outer path can be projected onto the boundary of the convex polytope in additional $O\left(n \log (1/\varepsilon) + 1/\varepsilon^3\right)$ time,

resulting in a path on ∂P which is an ε-approximation to the shortest path (see [1]). Note, however, that the performance of the enhanced data structure is poorer in terms of both storage and query time.

4. Constructing spatial approximate shortest path maps in $\mathbb{R}^3$. Let $\mathcal{O}$ be a collection of pairwise-disjoint polyhedral obstacles in $\mathbb{R}^3$, s a source point in $\mathbb{R}^3 \setminus int \cup \mathcal{O}$, and $0 < \varepsilon \leq 1$ a parameter. In this section, we present an algorithm for preprocessing $\mathcal{O}$ such that for any point in $\mathbb{R}^3$ (or, more precisely, for any "free" point that can be reached from s without penetrating an obstacle) one can compute in $O(\log(n/\varepsilon))$ time a distance $\Delta_{\mathcal{O},s}(t)$ satisfying $d_{\mathcal{O},s}(t) \leq \Delta_{\mathcal{O},s}(t) \leq (1+\varepsilon)d_{\mathcal{O},s}(t)$, where $d_{\mathcal{O},s}(t)$ is the length of a shortest path between s and t that avoids the interiors of the obstacles.

The preprocessing time of the algorithm is roughly $O(n^4/\varepsilon^6)$, which shows that the problem of approximating the distance from a single source to all the "free" points in $\mathbb{R}^3$ is not much harder (computationally) than approximating the distance between any specific pair of points (which can be done in roughly $O(n^2/\varepsilon^4)$ time (see [7])). In fact, for a fixed source point and many destination points, our algorithm will actually be faster. The problem of computing the *exact* distance between two points in $\mathbb{R}^3$ among polyhedral obstacles is NP-hard, as shown by Canny and Reif [3], and the fastest available algorithms for this problem run in time that is exponential in the total number of obstacle vertices [20, 21, 22].

DEFINITION 4.1. *Let $\mathcal{O}$ be a collection of pairwise-disjoint polyhedral obstacles with a total of n edges in $\mathbb{R}^3$ and s a source point in $FP(\mathcal{O}) = \mathbb{R}^3 \setminus int \cup_{O \in \mathcal{O}} O$. Let $FP(\mathcal{O}, s)$ denote the set of all points in $FP(\mathcal{O})$ that can be connected to s by a path that avoids the interiors of the obstacles of $\mathcal{O}$.*

For any $t \in FP(\mathcal{O}, s)$, as above, we denote by $d_{\mathcal{O},s}(t)$ the length of a shortest path between s and t that avoids the interiors of the obstacles of $\mathcal{O}$.

As noted in Example 2.2 (ii), $d_{\mathcal{O},s}(\cdot)$ is a distance function over $FP(\mathcal{O}, s)$, and for any convex set $\mathcal{I} \subseteq FP(\mathcal{O}, s)$ such that $s \notin \mathcal{I}$, the function $d_{\mathcal{O},s}(\cdot)$ is boundary induced over $\mathcal{I}$.

THEOREM 4.2 (Clarkson [7]). *Given a set $\mathcal{O}$ of polyhedral obstacles in $\mathbb{R}^3$, and points s and t, an ε-approximate path between s and t that does not penetrate into any obstacle in $\mathcal{O}$ can be computed in*

$$O\left(\frac{n^2}{\varepsilon^4}\beta(n)\log\frac{n}{\varepsilon} + n^2\log(n\rho)\log(n\log\rho)\right)$$

time, where n is the number of obstacle edges, ρ is the ratio of the length of the longest edge in $\mathcal{O}$ to the Euclidean distance between s and t, $\beta(n) = \alpha(n)^{O(\alpha(n))^{O(1)}}$, and $\alpha(n)$ is the inverse of the Ackermann function.

The following theorem is the main result of this section.

THEOREM 4.3. *Let $\mathcal{O}$ be a collection of pairwise-disjoint polyhedral obstacles with n edges in $\mathbb{R}^3$, s a source point in $FP(\mathcal{O})$, and $0 < \varepsilon < 1$ a parameter. Then a subdivision $\mathcal{M}$ of $FP(\mathcal{O}, s)$ of complexity $O(n^2/\varepsilon^{4+\delta})$, for any $1 > \delta > 0$, can be computed in*

$$O\left(\frac{n^4}{\varepsilon^2}\left(\frac{\beta(n)}{\varepsilon^4}\log\frac{n}{\varepsilon} + \log(n\rho)\log(n\log\rho)\right)\log\frac{1}{\varepsilon}\right)$$

time, where ρ, and $\beta(n)$ are as above.

For any query point $t \in FP(\mathcal{O}, s)$, one can compute in $O(\log(n/\varepsilon))$ time a distance $\Delta_{\mathcal{O},s}(t)$ such that $d_{\mathcal{O},s}(t) \leq \Delta_{\mathcal{O},s}(t) \leq (1+\varepsilon)\ d_{\mathcal{O},s}\ (t)$.

Proof. First, we partition $FP(\mathcal{O})$ into $O(n^2)$ vertical prisms. This can be easily done by erecting a vertical wall from each edge of the obstacles. For an edge e of the obstacles, such a wall is the set of all points in $FP(\mathcal{O})$ that lie on vertical rays emanating from the edge and not intersecting the obstacles. Let $\mathcal{M}'''$ denote the resulting partition of $FP(\mathcal{O})$. It is easy to verify that the complexity of $\mathcal{M}'''$ is $O(n^2)$ and that it can be computed in $O(n^2 \log n)$ time.

We refine $\mathcal{M}'''$ by further partitioning each cell of $\mathcal{M}'''$ into vertical triangular prisms. This is done by projecting each cell of $\mathcal{M}'''$ into the xy-plane and by triangulating the resulting polygon in $O(m \log m)$ time (see [18]), where m is the number of vertices of the polygon. For each new edge created, we erect a corresponding vertical wall inside the cell. Let $\mathcal{M}''$ be the resulting subdivision of $FP(\mathcal{O})$. Clearly, the complexity of $\mathcal{M}''$ remains $O(n^2)$, and it can be computed in additional $O(n^2 \log n)$ time.

Let T_v be the vertical prism in $\mathcal{M}''$ that contains s. We construct an adjacency graph G on the vertical prisms of $\mathcal{M}''$. By computing the connected component of G that contains T_v, one obtains the subdivision $\mathcal{M}' = \mathcal{M}'' \cap FP(\mathcal{O}, s)$.

Each cell $\mathcal{I}$ of $\mathcal{M}'$ is a vertical prism, having at most five faces. We can approximate the distance function $d_{\mathcal{O},s}(t)$ inside $\mathcal{I}$ by computing a weighted set $\mathcal{S}_{\mathcal{I}} = (S_{\mathcal{I}}, w_{\mathcal{I}})$, as specified in the proofs of Lemmas 2.9 and 2.14. To do so, it is necessary to $(\varepsilon/8)$-approximate the value of $d_{\mathcal{O},s}(\cdot)$ for $O((1/\varepsilon^2)\log(1/\varepsilon))$ points (i.e., the points of $\mathcal{S}_{\mathcal{I}}$). By Theorem 4.2, this takes

$$O\left(\left(\frac{\beta(n)}{\varepsilon^4}\log\frac{n}{\varepsilon} + \log(n\rho)\log(n\log\rho)\right)\frac{n^2}{\varepsilon^2}\log\frac{1}{\varepsilon}\right)$$

time. The weighted Voronoi diagram $\mathcal{V}_{\mathcal{S}_{\mathcal{I}}}$ induced by $\mathcal{S}_{\mathcal{I}}$ inside $\mathcal{I}$ approximates $d_{\mathcal{O},s}$ inside $\mathcal{I}$ up to a factor of $1+\varepsilon$.

Let $\mathcal{S} = \cup_{\mathcal{I}\in\mathcal{M}'}\mathcal{S}_{\mathcal{I}}$. Clearly, one can $(\varepsilon/8)$-approximate the distance between s and all the sites of $\mathcal{S}$ in

$$O\left(\frac{n^4}{\varepsilon^2}\left(\frac{\beta(n)}{\varepsilon^4}\log\frac{n}{\varepsilon} + \log(n\rho)\log(n\log\rho)\right)\log\frac{1}{\varepsilon}\right)$$

time.

Let $\mathcal{M}$ be the subdivision $\cup_{\mathcal{I}\in\mathcal{M}'}\ (\mathcal{V}_{\mathcal{S}_{\mathcal{I}}} \cap \mathcal{I})$. We preprocess $\mathcal{M}$ for spatial point location by constructing a two-level spatial point location data structure. First, we preprocess $\mathcal{M}'$ for point location in $O(n^2 \log n)$ time, using the algorithm of [23]. Next, we preprocess each cell $\mathcal{I}$ of $\mathcal{M}'$ for nearest neighbor queries for the weighted set $\mathcal{S}_{\mathcal{I}}$. By Lemma 4.5, performing this preprocessing for all the cells of $\mathcal{M}'$ takes a total of $O\left(n^2/\varepsilon^{4+\delta}\right)$ randomized expected time and space, for any $\delta > 0$.

For any query point $t \in FP(\mathcal{O}, s)$, we can compute in $O(\log n + \log(1/\varepsilon)) = O(\log(n/\varepsilon))$ time the cell of $\mathcal{M}$ that contains t; that is, in $O(\log(n/\varepsilon))$ time, one can compute a distance $\Delta_{\mathcal{O},s}(t)$ such that $d_{\mathcal{O},s}(t) \leq \Delta_{\mathcal{O},s}(t) \leq (1+\varepsilon)d_{\mathcal{O},s}(t)$. ∎

To complete the description and analysis of the algorithm, we next show how to preprocess a weighted set in $\mathbb{R}^3$ so that one can perform efficient nearest neighbor queries in the additive weighted Voronoi diagram that it induces.

DEFINITION 4.4. *Let $\mathcal{S} = (S, w)$ be a weighted set in $\mathbb{R}^3$. We decompose the weighted Voronoi diagram $\mathcal{V}_{\mathcal{S}}$ into "simpler" cells in the following way: For each cell C in $\mathcal{V}_S$, we compute the spherical map S_C of the cell by projecting the boundary of the*

cell onto the sphere of directions centered at p_C, where p_C is the site of C in S. (We use here the well-known property that C is star-shaped with respect to p_C.) We decompose S_C into pseudovertical subcells on the sphere of directions by drawing a meridian arc upward and downward from each vertex S_C and from each locally longitude-extremal point on any arc of S_C, and by extending each of these meridian arcs until it hits another arc of S_C or, failing this, all the way to the poles of the sphere of directions. Clearly, the complexity of S_C is linear in the complexity of cell C.

We project each "vertical" trapezoid in S_C back into C to obtain the portion within C of the cone with apex p_C spanned by the trapezoid. This defines a decomposition of C into simple subcells such that each subcell is uniquely defined by at most six points of S. We decompose all the cells of $\mathcal{V}_S$ in a similar manner and let $\mathcal{C}(\mathcal{S})$ denote the resulting subdivision. We call $\mathcal{C}(\mathcal{S})$ the spherical decomposition *of $\mathcal{V}_\mathcal{S}$.*

For a weighted set $\mathcal{R} \subseteq \mathcal{S}$ and a subcell $\mathcal{T} \in \mathcal{C}(\mathcal{R})$, a weighted point $(p, w_p) \in \mathcal{S}$ conflicts with *$\mathcal{T}$ if there exists a point $t \in \mathcal{T}$ such that $V_{(p,w_p)}(t) < V_\mathcal{R}(t)$. Let $K(\mathcal{S}, \mathcal{T})$ denote the set of all the points of $\mathcal{S}$ that conflict with $\mathcal{T}$. The* conflict size *of $\mathcal{T}$ is $w(\mathcal{S}, \mathcal{T}) = |K(\mathcal{S}, \mathcal{T})|$.*

LEMMA 4.5. *Let $\mathcal{S} = (S, w)$ be a weighted set of m points in $\mathbb{R}^3$ and $\delta > 0$ a parameter. One can compute, in $O(m^{2+\delta})$ randomized expected time, a data structure for nearest-neighbor queries, of size $O(m^{2+\delta})$, such that for any point $p \in \mathbb{R}^3$ one can compute, in $O(\log m)$ time, the cell of $\mathcal{V}_\mathcal{S}$ that contains p, that is, the point in $\mathcal{S}$ realizing the distance $V_\mathcal{S}(p)$.*

Proof. We construct the data structure using a randomized divide and conquer algorithm. We randomly pick a subset R of S of size r, where r is a parameter to be specified later. One can compute the weighted Voronoi diagram of $\mathcal{R} = (R, w)$, in $O(r^2)$ time, by Remark 2.1 and construct the spherical decomposition $\mathcal{C}(\mathcal{R})$ in $O(r^2 \log r)$ additional time, using plane sweeping techniques on the sphere of directions (see [18]).

For each subcell $\mathcal{T}$ in $\mathcal{C}(\mathcal{R})$, we compute its conflict size $w(\mathcal{S}, \mathcal{T})$. Each subcell in $\mathcal{C}(\mathcal{R})$ is uniquely defined by at most six sites in $\mathcal{R}$, and if $K(\mathcal{S}, \mathcal{T}) \cap \mathcal{R} \neq \emptyset$, then $\mathcal{T} \notin C(\mathcal{R})$. We can thus apply the analysis of Clarkson and Shor. By [8, Corollary 3.8], $w(\mathcal{S}, \mathcal{T}) \leq c \cdot (n \log r)/r$ for all $\mathcal{T} \in \mathcal{C}(\mathcal{R})$, with probability at least $1/2$, where $c > 0$ is an appropriate constant. We sample $\mathcal{R}$ from $\mathcal{S}$ repeatedly until we get a sample that fulfills this condition. Overall, this stage takes $O(mr^2 + r^2 \log r)$ expected running time. For each cell $\mathcal{T} \in \mathcal{C}(\mathcal{R})$, we construct recursively a data structure for point-location in the Voronoi diagram $\mathcal{V}_{K(\mathcal{S},\mathcal{T})}$.

For any query point p, locating the subcell $\mathcal{T}$ in $\mathcal{C}(\mathcal{R})$ that contains p is done by a brute force search inside $\mathcal{C}(\mathcal{R})$ in $O(r^2)$ time. Then we compute the point realizing $V_\mathcal{S}(p)$ by recursively performing a nearest neighbor query in the data structure computed for $\mathcal{V}_{K(\mathcal{S},\mathcal{T})}$. Thus, a query takes $Q(m) = Q(c(m \log r)/r) + O(r^2)$ time, and the data structure can be computed in randomized expected time as

$$T(m) = T(r) + O(r^2)T\left(\frac{cm \log r}{r}\right) + O\left(mr^2 + r^2 \log r\right).$$

Choosing r to be a sufficiently large constant, we have $Q(m) = O(\log m)$, and $T(m) = O(m^{2+\delta})$ (where the constants of proportionality depend on δ). A similar bound holds for the space required by the algorithm. ∎

Remark 4.1. The only stage in the algorithm of Theorem 4.3 that uses randomization is the construction of the spatial point-location data described in Lemma 4.5. This can be replaced by a deterministic data structure as follows.

We observe that each spherical cell, in the decomposition described, can be parametrized by 24 parameters (6 sites and their respective weights). Thus, we define a range space $(\mathcal{S}, \mathfrak{R})$, where $\mathfrak{R}$ is the set of all possible subsets of $\mathcal{S}$ that are contained inside such a spherical cell. It is easy to verify that this is a range space having finite VC-dimension. By a result of Matoušek [15], we can compute in $O(mr^{O(1)})$ time a subset $\mathcal{R}$ of $\mathcal{S}$ having $O(r \log r)$ points, which is $(1/r)$-net of $(\mathcal{S}, \mathfrak{R})$. In particular, the set $\mathcal{R}$ can replace the random sample in the proof of Lemma 4.5; see [15]. This yields a deterministic algorithm with the same time/space complexity as in Lemma 4.5.

Alternatively, one can naively preprocess the Voronoi diagram $\mathcal{V}_{\mathcal{S}}$ for spatial point-location directly; see [23]. However, this approach is considerably less efficient than the approach proposed above.

5. Conclusions. In this paper we have presented two results for computing approximate maps that facilitate shortest path queries on the surface of a convex polytope or on a polyhedral surface in 3-space, or among polyhedral obstacles in 3-space. We conclude by mentioning the following open problems:

(i) Can an ε-approximate shortest path between two points on a polyhedral terrain, or on the surface of a nonconvex polyhedron, be computed in time that is near-linear in the number of edges? A recent subquadratic solution has been obtained by Varadarajan and Agarwal [24], but it computes only a constant-factor approximation to the shortest path.

(ii) Can the exact shortest path between two points on a convex polyhedron be computed in near-linear time? in subquadratic time?

(iii) Can the methods and techniques used in this paper be extended to handle shortest path queries for weighted surfaces (as in [13, 14])?

Acknowledgments. The author wishes to thank Pankaj Agarwal and Micha Sharir for helpful discussions concerning the problems studied in this paper and related problems. Micha Sharir also suggested Lemma 4.5. The author also wishes to thank the referees for their comments and suggestions.

REFERENCES

[1] P. Agarwal, S. Har-Peled, M. Sharir, and K. R. Varadarajan, *Approximate shortest paths on a convex polytope in three dimensions*, J. Assoc. Comput. Mach., 44 (1997), pp. 567–584.

[2] F. Aurenhammer, *Voronoi diagrams: A survey of a fundamental geometric data structure*, ACM Comput. Surv., 23 (1991), pp. 345–405.

[3] J. Canny and J. H. Reif, *New lower bound techniques for robot motion planning problems*, in Proc. 28th IEEE Sympos. Found. Comput. Sci., Los Angeles, 1987, pp. 49–60.

[4] B. Chazelle, *Triangulating a simple polygon in linear time*, Discrete Comput. Geom., 6 (1991), pp. 485–524.

[5] J. Chen and Y. Han, *Shortest paths on a polyhedron; Part* I: *Computing shortest paths*, Internat. J. Comput. Geom. Appl., 6 (1996), pp. 127–144.

[6] J. Choi, J. Sellen, and C. K. Yap, *Approximate Euclidean shortest path in* 3*-space*, J. Comput. Geom. Appl., 7 (1997), pp. 271–295.

[7] K. L. Clarkson, *Approximation algorithms for shortest path motion planning*, in Proc. 19th ACM Sympos. Theory Comput., New York, 1987, pp. 56–65.

[8] K. L. Clarkson and P. W. Shor, *Applications of random sampling in computational geometry,* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[9] D. P. Dobkin and D. G. Kirkpatrick, *A linear algorithm for determining the separation of convex polyhedra*, J. Algorithms, 6 (1985), pp. 381–392.

[10] S. J. Fortune, *A sweepline algorithm for Voronoi diagrams*, Algorithmica, 2 (1987), pp. 153–174.

[11] S. Har-Peled, *Approximate shortest paths and geodesic diameters on convex polytopes in three dimensions*, in Proc. 13th ACM Sympos. Comput. Geom., Nice, France, 1997, pp. 359–365.

[12] J. Hershberger and S. Suri, *Practical methods for approximating shortest paths on a convex polytope in* $\mathbb{R}^3$, in Proc. 6th ACM-SIAM Sympos. Discrete Algorithms, San Francisco, 1995, pp. 447–456.

[13] M. Lanthier, A. Maheshwari, and J.-R. Sack, *Approximating weighted shortest paths on polyhedral surfaces*, in Proc. 13th ACM Sympos. Comput. Geom., Nice, France, 1997, pp. 274–283.

[14] C. Mata and J. Mitchell, *A new algorithm for computing the shortest paths in planar subdivisions*, in Proc. 13th ACM Sympos. Comput. Geom., Nice, France, 1997, pp. 264–273.

[15] J. Matoušek, *Approximations and optimal geometric divide-and-conquer*, in Proc. 23rd ACM Sympos. Theory Comput., New Orleans, 1991, pp. 505–511. Also to appear in J. Comput. Syst. Sci.

[16] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou, *The discrete geodesic problem*, SIAM J. Comput., 16 (1987), pp. 647–668.

[17] D. M. Mount, *Storing the subdivision of a polyhedral surface*, Discrete Comput. Geom., 2 (1987), pp. 153–174.

[18] J. O'Rourke, *Computational Geometry in C*, Cambridge University Press, Cambridge, U.K. 1994.

[19] C. H. Papadimitriou, *An algorithm for shortest path motion in three dimensions*, Inform. Process. Lett., 20 (1985), pp. 259–263.

[20] J. H. Reif and J. A. Storer, *A single-exponential upper bound for finding shortest paths in three dimensions*, J. ACM, 41 (1994), pp. 1013–1019.

[21] M. Sharir, *On shortest paths amidst convex polyhedra*, SIAM J. Comput., 16 (1987), pp. 561–572.

[22] M. Sharir and A. Schorr, *On shortest paths in polyhedral spaces*, SIAM J. Comput., 15 (1986), pp. 193–215.

[23] X.-H. Tan, T. Hirata, and Y. Inagaki, *Spatial Point Location and Its Applications*, Lecture Notes in Comput. Sci. 450, Springer-Verlag, New York, 1990, pp. 241–250.

[24] K. Varadarajan and P. Agarwal, *Approximating shortest paths on a nonconvex polyhedron*, in Proc. 38th IEEE Sympos. Found. Comput. Sci., Miami Beach, 1997, pp. 182–191.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1198–1214

NEW LOWER BOUNDS FOR CONVEX HULL PROBLEMS IN ODD DIMENSIONS*

JEFF ERICKSON†

Abstract. We show that in the worst case, $\Omega(n^{\lceil d/2\rceil-1}+n\log n)$ sidedness queries are required to determine whether the convex hull of n points in $\mathbb{R}^d$ is simplicial or to determine the number of convex hull facets. This lower bound matches known upper bounds in any odd dimension. Our result follows from a straightforward adversary argument. A key step in the proof is the construction of a quasi-simplicial n-vertex polytope with $\Omega(n^{\lceil d/2\rceil-1})$ degenerate facets. While it has been known for several years that d-dimensional convex hulls can have $\Omega(n^{\lfloor d/2\rfloor})$ facets, the previously best lower bound for these problems is only $\Omega(n\log n)$. Using similar techniques, we also obtain simple and correct proofs of Erickson and Seidel's lower bounds for detecting affine degeneracies in arbitrary dimensions and circular degeneracies in the plane. As a related result, we show that detecting simplicial convex hulls in $\mathbb{R}^d$ is $\lceil d/2\rceil$SUM-hard in the sense of Gajentaan and Overmars.

Key words. computational geometry, convex polytopes, degeneracy, lower bounds, decision trees, adversary arguments

AMS subject classifications. 68Q25, 68U05, 52B55, 52B05

PII. S0097539797315410

1. Introduction.

The construction of convex hulls is one of the most basic and well-studied problems in computational geometry [2, 3, 5, 10, 11, 12, 13, 15, 17, 18, 29, 34, 35, 38, 39, 47, 41, 45, 43, 44, 48]. Over 20 years ago, Graham described an algorithm that constructs the convex hull of n points in the plane in $O(n\log n)$ time [29]. The same running time was first achieved in three dimensions by Preparata and Hong [38]. Yao [48] proved a lower bound of $\Omega(n\log n)$ on the complexity of identifying the convex hull vertices, in the quadratic decision tree model. This lower bound was later generalized to the algebraic decision tree and algebraic computation tree models by Ben-Or [7]. It follows that both Graham's scan and Preparata and Hong's algorithm are optimal in the worst case. If the output size f is also taken into account, the lower bound drops to $\Omega(n\log f)$ [34], and a number of algorithms match this bound both in the plane [34, 12, 10] and in three dimensions [18, 16, 10].

In higher dimensions, the problem is not quite so completely solved. Seidel's "beneath-beyond" algorithm [41] constructs d-dimensional convex hulls in $O(n^{\lceil d/2\rceil})$ time. After a 10-year wait, Chazelle [15] improved the running time to $O(n^{\lfloor d/2\rfloor})$ by derandomizing a randomized incremental algorithm of Clarkson and Shor [18]; see also [44]. Since an n-vertex polytope in $\mathbb{R}^d$ can have $\Omega(n^{\lfloor d/2\rfloor})$ facets [27], Seidel's algorithm is optimal in even dimensions, and Chazelle's algorithm is optimal in all dimensions, in the worst case.

However, several faster algorithms are known when the output size f is also considered, at least when the input points are in general position. In 1970, Chand and Kapur [13] described a "gift-wrapping" algorithm that constructs convex hulls in ar-

*Received by the editors January 27, 1997; accepted for publication (in revised form) September 5, 1997; published electronically March 22, 1999. This research was done while the author was a graduate student at the University of California at Berkeley, with the support of a Graduate Assistance in Areas of National Need Fellowship. An extended abstract of this paper was presented at the 12th Annual ACM Symposium on Computational Geometry [22].

http://www.siam.org/journals/sicomp/28-4/31541.html

†Department of Computer Science, University of Illinois, 1504 W. Springfield Ave., Urbana, Illinois, 61801 (jeffe@cs.uiuc.edu).

bitrary dimensions in time $O(nf)$; see also [47]. Seidel's "shelling" algorithm runs in time $O(n^2 + f \log n)$ [43]. A divide-and-conquer algorithm of Chan, Snoeyink, and Yap [12] constructs four-dimensional hulls in time $O((n + f) \log^2 f)$, and a recent improvement by Amato and Ramos [2] constructs five-dimensional hulls in time $O((n + f) \log^3 f)$. In dimensions higher than five, the fastest algorithms are an improvement of the gift-wrapping algorithm by Chan [11] with running time $O(n \log f + (nf)^{1-1/(\lfloor d/2 \rfloor + 1)} \operatorname{polylog} n)$, an extension of Chan, Snoeyink, and Yap's divide-and-conquer algorithm [12] with running time $O((n + (nf)^{1-1/\lceil d/2 \rceil} + fn^{1-2/\lceil d/2 \rceil}) \operatorname{polylog} n)$, and an improvement of Seidel's shelling algorithm by Matoušek [35] with running time $O(n^{2-2/(\lfloor d/2 \rfloor + 1)} \operatorname{polylog} n + f \log n)$. For related results, see [6, 13, 17, 18, 34, 45].

Except when f is extremely small or extremely large, there are still large gaps between all these upper bounds and the lower bound $\Omega(n \log f + f)$. Moreover, most of these algorithms compute either the complete face lattice of the convex hull or a triangulation of its boundary, both of which can be significantly larger than the number of facets if the input is not in general position. Avis, Bremner, and Seidel [5] describe families of polytopes on which current convex hull algorithms perform quite badly, sometimes requiring exponential time (in d) even when the number of facets is only polynomial.

In this paper, we consider convex hull problems for which the result is a single integer, or even a single bit, although the convex hull itself may be large. We show that in the worst case, $\Omega(n^{\lceil d/2 \rceil - 1} + n \log n)$ sidedness queries are required to decide whether the convex hull of n points in $\mathbb{R}^d$ is simplicial or to determine the number of convex hull facets, where d is any fixed constant. This matches known upper bounds when d is odd [15]. The only lower bound previously known for either of these problems is $\Omega(n \log n)$, following from the techniques of Yao [48] and Ben-Or [7]. When the dimension is allowed to vary with the input size, deciding if a convex hull is simplicial is coNP-complete [14, 19], and counting the number of facets is #P-hard [19].

Our lower bounds follow from a straightforward adversary argument. We start by constructing a set whose convex hull contains a large number of independent degenerate facets. To obtain the adversary configuration, we perturb this set to eliminate the degeneracies, but in a way that the degeneracies are still "almost there." An adversary can reintroduce any one of the degenerate facets, by moving its vertices back to their original position, without changing the result of any other sidedness query.

Our argument is similar to earlier arguments of Erickson and Seidel [23]; however, many of the proofs in that paper were flawed [24]. Our proof technique yields correct and very simple proofs of Erickson and Seidel's claimed lower bounds for affine degeneracy detection in arbitrary dimensions and circular degeneracy detection in the plane.

The paper is organized as follows. Section 2 contains definitions and some preliminary results. In section 3, we describe some relative complexity results. Section 4 contains the proof of our main theorem. We discuss extensions of our model of computation in section 5. In section 6, we discuss the relevance of our results in light of existing convex hull algorithms. In section 7, we prove lower bounds for some related degeneracy-detection problems. Finally, in section 8, we summarize and suggest directions for further research.

2. Geometric preliminaries.

2.1. Definitions. We begin by reviewing basic terminology from the theory of convex polytopes. For a more detailed introduction, we refer the reader to Ziegler [49]

or Grünbaum [30].

The *convex hull* of a set of points is the smallest convex set that contains it. A *polytope* is the convex hull of a finite set of points. A hyperplane h *supports* a polytope if the polytope intersects h and lies in a closed half-space of h. The intersection of a polytope and a supporting hyperplane is called a *face* of the polytope. The *dimension* of a face is the dimension of the smallest affine space that contains it; a face of dimension k is called a *k-face*. The faces of a polytope are also polytopes. Given a d-dimensional polytope, its $(d-1)$-faces are called *facets*, its $(d-2)$-faces are called *ridges*, its 1-faces are called *edges*, and its 0-faces are called *vertices*.

A polytope is *simplicial* if all its facets, and thus all its faces, are simplices. A polytope is *quasi simplicial* if all of its ridges are simplices, or equivalently, if its facets are simplicial polytopes. A *degenerate facet* of a quasi-simplicial polytope is any facet that is not a simplex.

The basic computational primitive that we consider is the *sidedness query*: Given $d+1$ points $p_0, p_1, \ldots, p_d \in \mathbb{R}^d$, does the point p_0 lie "above," on, or "below" the oriented hyperplane determined by the other d points? Algebraically, the result of a sidedness query is given by the sign of the following $(d+1) \times (d+1)$ determinant, where p_{ij} denoted the jth coordinate of p_i:

$$\begin{vmatrix} 1 & p_{01} & p_{02} & \cdots & p_{0d} \\ 1 & p_{11} & p_{12} & \cdots & p_{1d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & p_{d1} & p_{d2} & \cdots & p_{dd} \end{vmatrix}.$$

The value of this determinant is $d!$ times the signed volume of the simplex spanned by the points. The algorithms we consider can be modeled as a family of decision trees, one for each possible value of n, in which every decision is based on the result of a sidedness query. (We will consider other computational primitives in section 5.)

The *orientation* of a simplex $(p_0, p_1, \ldots, p_d)$ is the result of a sidedness query on its vertices (in the order presented). If the orientation is zero, we say that the simplex is *degenerate*. A set of points is *affinely degenerate* if any $d+1$ of its elements lie on a single hyperplane, or equivalently, if the set contains the vertices of a degenerate simplex. The convex hull of an affinely nondegenerate set of points is simplicial, but the converse is not true in general—consider the regular octahedron in $\mathbb{R}^3$. Note that any $d+1$ vertices of a degenerate facet are also the vertices of a degenerate simplex.

2.2. The weird moment curve. The *weird moment curve* in $\mathbb{R}^d$, denoted $\omega_d(t)$, is the set of points

$$\omega_d(t) = (t, t^2, \ldots, t^{d-1}, t^{d+1}),$$

where the parameter t ranges over the reals. The weird moment curve is similar to the standard moment curve $(t, t^2, \ldots, t^{d-1}, t^d)$, except that the degree of the last coordinate is increased by 1.

If we project the weird moment curve down a dimension by dropping the last coordinate, we get a standard moment curve. Since every set of points on the standard moment curve is in convex position, every set of points on the d-dimensional weird moment curve is in convex position if $d \geq 3$. Similarly, since every set of points on the standard moment curve is affinely nondegenerate, no d points on the d-dimensional weird moment curve lie on a single $(d-2)$-flat. It follows immediately that the convex

hull of any set of points on the weird moment curve is quasi-simplicial; however, degenerate facets are possible.

LEMMA 2.1. *Let $x_0 < x_1 < \cdots < x_d$ be real numbers. The orientation of the simplex $(\omega_d(x_0), \omega_d(x_1), \ldots, \omega_d(x_d))$ is given by the sign of $\sum_{i=0}^{d} x_i$. In particular, the simplex is degenerate if and only if $\sum_{i=0}^{d} x_i = 0$.*

Proof. The orientation of the simplex $(\omega_d(x_0), \omega_d(x_1), \ldots, \omega_d(x_d))$ is given by the sign of the determinant of the following matrix:

$$M = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{d-1} & x_0^{d+1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{d-1} & x_1^{d+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_d & x_d^2 & \cdots & x_d^{d-1} & x_d^{d+1} \end{bmatrix}.$$

The determinant of M is an antisymmetric polynomial of degree $\binom{d+1}{2} + 1$ in the variables x_i, and it is divisible by $(x_i - x_j)$ for all $i < j$. It follows that

$$\frac{\det M}{\prod_{i<j}(x_j - x_i)}$$

is a symmetric polynomial of degree 1, and we easily observe that its leading coefficient is 1. (This polynomial is well defined, since the x_i's are distinct.) The only such polynomial is $\sum_{i=0}^{d} x_i$. □

This result, or at least its proof, is hardly new. If we replace the weird moment curve by any polynomial curve, the orientation of a simplex is given by the sign of a symmetric *Schur polynomial* [40]. A determinantal formula for Schur polynomials was discovered by Jacobi in the mid-1800s [32].[1]

The next lemma characterizes degenerate convex hull facets on the weird moment curve. The result is similar to Gale's "evenness condition" [27], which describes which vertices of a cyclic polytope form its facets.

LEMMA 2.2. *Let X be a set of real numbers, and let $x_0 < x_1 < \cdots < x_d$ be elements of X such that $\sum_{i=0}^{d} x_i = 0$. The points $\omega_d(x_0), \omega_d(x_1), \ldots, \omega_d(x_d)$ are the vertices of a degenerate facet of $\operatorname{conv}(\omega_d(X))$ if and only if, for any two elements $y, z \in X \setminus \{x_0, x_1, \ldots, x_d\}$, the number of elements of $\{x_0, x_1, \ldots, x_d\}$ between y and z is even.*

Proof. Let h be the hyperplane passing through the points $\omega_d(x_0), \omega_d(x_1), \ldots, \omega_d(x_d)$. For any real number x, the point $\omega_d(x)$ lies above, on, or below h according

[1] Jacobi proved that for any nonnegative integers $\gamma_0, \gamma_1, \ldots, \gamma_d$,

$$\begin{vmatrix} x_0^{\gamma_0} & x_0^{\gamma_1} & \cdots & x_0^{\gamma_d} \\ x_1^{\gamma_0} & x_1^{\gamma_1} & \cdots & x_1^{\gamma_d} \\ \vdots & \vdots & \ddots & \vdots \\ x_d^{\gamma_0} & x_d^{\gamma_1} & \cdots & x_d^{\gamma_d} \end{vmatrix} = \begin{vmatrix} \Sigma_{\gamma_0} & \Sigma_{\gamma_1} & \cdots & \Sigma_{\gamma_d} \\ \Sigma_{\gamma_0 - 1} & \Sigma_{\gamma_1 - 1} & \cdots & \Sigma_{\gamma_d - 1} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{\gamma_0 - d} & \Sigma_{\gamma_1 - d} & \cdots & \Sigma_{\gamma_d - d} \end{vmatrix} \cdot \prod_{0 \le i < j \le d} (x_j - x_i),$$

where Σ_k is the sum of all possible monomials of total degree k in the variables $x_0, x_1, \ldots, x_d$. In particular, $\Sigma_0 = 1$ and $\Sigma_k = 0$ for all $k < 0$.

to the sign of the determinant

$$\begin{vmatrix} 1 & x & x^2 & \cdots & x^{d-1} & x^{d+1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{d-1} & x_1^{d+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_d & x_d^2 & \cdots & x_d^{d-1} & x_d^{d+1} \end{vmatrix} = \left(\prod_{1 \le i < j \le d} (x_j - x_i) \right) \left(\prod_{i=1}^{d} (x - x_i) \right) \left(x + \sum_{i=1}^{d} x_i \right)$$

$$= \left(\prod_{1 \le i < j \le d} (x_j - x_i) \right) \left(\prod_{i=0}^{d} (x - x_i) \right).$$

(See the proof of Lemma 2.1.) Since all factors of the form $(x_j - x_i)$ are positive, the sign of this determinant is equal to the sign of the polynomial $f(x) = \prod_{i=0}^{d}(x - x_i)$. The hyperplane h supports $\text{conv}(\omega_d(X))$ if and only if $f(x)$ has the same sign for all $x \in X \setminus \{x_0, x_1, \ldots, x_d\}$.

The polynomial $f(x)$ has degree $d+1$ and vanishes at each x_i. Thus, the sign of $f(x)$ changes at each x_i. In more geometric terms, the weird moment curve crosses the hyperplane h at each of the points $\omega_d(x_i)$. It follows that $f(y)$ and $f(z)$ both have the same sign if and only if an even number of x_i's lie between y and z. □

3. $\lceil d/2 \rceil$SUM hardness. Gajentaan and Overmars [26] define the class of 3*SUM-hard* problems, all of which are harder than the following base problem:

3SUM: Given a set of n distinct integers, do any three sum to zero?

This problem can be easily solved in $O(n^2)$ time, which is believed to be optimal, but the best lower bound in any general model of computation is only $\Omega(n \log n)$ [7].

Formally, a problem is 3SUM-hard if there is a subquadratic reduction from 3SUM to the problem in question. Thus, a subquadratic algorithm for any 3SUM-hard problem would imply a subquadratic algorithm for 3SUM, and a sufficiently powerful quadratic lower bound for 3SUM would imply similar lower bounds for every 3SUM-hard problem. (For this reason, some earlier papers call these problems "n^2-hard," but see [8].) Examples of 3SUM-hard problems include several degeneracy detection, separation, hidden surface removal, and motion planning problems in two and three dimensions.

More generally, we will say that a problem is *rSUM-hard* if the following problem can be reduced to it in $o(n^{\lceil r/2 \rceil})$ time:

rSUM: Given a set of n distinct integers, do any r sum to zero?

The problem rSUM can be solved in time $O(n^{(r+1)/2})$ when r is odd or in time $O(n^{r/2} \log n)$ when r is even. We conjecture that these algorithms are optimal; however, the best lower bound in any general model of computation, for any fixed r, is again only $\Omega(n \log n)$ [7]. Higher-dimensional versions of many 3SUM-hard problems are rSUM-hard for larger values of r. For example, Lemma 2.1 immediately implies that detecting affine degeneracies in $\mathbb{R}^d$ is dSUM-hard.

THEOREM 3.1. *Deciding whether the convex hull of n points in $\mathbb{R}^d$ is simplicial, for any fixed d, is $\lceil d/2 \rceil$SUM-hard.*

Proof. We describe the proof explicitly only for the case $d = 5$; generalizing the proof to higher dimensions is straightforward.

Given a set of integers $X = \{x_1, x_2, \ldots, x_n\}$, we first replace them with the larger set $X' = \{x_1^\flat, x_1^\sharp, x_2^\flat, x_2^\sharp, \ldots, x_n^\flat, x_n^\sharp\}$, where $x_i^\flat = x_i - 2^{-i}$ and $x_i^\sharp = x_i + 2^{-i}$ for all i.

We then consider the points $\omega_5(X')$ obtained by lifting X' onto the weird moment curve in $\mathbb{R}^5$. To prove the theorem, it suffices to show that the convex hull of the points $\omega_5(X')$ is nonsimplicial if and only if some three elements of X sum to zero.

Suppose that the convex hull of $\omega_5(X')$ is nonsimplicial. Then some six points in $\omega_5(X')$ lie on the same hyperplane. By Lemma 2.1, the corresponding six elements of X' sum to zero. These must consist of three matched pairs $a^\flat, a^\sharp, b^\flat, b^\sharp, c^\flat, c^\sharp$ for some $a, b, c \in X$. Otherwise, the various "fudge factors" $\pm 2^{-i}$ do not cancel out, and the sum of six elements is not even an integer. Thus, X has three elements whose sum is zero.

Conversely, suppose that $a + b + c = 0$ for some $a, b, c \in X$. This immediately implies $a^\flat + a^\sharp + b^\flat + b^\sharp + c^\flat + c^\sharp = 0$, and thus, by Lemma 2.1, the corresponding points in $\omega_5(X')$ all lie on a single hyperplane. Moreover, by Lemma 2.2, this hyperplane supports a facet of the convex hull of $\omega_5(X')$, since no other elements of X' lie in the intervals $(a^\flat, a^\sharp)$, $(b^\flat, b^\sharp)$, or $(c^\flat, c^\sharp)$. Thus, the convex hull of $\omega_5(X')$ is not simplicial. □

The best lower bound we can ever hope to derive using this reduction is $\Omega(n^{\lceil d/4 \rceil} + n \log n)$, which is significantly smaller than the best known upper bound $O(n^{\lfloor d/2 \rfloor} + n \log n)$, except for the single case $d = 5$. In particular, Theorem 3.1 tells us absolutely nothing about the four-dimensional case, since we already have a lower bound of $\Omega(n \log n)$ in all dimensions.

In an earlier paper [21], we derive an $\Omega(n^{\lceil r/2 \rceil})$ lower bound for rSUM in the r-linear decision tree model. In this model, decisions are based on the signs of arbitrary affine combinations of r or fewer input variables. Unfortunately, since the reduction described by the previous theorem does not follow this model, we do not automatically get similar lower bounds for detecting simplicial convex hulls. In the next section of the paper, we derive such lower bounds directly.

Remark. If r is not fixed, the problem rSUM is NP-complete, by a simple reduction to SUBSET SUM [28]. We can use this fact to give simple proofs that certain geometric problems in arbitrary dimensions are NP-hard. For example, Khachiyan [33] proves that detecting affine degeneracies is NP-complete. This result follows directly from Lemma 2.1. Chandrasekaran, Kabadi, and Murty [14] and Dyer [19] independently prove that deciding whether the convex hull of a set of points is simplicial is coNP-complete; this result also follows immediately from Theorem 3.1. Moreover, since the reductions are parsimonious [28], the corresponding counting problems (how many degenerate simplices/facets?) are #P-complete.

4. Lower bounds for convex hull problems. Our main result is based on the following combinatorial construction.

LEMMA 4.1. *For all n and d, there is a quasi-simplicial polytope in $\mathbb{R}^d$ with $O(n)$ vertices and $\Omega(n^{\lceil d/2 \rceil - 1})$ degenerate facets.*

Proof. First consider the case when d is odd, and let $r = (d-1)/2$. Without loss of generality, we assume that n is a multiple of r. Let X denote the following set of $n + 2n/r = O(n)$ integers:

$$X = \{-rn, -rn + r, \ldots, -r;\ r, r+1, 2r, 2r+1, \ldots, n, n+1\}.$$

We can specify a degenerate facet of $\omega_5(X)$ as follows. Choose r distinct elements $a_1, a_2, \ldots, a_r \in X$, all positive multiples of r. Let $a_0 = -\sum_{i=1}^{r} a_i$, let $b_0 = a_0 + r$, and for all $i > 0$, let $b_i = a_i + 1$. Each a_i and b_i is a unique element of X, and no element of X lies between a_i and b_i for any i. The $d+1$ points $\omega_d(a_0), \omega_d(b_0), \ldots, \omega(a_r), \omega(b_r)$

all lie on a single hyperplane by Lemma 2.1, since

$$\sum_{i=0}^{r}(a_i + b_i) = 2\sum_{i=0}^{r} a_i = 0.$$

Moreover, since any pair of elements of $X \setminus \{a_0, b_0, a_1, b_1, \ldots, a_r, b_r\}$ has an even number of elements of $\{a_0, b_0, \ldots, a_r, b_r\}$ between them, Lemma 2.2 implies that these points are the vertices of a single facet of $\text{conv}(\omega_d(X))$. There are at least $\binom{n/r}{r} = \Omega(n^r)$ ways of choosing such a degenerate facet.

In the case where d is even, let $r = d/2 - 1$, and assume without loss of generality that n is a multiple of r. Let X be the following set of $n + 2n/r + 1 = O(n)$ integers:

$$X = \{-n - rn, -n - rn + r, \ldots, -n - r;\ r, r + 1, 2r, 2r + 1, \ldots, n, n + 1;\ 2n\}.$$

Using similar arguments as above, we easily observe that the polytope $\text{conv}(\omega_d(X))$ has $\Omega(n^r)$ degenerate facets, each of which has $\omega_d(2n)$ as a vertex. □

This result is the best possible when d is odd, since an odd-dimensional n-vertex polytope has at most $O(n^{(d-1)/2})$ facets [49]. In the case where d is even, the best known upper bound is $O(n^{d/2})$, which is a factor of n bigger than the result we prove here. We conjecture that this upper bound is tight. However, if we consider only sets of points on the weird moment curve, the bound given in the lemma is tight. That is, the convex hull of any set of n points on ω_d has at most $O(n^{\lceil d/2 \rceil - 1})$ degenerate facets.

We now prove the main result of the paper.

THEOREM 4.2. *Any decision tree that decides whether the convex hull of a set of n points in $\mathbb{R}^d$ is simplicial, using only sidedness queries, must have depth $\Omega(n^{\lceil d/2 \rceil - 1} + n \log n)$.*

Proof. Let X be the set of numbers described in the proof of Lemma 4.1, and let $X' = X + 1/(2d+2) = \{x + 1/(2d+2) \mid x \in X\}$. Initially, the adversary presents the set of points $\omega_d(X')$. Since $\sum_{i=0}^{d} x'_i$ is always a half-integer, this point set is affinely nondegenerate, so its convex hull is simplicial.

It suffices to consider the case where d is odd. Let $r = (d-1)/2$. Choose distinct elements $a'_0, b'_0, a'_1, b'_1, \ldots, a'_r, b'_r \in X'$ so that $\sum_{i=0}^{r}(a'_i + b'_i) = 1/2$ and no other elements of X' lie between a'_i and b'_i for any i. The corresponding points $\omega(a'_0), \omega(b'_0), \ldots, \omega(a'_r), \omega(b'_r)$ form a *collapsible simplex*. To collapse it, the adversary simply moves the points back to their original positions in $\omega_d(X)$. Lemmas 2.1 and 2.2 imply that the collapsed simplex forms a degenerate facet of the new convex hull. Since the sum of any other $(d+1)$-tuple changes by at most $1/2 - 1/(2d+2)$, no other simplex changes orientation. In other words, the only way for an algorithm to distinguish between the original configuration and the collapsed configuration is to perform a sidedness query on the collapsible simplex.

Thus, if an algorithm does not perform a separate sidedness query for every collapsible simplex, then the adversary can introduce a degenerate facet that the algorithm cannot detect. There are $\Omega(n^{\lceil d/2 \rceil - 1})$ collapsible simplices, one for each degenerate facet in $\text{conv}(\omega_d(X))$.

Finally, the $n \log n$ term follows from the algebraic decision tree lower bound of Ben-Or [7]. □

A three-dimensional version of our construction is illustrated in Figure 4.1. (See also the proof of Theorem 7.4 below.)

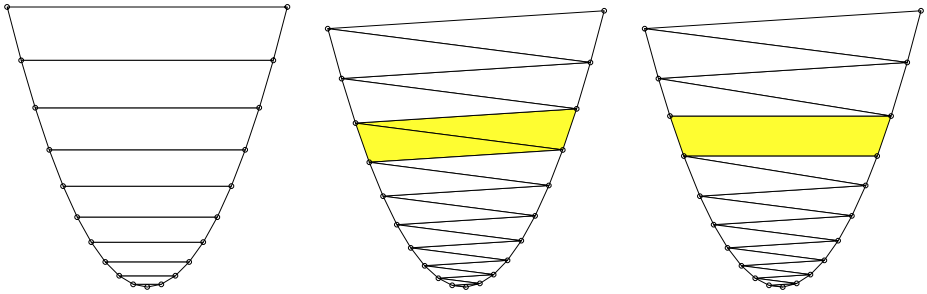


FIG. 4.1. *Our adversary construction in three dimensions. Bottom views of* (a) *a quasi-simplicial polytope with* $\Omega(n)$ *degenerate facets,* (b) *the simplicial adversary polytope with one collapsible simplex highlighted, and* (c) *the corresponding collapsed polytope.*

Our lower bound matches known upper bounds when d is odd [15]. We emphasize that if the points are known *in advance* to lie on the weird moment curve, this problem can be solved in $O(n^{\lceil d/4 \rceil})$ time if $\lceil d/2 \rceil$ is odd, and in $O(n^{\lceil d/4 \rceil} \log n)$ time if $\lceil d/2 \rceil$ is even, by an algorithm that uses more complicated queries not allowed by Theorem 4.2, namely, evaluating the signs of certain linear combinations of x_1-coordinates. (See [21].)

The convex hull of the adversary configuration $\omega_d(X')$ has $\lceil d/2 \rceil - 1$ more facets than the convex hull of any collapsed configuration. Thus, we immediately have the following lower bound.

THEOREM 4.3. *Any decision tree that computes the number of convex hull facets of a set of* n *points in* $\mathbb{R}^d$*, using only sidedness queries, must have depth* $\Omega(n^{\lceil d/2 \rceil - 1} + n \log n)$.

A simple modification of our argument implies the following "output-sensitive" version of our lower bound.

THEOREM 4.4. *Any decision tree that decides whether the convex hull of a set of* n *points in* $\mathbb{R}^d$ *is simplicial or computes the number of convex hull facets, using only sidedness queries, must have depth* $\Omega(f)$ *when* d *is odd and* $\Omega(f^{1-2/d})$ *when* d *is even, where* f *is the number of faces of the convex hull.*

Proof. Assume that $f > n$, since otherwise we have nothing to prove. We construct a modified degenerate polytope as follows. We start by constructing a degenerate polytope with f faces, exactly as described in the proof of Lemma 4.1. When d is odd, this polytope is the convex hull of $\Theta(f^{2/(d-1)})$ points on the weird moment curve and has $\Omega(f)$ degenerate facets. When d is even, the polytope is the convex hull of $\Theta(f^{2/d})$ points and has $\Omega(f^{1-2/d})$ degenerate facets.

By introducing a new vertex extremely close to the relative interior of any facet of a simplicial polytope, we can split that facet into d smaller facets. Each such split increases the number of polytope faces by $2^d - 2$. To bring the number of vertices of our adversary polytope up to n, we choose some facet and repeatedly split it in this fashion, being careful not to introduce any new degenerate simplices. The augmented polytope has at most $f + (2^d - 2)n = O(f)$ faces.

To get a modified *adversary* polytope, we slide the original vertices of the degenerate polytope along the weird moment curve, just enough to remove the degeneracies, leaving the new vertices in place. Each of the degenerate facets becomes a collapsible simplex. As long as we do not slide the vertices too far, collapsing a simplex will

not change the orientation of any simplex involving a new vertex. (In effect, we are treating sidedness queries involving new vertices as "allowable" queries; see below.) The lower bound now follows from the usual adversary argument. □

5. Other computational primitives. In this section, we identify a general class of computational primitives which, if added to our model of computation, do not affect our lower bounds. In fact, even if we allow any finite number of these primitives to be performed at no cost, the number of required sidedness queries is the same. These primitives include comparisons between coordinates of input points in any number of directions, comparisons between coordinates of hyperplanes defined by d-tuples of points, and in-sphere queries.

The primitives we consider are all *algebraic queries*. The result of an algebraic query is given by the sign of a multivariate *query polynomial*, evaluated at the coordinates of the input. If the sign is zero (resp., nonzero), we say that the input is *degenerate* (resp., *nondegenerate*) with respect to that query. For example, a set of points is affinely degenerate if and only if it is degenerate with respect to some sidedness query.

A *projective transformation* of $\mathbb{R}^d$ (or more properly, of the projective space $\mathbb{RP}^d$) is any map that takes hyperplanes to hyperplanes. If we represent the points of $\mathbb{R}^d$ in homogeneous coordinates, a projective transformation is equivalent to a linear transformation of $\mathbb{R}^{d+1}$. In Stolfi's two-sided projective model [46], projective maps preserve (or reverse) the orientation of every simplex in $\mathbb{R}^d$ and thus preserve the combinatorial structure of convex hulls. (See Chapter 14 of [46].)

Let X be the set of numbers described in the proof of Lemma 4.1. We call an algebraic query *allowable* if for some projective transformation ϕ the configuration $\phi(\omega_d(X))$ is nondegenerate with respect to that query. Our choice of terminology is justified by the following theorem.

THEOREM 5.1. *Any decision tree that decides whether the convex hull of n points in $\mathbb{R}^d$ is simplicial, using only sidedness queries and a finite number of allowable queries, requires $\Omega(n^{\lceil d/2\rceil-1})$ sidedness queries in the worst case.*

Proof. If *some* projective transformation makes $\omega_d(X)$ nondegenerate with respect to an algebraic query, then *almost every* projective transformation (i.e., all but a measure zero subset) makes $\omega_d(X)$ nondegenerate. Thus, for any finite set of allowable queries, almost every projective transformation makes $\omega_d(X)$ nondegenerate with respect to all of them. Let ϕ be such a transformation.

If $\phi(\omega_d(X))$ is nondegenerate with respect to some finite set of allowable queries, then for all X' in an open neighborhood of X in $\mathbb{R}^n$, the configuration $\phi(\omega(X'))$ is also nondegenerate with respect to that set of queries.

The theorem now follows from a slight modification of the proof of Theorem 4.2. Let $\varepsilon > 0$ be some sufficiently small real number. The set $\phi(\omega_d(X + \varepsilon))$ has a simplicial convex hull but has $\Omega(n^{\lceil d/2\rceil+1})$ collapsible simplices, each corresponding to a degenerate facet in $\phi(\omega_d(X))$. No allowable query can distinguish between $\phi(\omega_d(X + \varepsilon))$ and any collapsed configuration, or even between $\phi(\omega_d(X + \varepsilon))$ and $\phi(\omega_d(X))$. □

We characterize allowable queries algebraically as follows. Consider the degenerate configuration $\omega_d(X)$ as a single point in the configuration space $\mathbb{R}^{dn}$. Any algebraic query induces a surface in configuration space, consisting of all configurations that are degenerate with respect to that query. Since any projective map ϕ can be represented by a $(d + 1) \times (d + 1)$ matrix with determinant ± 1, the set of projectively transformed configurations $\phi(\omega_d(X))$ forms a $(d^2 + 2d)$-dimensional algebraic

variety in configuration space. Any query whose surface does not completely contain this variety is allowable.

We give below a (nonexhaustive!) list of allowable queries. We leave the proofs that these queries are in fact allowable as easy exercises.

- Comparisons of point coordinates, or more generally comparing inner products of two points with a fixed direction vector, is allowable. In fact, we can allow the input points to be presorted in any finite number of fixed directions. Seidel describes a similar result in the context of three-dimensional convex hull lower bounds [42, Theorem 5]. We emphasize that the directions in which these comparisons are made must be fixed in advance. No matter how we transform the adversary configuration, there is always *some* direction in which a point comparison can distinguish it from a collapsed configuration.
- More generally, deciding which of two points is hit first by a hyperplane rotating around a fixed $(d-2)$-flat is allowable. We can even presort the points by their cyclic orders around any finite number of fixed $(d-2)$-flats. If the $(d-2)$-flat is "at infinity," then "rotation" is just translation, and we have the previous notion of point comparison. We can interpret this type of query in dual space as a comparison between the intersections of two hyperplanes with a fixed line. Again, we emphasize that the $(d-2)$-flats must be fixed in advance.
- Sidedness queries in any fixed lower-dimensional projection are allowable. This is a natural generalization of point comparisons, which can be considered sidedness queries in a one-dimensional projection. We can even specify in advance the complete order types of the projections onto any finite number of fixed affine subspaces. (As a technical point, we would not actually include this information as part of the input, since this would drastically increase the input size; instead, knowledge of the projected order types would be hardwired into the algorithm.)
- "Second-order" comparisons between vertices of the dual hyperplane arrangement, in any fixed direction, are also allowable. Such a query can be interpreted in the primal space as a comparison between the intersections of two hyperplanes, each defined by a d-tuple of input points, with a fixed line. To prove that such a query is allowable, it suffices to observe that a projective transformation of the primal space induces a projective transformation of the dual space, and vice versa. Note that a second-order comparison is algebraically equivalent to a sidedness query if the two d-tuples share $d-1$ points.
- Since most projective transformations do not map spheres to spheres, in-sphere queries are allowable. Given $d+2$ points, an in-sphere query asks whether the first point lies "inside," on, or "outside" the oriented sphere determined by the other $d+1$ points. Similarly, in-sphere queries in any fixed lower-dimensional projection are allowable.
- Distance comparisons between pairs of points or pairs of projected points are allowable. More generally, comparing the measures of pairs of simplices of dimension less than d—for example, comparing the areas of two triangles when $d > 2$—defined either by the original points or by any fixed projection, are allowable.

However, comparing the volumes of arbitrary simplices of *full* dimension is not allowable. In any projective transformation of $\omega_d(X)$, all of the degenerate simplices

have the same (zero) volume. It is not possible to collapse a simplex in any adversary configuration while maintaining the order of the volumes of the other collapsible simplices.

6. Our models vs. real convex hull algorithms. A large number of convex hull algorithms rely (or can be made to rely) exclusively on sidedness queries. These include the "gift-wrapping" algorithms of Chand and Kapur [13] and Swart [47], the "beneath-beyond" method of Seidel [41], Clarkson and Shor's [18] and Seidel's [44] randomized incremental algorithms, Chazelle's worst-case optimal algorithm [15], and the recursive partial-order algorithm of Clarkson [17]. Seidel's "shelling" algorithm [43] and the space-efficient gift-wrapping algorithms of Avis and Fukuda (at least if Bland's pivoting rule is used) [6] and Rote [39] require only sidedness queries and second-order comparisons.

Matoušek [35] and Chan [11] improve the running times of these algorithms (in an output-sensitive sense) by finding the extreme points more quickly. Clarkson [17] describes a similar improvement to a randomized incremental algorithm. Since every point in our adversary configuration is extreme, our lower bound still holds even if the extremity of a point can be decided for free. We are not suggesting that the computational primitives used by these algorithms cannot be used to break our lower bounds, only that the ways in which these primitives are currently applied are inherently limited.

Chan [11] describes an improvement of the gift-wrapping algorithm that uses ray shooting data structures of Agarwal and Matoušek [1] and Matoušek and Schwarzkopf [36] to speed up the pivoting step. In each pivoting step, the gift-wrapping algorithm finds a new facet containing a given ridge of the convex hull. In the dual, this is equivalent to shooting a ray from a vertex of the dual polytope along one of its outgoing edges. The dual vertex that the ray hits corresponds in the primal to the new facet. A single pivoting step tells us the orientation of $n - d$ simplices, all of which share the d vertices of the new facet. However, at most one of these simplices can be collapsible, since two collapsible simplices share at most $d/2$ vertices. Thus, even if we allow a pivoting step to be performed in constant time, our lower bound still holds.

There are a few convex hull algorithms which seem to fall outside our framework, most notably the divide-and-conquer algorithm of Chan, Snoeyink, and Yap [12], and its improvement by Amato and Ramos [2]. The four-dimensional version of their algorithm uses primitives involving up to 22 points.[2] Higher-dimensional versions of their algorithm require the use of linear programming queries and ray-shooting queries in certain $(d-1)$-dimensional projections of the input; the fastest known algorithms to answer these queries [1, 11, 35, 36] do not even fit into the algebraic decision tree model.

7. Related problems.

7.1. Affine degeneracies.

THEOREM 7.1. *Any decision tree that decides whether a set of n points in $\mathbb{R}^d$ is affinely nondegenerate, using only sidedness queries, must have depth $\Omega(n^d)$. If*

[2]The most elaborate primitive is a sidedness query on a three-dimensional projection of four input points, where the direction of projection is defined by the intersection of three planes, each the affine hull of three points, each the intersection of a fixed hyperplane and the affine hull of two input points.

$d \geq 3$, this lower bound holds even when the points are known in advance to be in convex position.

Proof. Let X denote the set of integers from $-dn$ to n, and let $X' = X + 1/(2d+2)$. The adversary initially presents the point set $\omega_d(X')$. This point set is affinely nondegenerate, since the expression $\sum_i x'_i$ is always a half-integer.

Choose arbitrary distinct positive elements $x_1, x_2, \ldots, x_d \in X$, and let $x_0 = -\sum_i x_i$; this is also an element of X. Then the points $\omega_d(x'_i)$ form a collapsible simplex. To collapse it, the adversary just shifts the points back down to $\omega_d(x_i)$; the collapsed simplex is obviously degenerate. Since the expression $\sum_{i=0}^{d} x'_i$ changes by at most $1/2 - 1/(2d+2)$ for any other simplex, no other simplex changes orientation.

Thus, if an algorithm does not perform a sidedness query on every collapsible simplex, the adversary can introduce an affine degeneracy that the algorithm cannot detect. There are at least $\binom{n}{d} = \Omega(n^d)$ such simplices. If $d \geq 3$, the original point set and each collapsed point set is in convex position. □

Erickson and Seidel [23] prove an $\Omega(n^d)$ lower bound for a restricted problem: Do any $d+1$ points lie on a *nonvertical* hyperplane? Except in the two-dimensional case, where an explicit adversary construction is given, their extension to the general problem is flawed [24].

The previous theorem easily generalizes to allow additional queries, as described in section 5.

THEOREM 7.2. *Any decision tree that decides whether a set of n points in $\mathbb{R}^d$ is affinely nondegenerate, using only sidedness queries and a finite number of allowable queries, requires $\Omega(n^d)$ sidedness queries in the worst case. If $d \geq 3$, this lower bound holds even when the points are known in advance to be in convex position.*

7.2. An alternate proof in two dimensions. According to Grünbaum [31], A. H. Stone observed that a set of n integer points on the unit cubic can have $n^2/8$ collinear triples. Füredi and Palásti [25] discovered an elegant construction, which we describe below, that improves this lower bound to roughly $n^2/6$. We can use their construction to slightly improve our lower bound for the two-dimensional affine degeneracy problem. The resulting lower bound is the best that can be derived using our techniques, except possibly for some lower-order terms.

Füredi and Palásti describe their construction in the dual. Let $L(\alpha)$ be the line passing through the point $(\cos\alpha, \sin\alpha)$ at angle $-\alpha/2$ to the x-axis. The line $L(\alpha)$ also passes through the point $(cos(\pi - 2\alpha), \sin(\pi - 2\alpha))$; if this is the same point as $(\cos\alpha, \sin\alpha)$, then the line is tangent to the unit circle at that point. Three lines $L(\alpha), L(\beta), L(\gamma)$ are concurrent if and only if $\alpha + \beta + \gamma \equiv 0 \pmod{2\pi}$. It follows that the set of lines $\{L(2\pi i/n) \mid 1 \leq i \leq n\}$ has $1 + \lfloor n(n-3)/6 \rfloor$ concurrent triples. See Figure 7.1(a). See [25] for further details. Related results are described in [9, 20, 31].

The set of lines $\{L((2i-1)\pi/n) \mid 1 \leq i \leq n\}$ has no concurrent triples, but its arrangement has $\lceil n(n-3)/3 \rceil$ triangular cells, each bounded by a triple of lines of the form

$$L((2i-1)\pi/n),\ L((2j-1)\pi/n),\ L((2k-1)\pi/n),$$

where $i + j + k \equiv 1$ or $2 \pmod{n}$. See Figure 7.1(b). Each of these triangles is collapsible; to collapse such a triangle, we shift each of its three defining lines by $\pi/3n$, resulting in the lines

$$L((2i-2/3)\pi/n),\ L((2j-2/3)\pi/n),\ L((2k-2/3)\pi/n)$$

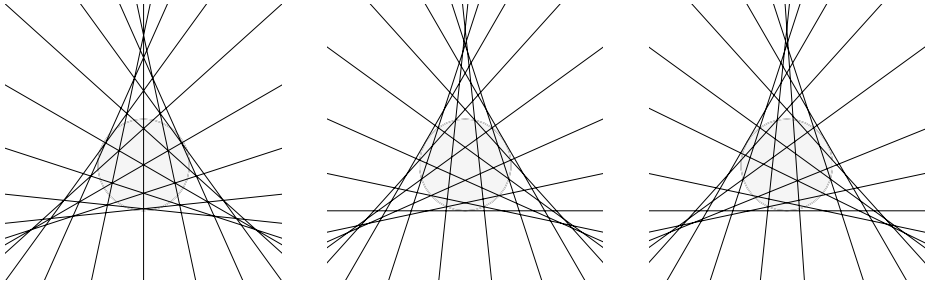


FIG. 7.1. *Another adversary construction for arbitrary degeneracies in the plane, following a construction of Füredi and Palásti.* (a) *The degenerate configuration.* (b) *The adversary configuration.* (c) *A collapsed configuration.*

if $i + j + k \equiv 1 \pmod{n}$, or

$$L((2i - 4/3)\pi/n),\ L((2j - 4/3)\pi/n),\ L((2k - 4/3)\pi/n)$$

if $i+j+k \equiv 2 \pmod{n}$; see Figure 7.1(c). We easily verify that the collapsed triangle is degenerate and that no other triangle changes orientation, since the sum of any other triple of defining angles changes by at most $2\pi/3n < \pi/n$.

THEOREM 7.3. *Any decision tree that decides whether a set of n points in $\mathbb{R}^2$ is affinely degenerate, using only sidedness queries, must have depth at least $\lceil n(n-3)/3 \rceil$.*

Grünbaum [31] proves that a simple arrangement of n lines in the projective plane can have at most $\lfloor n(n-1)/3 \rfloor$ triangular cells if n is even, and at most $\lfloor n(n-2)/3 \rfloor$ if n is odd. Thus, we cannot hope to prove a lower bound bigger than $n^2/3 + \Omega(n)$ using collapsible triangles.

7.3. Circular degeneracies. We also easily prove the following related theorem first proved in [23]. A set of points in the plane is *circularly degenerate* if any four points lie on a circle. The basic computational primitive used to detect circular degeneracies is the *in-circle query*: Given four points, is the first point inside, on, or outside the oriented circle defined by the other three points? In-circle queries can be answered by lifting the points to the unit paraboloid $z = x^2 + y^2$, or stereographically projecting them onto a sphere, and performing a three-dimensional sidedness query.

THEOREM 7.4. *Any decision tree that decides whether n points in $\mathbb{R}^2$ is circularly degenerate, using only in-circle queries, must have depth $\Omega(n^3)$.*

Proof. An in-circle query on four points on the unit parabola (t, t^2) is algebraically equivalent to a sidedness query for four points on the three-dimensional weird moment curve (t, t^2, t^2). Thus, Lemma 2.1 implies that four points, $(a, a^2), (b, b^2), (c, c^2)$, and (d, d^2), on the unit parabola are cocircular if and only if $a + b + c + d = 0$. Let X be the set of integers from $-n$ to n. There are $\Theta(n^3)$ 4-tuples in X whose sums are zero. The adversary presents a set of points on the unit parabola with x-coordinates taken from the set $X + 1/8$. This set is nondegenerate and has $\Omega(n^3)$ collapsible 4-tuples. □

We can extend the model of computation in a similar fashion as before, but with a different set of new queries. A *linear fractional transformation* of the plane (or more formally, of the Riemann sphere $\mathbb{CP}^1$) is any transformation that maps circles

to circles. If we represent the points of $\mathbb{R}^2$ in complex homogeneous coordinates—representing $(x, y) \in \mathbb{R}^2$ by any complex multiple of $(1 + 0i, x + yi) \in \mathbb{C}^2$—then a linear fractional transformation is equivalent to a linear transformation of $\mathbb{C}^2$.

We say that a query is *circularly allowable* if some linear fractional transformation of the set (X, X^2) is nondegenerate with respect to that query, where X is the set of numbers described in the proof of Theorem 7.4. Circularly allowable queries include first- and second-order point comparisons and sidedness queries but do not include comparisons between arbitrary in-circle determinants.

Arguments similar to those in section 5 give us the following theorem.

THEOREM 7.5. *Any decision tree that decides whether n points in $\mathbb{R}^2$ is circularly degenerate, using only in-circle queries and a finite number of circularly allowable queries, requires $\Omega(n^3)$ in-circle queries in the worst case.*

We conjecture that $\Omega(n^{d+1})$ insphere queries are required to decide if a set of n points in $\mathbb{R}^d$ is spherically degenerate, but we have been unable to generalize our proof of the two-dimensional case to higher dimensions. A proof would follow immediately from the construction of a set of numbers having $\Omega(n^{d+1})$ $(d+2)$-tuples in the zeroset of a certain symmetric polynomial, by applying our usual adversary argument. For example, in three dimensions, we need $\Omega(n^4)$ 5-tuples in the zeroset of the polynomial

$$1 + \sum_{1 \le i \le j \le 5} x_i x_j.$$

Erickson and Seidel [23] prove that $\Omega(n^{d+1})$ in-sphere queries are required to detect *proper* spherical degeneracies, i.e., sets of $d + 2$ points on a sphere of finite radius, but their proof for the general problem was flawed [24]. Unlike all the adversary sets in this paper, the adversary set they use is *not* obtained by perturbing a highly degenerate point set. Is there a set of n points in $\mathbb{R}^d$ with $\Omega(n^{d+1})$ independent spherical degeneracies? Such a set might lead to a proof of our conjecture.

8. Conclusions and open problems. We have presented new lower bounds on the worst-case complexity of detecting simplicial convex hulls or counting convex hull facets in a fairly natural model of computation. Our lower bounds follow from a simple adversary argument, based on the construction of a convex polytope with a large number of degenerate features. In order to be correct, any algorithm must individually check that each of those degenerate features is not present in the input. Similar arguments give us simple proofs of lower bounds for several degeneracy-detection problems.

Several open problems remain to be answered. While our lower bounds match existing upper bounds in odd dimensions, there is still a gap when the dimension is even. A first step in improving our lower bounds would be to improve the combinatorial bounds in Lemma 4.1. Is there a four-dimensional polytope with n vertices and $\Omega(n^2)$ degenerate facets? However, we conjecture that no such polytope (or even polyhedral 3-sphere) exists. Simple variations on the weird moment curve will not suffice, since an "evenness condition" like Lemma 2.2 always forces the number of degenerate facets to be linear. Arguments based on merging facets of cyclic or product polytopes also fail, as do variations on Amenta and Ziegler's deformed products [3, 4]. The best example we can construct is the connected sum of $n/5 - 1$ copies of a bipyramid over a cube, which has n vertices and $2n - 8$ facets, each a square pyramid.

A common application of convex hull algorithms is the construction of Delaunay triangulations and Voronoi diagrams. Are $\Omega(n^{\lceil d/2 \rceil})$ in-sphere queries required to decide if the Delaunay triangulation is simplicial (i.e., really a triangulation)? Again, a

first step is to construct a Delaunay triangulation with $\Omega(n^{\lceil d/2 \rceil})$ independent degenerate features.

Another similar problem is deciding, given a set of points, which ones are vertices of the set's convex hull. This problem can be decided in $O(n^2)$ time (using only sidedness queries!) by invoking a linear-time linear programming algorithm once for each point [37]. This upper bound can be improved to $O(n^{2\lfloor d/2 \rfloor/(\lfloor d/2 \rfloor - 1)} \operatorname{polylog} n)$ using an algorithm due to Chan [11]. Except for the polylogarithmic term, this algorithm is almost certainly optimal, but as usual the only known lower bound is $\Omega(n \log n)$ [7]. It seems unlikely that a collapsible simplex argument could be used to imply a reasonable lower bound for this problem.

Another interesting open problem is to strengthen the models in which our lower bounds hold. Quadratic lower bounds for either the five-dimensional convex hull problem or the two-dimensional affine degeneracy problem in stronger models of computation would imply similar lower bounds for a number of other 3SUM-hard problems. While the lower bounds we prove here and in earlier papers [23, 21] are in fairly natural models, there are still 3SUM-hard problems that cannot even be solved in these models. For example, one of the simplest problems for which our techniques fail is finding the minimum area triangle determined by a set of points in the plane. In order to prove a useful lower bound for this problem, we must consider a model that allows comparison of signed triangle areas. It seems impossible to apply our "collapsible simplex" adversary argument in such a model; a radically new idea is called for.

Ultimately, of course, we would like a lower bound bigger than $\Omega(n \log n)$ that holds in some general model of computation, such as algebraic decision trees or algebraic computation trees.

Acknowledgments. I would like to thank Raimund Seidel for several helpful discussions, David Bremner for pointing out the NP-hardness results in [14, 19], and Jack Snoeyink for pointing out [9].

REFERENCES

[1] P. K. Agarwal and J. Matoušek, *Ray shooting and parametric search*, SIAM J. Comput., 22 (1993), pp. 794–806.

[2] N. M. Amato and E. A. Ramos, *On computing Voronoi diagrams by divide-prune-and-conquer*, in Proceedings of the 12th ACM Sympos. Comput. Geom., Philadelphia, PA, 1996, pp. 166–175.

[3] N. Amenta and G. Ziegler, *Deformed products and maximal shadows of polytopes*, Report 502-1996, Technische Univesität Berlin, May 1996. In Advances in Discrete and Computational Geometry, B. Chazelle, J. E. Gooman, and R. Pollack, eds., American Mathematical Society, Providence, RI, 1999, pp. 57–90 (full version of [4]).

[4] N. Amenta and G. Ziegler, *Shadows and slices of polytopes*, in Proceedings of the 12th ACM Sympos. Comput. Geom., Philadelphia, PA, 1996, pp. 10–19.

[5] D. Avis, D. Bremner, and R. Seidel, *How good are convex hull algorithms?*, Comput. Geom. Theory Appl., 7 (1997), pp. 265–302.

[6] D. Avis and K. Fukuda, *A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra*, Discrete Comput. Geom., 8 (1992), pp. 295–313.

[7] M. Ben-Or, *Lower bounds for algebraic computation trees*, in Proceedings of the 15th ACM Sympos. Theory Comput., Boston, MA, 1983, pp. 80–86.

[8] S. Bloch, J. Buss, and J. Goldsmith, *How hard are n^2-hard problems?*, SIGACT News, 25 (1994), pp. 83–85.

[9] S. A. Burr, B. Grünbaum, and N. J. A. Sloane, *The orchard problem*, Geom. Dedicata, 2 (1974), pp. 397–424.

[10] T. M. Chan, *Optimal output-sensitive convex hull algorithms in two and three dimensions*, Discrete Comput. Geom., 16 (1996), pp. 361–368.

[11] T. M. Chan, *Output-sensitive results on convex hulls, extreme points, and related problems*, Discrete Comput. Geom., 16 (1996), pp. 369–387.
[12] T. M. Chan, J. Snoeyink, and C.-K. Yap, *Primal dividing and dual pruning: Output-sensitive construction of 4-d polytopes and 3-d Voronoi diagrams*, Discrete Comput. Geom., 18 (1997), pp. 433–454.
[13] D. R. Chand and S. S. Kapur, *An algorithm for convex polytopes*, J. Assoc. Comput. Mach., 17 (1970), pp. 78–86.
[14] R. Chandrasekaran, S. N. Kabadi, and K. G. Murty, *Some NP-complete problems in linear programming*, Oper. Res. Lett., 1 (1982), pp. 101–104.
[15] B. Chazelle, *An optimal convex hull algorithm in any fixed dimension*, Discrete Comput. Geom., 10 (1993), pp. 377–409.
[16] B. Chazelle and J. Matoušek, *Derandomizing an output-sensitive convex hull algorithm in three dimensions*, Comput. Geom., 5 (1995), pp. 27–32.
[17] K. L. Clarkson, *More output-sensitive geometric algorithms*, in Proceedings of the 35th IEEE Sympos. Found. Comput. Sci., Santa Fe, NM, 1994, pp. 695–702.
[18] K. L. Clarkson and P. W. Shor, *Applications of random sampling in computational geometry II*, Discrete Comput. Geom., 4 (1989), pp. 387–421.
[19] M. E. Dyer, *The complexity of vertex enumeration methods*, Math. Oper. Res., 8 (1983), pp. 381–402.
[20] P. Erdős and G. Purdy, *Two combinatorial problems in the plane*, Discrete Comput. Geom., 13 (1995), pp. 441–443.
[21] J. Erickson, *Lower bounds for linear satisfiability problems*, in Proceedings of the 6th ACM-SIAM Sympos. Discrete Algorithms, San Francisco, CA, 1995, pp. 388–395. Chicago J. Theoret. Comput. Sci., to appear.
[22] J. Erickson, *New lower bounds for convex hull problems in odd dimensions*, in Proceedings of the 12th ACM Sympos. Comput. Geom., Philadelphia, PA, 1996, pp. 1–9.
[23] J. Erickson and R. Seidel, *Better lower bounds on detecting affine and spherical degeneracies*, Discrete Comput. Geom., 13 (1995), pp. 41–57.
[24] J. Erickson and R. Seidel, *Erratum to "Better lower bounds on detecting affine and spherical degeneracies,"* Discrete Comput. Geom., 18 (1997), pp. 239–240.
[25] Z. Füredi and I. Palásti, *Arrangements of lines with a large number of triangles*, Proceedings of the Amer. Math. Soc., 92 (1984), pp. 561–566.
[26] A. Gajentaan and M. H. Overmars, *On a class of $O(n^2)$ problems in computational geometry*, Comput. Geom. Theory Appl., 5 (1995), pp. 165–185.
[27] D. Gale, *Neighborly and cyclic polytopes*, in Convexity, V. Klee, ed., in Proceedings of the Sympos. Pure Math. VII, Amer. Math. Soc., Providence, RI, 1963, pp. 225–232.
[28] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.
[29] R. L. Graham, *An efficient algorithm for determining the convex hull of a finite planar set*, Inform. Process. Lett., 1 (1972), pp. 132–133.
[30] B. Grünbaum, *Convex Polytopes*, John Wiley, New York, 1967. Rev. ed., V. Klee and P. Kleinschmidt, ed., Graduate Texts in Math., Springer-Verlag, New York, in preparation.
[31] B. Grünbaum, *Arrangements and Spreads*, Regional Conf. Ser. Math. 10, Amer. Math. Soc., Providence, 1972.
[32] C. G. J. Jacobi, *De functionibus alternantibus earumque divisione per productum e differentiis elementorum conflatum*, J. Reine Angew. Mathematik, 22 (1841), pp. 360–371. Reprinted in *Gesammelten Werke* III, G. Reimer, Berlin, 1884.
[33] L. Khachiyan, *On the complexity of approximating determinants in matrices*, J. Complexity, 11 (1995), pp. 138–153.
[34] D. G. Kirkpatrick and R. Seidel, *The ultimate planar convex hull algorithm?*, SIAM J. Comput. 15 (1986), pp. 287–299.
[35] J. Matoušek, *Linear optimization queries*, J. Algorithms, 14 (1993), pp. 432–448.
[36] J. Matoušek and O. Schwarzkopf, *On ray shooting in convex polytopes*, Discrete Comput. Geom., 10 (1993), pp. 215–232.
[37] N. Megiddo, *Linear programming in linear time when the dimension is fixed*, J. Assoc. Comput. Mach., 31 (1984), pp. 114–127.
[38] F. P. Preparata and S. J. Hong, *Convex hulls of finite sets of points in two and three dimensions*, Commun. ACM, 20 (1977), pp. 87–93.
[39] G. Rote, *Degenerate convex hulls in high dimensions without extra storage*, in Proceedings of the 8th ACM Sympos. Comput. Geom., Berlin, Germany, 1992, pp. 26–32.
[40] I. Schur, *Über eine Klasse von Matrizen, die sich einer gegebenen Matrix zuordnen lassen*, thesis, Berlin, 1901. Reprinted in *Gesammelte Abhandlungen*, Springer, 1973.

[41] R. Seidel, *A convex hull algorithm optimal for point sets in even dimensions*, M.S. thesis, Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, 1981.
[42] R. Seidel, *A method for proving lower bounds for certain geometric problems*, in Computational Geometry, G. T. Toussaint, ed., North-Holland, Amsterdam, 1985, pp. 319–334.
[43] R. Seidel, *Constructing higher-dimensional convex hulls at logarithmic cost per face*, in Proceedings of the 18th ACM Sympos. Theory Comput., Berkeley, CA, 1986, pp. 404–413.
[44] R. Seidel, *Small-dimensional linear programming and convex hulls made easy*, Discrete Comput. Geom., 6 (1991), pp. 423–434.
[45] R. Seidel, *Convex hull computations*, in Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke, eds., CRC Press LLC, Boca Raton, FL, 1997, pp. 361–376.
[46] J. Stolfi, *Oriented Projective Geometry: A Framework for Geometric Computations*, Academic Press, New York, 1991.
[47] G. F. Swart, *Finding the convex hull facet by facet*, J. Algorithms, 6 (1985), pp. 17–48.
[48] A. C. Yao, *A lower bound to finding convex hulls*, J. Assoc. Comput. Mach., 28 (1981), pp. 780–787.
[49] G. M. Ziegler, *Lectures on Polytopes*, Graduate Texts in Math. 152, Springer-Verlag, New York, 1994.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1215–1224

THE HEIGHT AND SIZE OF RANDOM HASH TREES AND RANDOM PEBBLED HASH TREES*

LUC DEVROYE†

This paper is dedicated to the memory of Markku Tamminen, who died tragically in New York shortly after he finished his analysis of N-trees and random hash trees.

Abstract. The random hash tree and the N-tree were introduced by Ehrlich in 1981. In the random hash tree, n data points are hashed to values $X_1, \ldots, X_n$, independently and identically distributed random variables taking values that are uniformly distributed on $[0,1]$. Place the X_i's in n equal-sized buckets as in hashing with chaining. For each bucket with at least two points, repeat the same process, keeping the branch factor always equal to the number of bucketed points. If H_n is the height of tree obtained in this manner, we show that $H_n/\log_2 n \to 1$ in probability. In the random pebbled hash tree, we remove one point randomly and place it in the present node (as with the digital search tree modification of a trie) and perform the bucketing step as above on the remaining points (if any). With this simple modification,

$$\frac{H_n}{\sqrt{\frac{2\log n}{\log\log n}}} \to 1$$

in probability. We also show that the expected number of nodes in the random hash tree and random pebbled hash tree is asymptotic to $2.3020238\ldots n$ and $1.4183342\ldots n$, respectively.

Key words. data structures, probabilistic analysis, hashing with chaining, hash tables, N-trees, random hash trees, expected complexity

AMS subject classifications. 68Q25, 68P10

PII. S0097539797326174

1. Introduction.

In this paper, we analyze the height of the random hash tree (Ehrlich, 1981) defined as follows. We are given $X_1, \ldots, X_n$, n independent uniform $[0,1]$ random numbers. If $n > 1$, we partition $[0,1]$ into n equal intervals of length $1/n$ each, and place all points in the intervals. Let $N_1, \ldots, N_n$ be the cardinalities of the intervals (thus, $\sum_i N_i = n$). Repeat the partition process for every interval containing at least two points and keep going until no further divisions are possible. The root node represents $[0,1]$, and each node represents a given interval. All internal nodes have at least two of the X_i's, while all leaf nodes have one or zero of the X_i's. If this structure is to be used for storing the data, then two important quantities are the number of nodes in the tree, S_n, and the height of the tree, H_n. The former quantity obviously measures the storage, while the latter is the worst-case search time. In addition, S_n is also proportional to the time needed to construct the hash tree. The model is appropriate for situations in which a hash function can be constructed that delivers a uniform $[0,1]$ random variate. This may be a debatable hypothesis.

The distributive partitioning method invented by Dobosiewicz (1978) led Ehrlich (1981) to define the N-tree. In N-trees, the X_i's are unrestricted on the real line, and given that a node has $k \geq 2$ of the X_i's, it spawns k equal child intervals of $[\min_i X_i, \max_i X_i]$. Note that there are always at least two nonempty subintervals (the first and the last) so that the size of the N-tree is limited to $n(n+1)/2$. A basic

*Received by the editors August 18, 1997; accepted for publication (in revised form) March 17, 1998; published electronically March 22, 1999. This research was sponsored by NSERC grant A3456 and FCAR grant 90-ER-0291.

http://www.siam.org/journals/sicomp/28-4/32617.html

†School of Computer Science, McGill University, Montreal H3A 2K6, Canada (luc@cs.mcgill.ca).

study of N-trees and random hash trees was performed by Tamminen (1983). The results of Tamminen will be summarized in the next section. However, Tamminen did not study H_n. Hashing with several levels of buckets has been known since being introduced by Fagin et al. (1979) as extendible hashing. Its analysis was subsequently refined in several papers, including papers by Tamminen (1983, 1985) and Flajolet (1983). However, these structures differ fundamentally from the trees studied here.

The random hash tree and its modifications studied here are vaguely related to random tries (see Pittel (1985) for the main properties). We will show that the height H_n of the random hash tree is with high probability close to $\log_2 n$, which is rather disappointing. The reason for this phenomenon is the same reason why random binary tries have height close to $2\log_2 n$. This prompted us to consider a modification similar to the modification of a trie into the digital search tree of Coffman and Eve (1970): if n points belong to an interval associated with a node, remove one point uniformly at random and place it in the node ("pebble" the node). If $n > 1$, the interval is partitioned equally into $n-1$ child subintervals and the $n-1$ remaining points are placed in their subintervals. This process is repeated until all leaf nodes have cardinality zero or one. The tree thus obtained is called the random pebbled hash tree. We will show that this minor modification causes a major improvement in H_n, which is with high probability close to $\sqrt{2\log n/\log\log n}$.

Assuming that each bucketing operation is available at unit cost, the expected time for unsuccessful and for successful search (assuming all points are equally likely to be probed for) is $O(1)$ for both random hash tree and random pebbled hash tree, but the expected worst-case search time for the latter tree is much better than for classical hash structures in view of the behavior of H_n. For example, for standard hashing with chaining under the above model, $M_n \sim \log n/\log\log n$ in probability (Gonnet (1981); see Devroye (1985, 1986) for distributions with a bounded density), where M_n is the worst-case search time (or, equivalently, the maximal size of any chain).

The random pebbled hash tree is also superior to random binary search trees, where the height is in probability of the order of $\log n$ (under a simpler computational model, however). It also compares favorably with fusion trees (Fredman and Willard, (1990)) for standard dictionary operations. Unfortunately, hash trees are not appropriate without modifications for fully dynamic situations. A brief section is devoted to this issue.

There are hash structures with better expected worst-case search and insert times. Azar et al. (1994) have shown that the worst chain length in multihashing with $d > 1$ hash functions and insertion into the shortest chain leads to a maximum occupancy of about $\log_d \log n$ with high probability. This leads to expected worst-case search times about $d\log_d \log n$, which are much better than with random pebbled hash trees. However, multihashing is not an option when the table is to be used for sorting and order-preserving hash functions are called for. Also, the performance of multihashing is easily matched by bucketing followed by a binary search tree, known as the BSST structure, discussed below.

For a survey of known results on hashing and tries, we refer to Gonnet and Baeza-Yates (1991) and Vitter and Flajolet (1990). Throughout the paper, $B(n,p)$ denotes a binomial (n,p) random variable.

2. Survey of known results on N-trees and random hash trees. Assume that the X_i's are uniformly distributed on $[0,1]$. Then Tamminen (1983) shows the following for N-trees:

A. $\sup_n \frac{\mathbf{E}S_n}{n} \le 2$.

B. $1.64 \le \liminf_n \frac{\mathbf{E}S_n}{n} \le \limsup_n \frac{\mathbf{E}S_n}{n} \le 1.70$.

C. Let A_i be the depth of X_i in the N-tree so that $(1/n)\sum_{i=1}^n A_i$ is the average successful search time. Then

$$\mathbf{E}\left\{\frac{1}{n}\sum_{i=1}^n A_i\right\} \le 2$$

for all n.

D. $1.71 \le \liminf_n \mathbf{E}\left\{\frac{1}{n}\sum_{i=1}^n A_i\right\} \le \limsup_n \mathbf{E}\left\{\frac{1}{n}\sum_{i=1}^n A_i\right\} \le 1.80$.

If the X_i's have a density f bounded by $\|f\|_\infty$, then for the random hash tree, Tamminen (1983) obtained the following results:

A. $\sup_n \frac{\mathbf{E}S_n}{n} \le 3\|f\|_\infty$.

B. $\limsup_n \frac{\mathbf{E}S_n}{n} \le 4$.

C. Let A_i be the depth of X_i in the random hash tree so that $(1/n)\sum_{i=1}^n A_i$ is the average successful search time. Then

$$\limsup_n \mathbf{E}\left\{\frac{1}{n}\sum_{i=1}^n A_i\right\} \le 4.$$

Note in particular that the asymptotic bounds in parts B and C do not depend upon the density. Tamminen (1983) also offers heuristic arguments for densities with unbounded support and so-called hybrid trees, where the first level is split as in an N-tree and all other splits are as in a random hash tree.

3. The height of the random hash tree. In this section, we assume that $X_1, \ldots, X_n$ are independently and indentically distributed (i.i.d.) and have common density f on $[0,1]$. Let H_n be the height of the random hash tree. We show the following.

THEOREM 1. *For any increasing sequence a_n (however fast), there exists a density f for which $\mathbf{P}\{H_n \ge a_n\} \to 1$ as $n \to \infty$.*

Proof. Let F be the distribution function for f, supported on $[0,1]$. Let N_1 be the number of points in $[0, 1/n]$, N_2 the number of points in $[0, 1/n^2]$, and so forth. Clearly,

$$[N_{a_n} \ge 2] \subseteq [H_n \ge a_n].$$

But, putting $p = 1 - F(1/n^{a_n})$, we see that

$$\mathbf{P}\{N_{a_n} < 2\} = p^n + np^{n-1} \le (n+1)e^{-(n-1)F(1/n^{a_n})} \to 0$$

if $nF(1/n^{a_n})/\log n \to \infty$. It suffices to pick F such that $F(1/n^{a_n}) = 1/\sqrt{n}$ for all n. Then let f be a histogram with breakpoints at $1/n^{a_n}$. Conclude that $\mathbf{P}\{H_n \ge a_n\} \to 0$. □

THEOREM 2. *When f is the uniform distribution on $[0,1]$, we have*

$$\frac{H_n}{\log_2 n} \to 1 \text{ in probability.}$$

Proof. For a lower bound, it suffices to consider a subtree of the random pebbled hash tree. To do so, we consider only those nodes at depth one that contain precisely two points. Let L be the number of these nodes. Observe that

$$\mathbf{E}\{L\} = (n-1)\mathbf{P}\{B(n-1, 1/(n-1)) = 2\} = (n-1)\binom{n-1}{2}\left(\frac{1}{(n-1)^2}\right)\left(\frac{n-2}{n-1}\right)^{n-2} \sim \frac{n}{2e}$$

as $n \to \infty$. Furthermore, if one of the data points is changed, L changes by at most two. Thus, by McDiarmid's version of Azuma's inequality (McDiarmid, 1989),

$$\mathbf{P}\{L < \mathbf{E}\{L\}/2\} \le e^{-\frac{\mathbf{E}^2\{L\}}{8n}} = e^{-\frac{n}{32e^2+o(1)}}.$$

Each of the subtrees rooted at these nodes is independent of the others. Both points in one of these nodes are placed in the same subtree with probability 1/2. Thus, a subtree has height of at least $k-1$ with probability $1/2^{k-1}$. Therefore,

$$\begin{aligned}
\mathbf{P}\{H_n \le k\} &\le \mathbf{E}\left\{(1-1/2^{k-1})^L\right\} \\
&\le \mathbf{E}\left\{e^{-\frac{L}{2^{k-1}}}\right\} \\
&\le e^{-\frac{\mathbf{E}\{L\}}{2^k}} + \mathbf{P}\{L < \mathbf{E}\{L\}/2\} \\
&\le e^{-\frac{n}{(2e+o(1))2^k}} + e^{-\frac{n}{32e^2+o(1)}} \\
&\to 0,
\end{aligned}$$

as $n \to \infty$ if $k = \lfloor (1-\epsilon)\log_2 n \rfloor$ for $\epsilon \in (0,1)$.

For the upper bound, let $N_1, \ldots, N_n$ be the cardinalities of the children of the root (so that $\sum_i N_i = n$ unless $n = 1$). If X_i and X_j find themselves in the same child node of the root, then they will stay together at depth 2 with probability 1/2, at depth 3 with probability $1/2^2$, and at depth k with probability $1/2^{k-1}$. Let $A_{m,k}$ be the event that for the mth child of the root, one of the pairs of points in the node stays together to depth k. Clearly,

$$\begin{aligned}
\mathbf{P}\{H_n > k\} &\le \mathbf{P}\{\cup_{m=1}^n A_{m,k}\} \\
&\le n\mathbf{P}\{A_{1,k}\} \\
&\le n\mathbf{E}\left\{\frac{\binom{N_1}{2}}{2^{k-1}}\right\} \\
&= \frac{n-1}{2^k}
\end{aligned}$$

as N_1 is binomial $(n, 1/n)$. Therefore, for $\epsilon > 0$,

$$\lim_{n\to\infty} \mathbf{P}\{H_n > (1+\epsilon)\log_2 n\} = 0. \qquad \square$$

4. The height of the random pebbled hash tree. In this section, we assume that $X_1, \ldots, X_n$ are i.i.d. and have common density f on $[0,1]$. Let H_n be the height of the random pebbled hash tree. Clearly, $H_n \le n-1$. In a random pebbled hash tree we interchangeably speak of nodes, intervals (each node represents an interval), and cardinality (the number of X_i's that fall in a node's interval). We show the following.

THEOREM 3. *Consider a random pebbled hash tree. For any monotonically decreasing sequence $a_n \downarrow 0$ (however slow), there exists a density f for which $\mathbf{P}\{H_n \ge na_n\} \to 1$ as $n \to \infty$.*

Theorem 3 shows that no good universal results are possible for H_n unless the density of the X_i's is suitably restricted. As we may often assume that the hash function is very good, we will assume that the X_i's are uniformly distributed on $[0,1]$. It is worthwhile to note that Theorem 3 remains valid for the N-tree as well.

Proof. We take a density f that decreases monotonically on $[0,1]$ and has distribution function F. Remove one data point. Let N_1 be the number of points in $[0, 1/n]$.

Remove one point again, and let N_2 be the number of points in $[0, 1/n^2]$, and so forth. Assume without loss of generality that na_n is integer-valued and strictly increasing to ∞. Clearly,

$$[N_{na_n} \geq 2] \subseteq [H_n \geq na_n].$$

But N_k is binomial $(N_{k-1} - 1, F(1/n^k)/F(1/n^{k-1}))$, which is stochastically greater than a binomial $(N_{k-1}, F(1/n^k)/F(1/n^{k-1}))$ random variable minus one. Therefore, N_k is stochastically greater than a binomial $(n, F(1/n^k))$ random variable minus k. Thus, for n large enough, setting $p = F(1/n^{na_n}) = \sqrt{a_n + 1/n}$, we have, by Chebyshev's inequality,

$$\begin{aligned}
\mathbf{P}\{N_{na_n} < 2\} &\leq \mathbf{P}\{B(n,p) \leq na_n + 1\} \\
&= \mathbf{P}\{B(n,p) \leq np^2\} \\
&\leq \frac{np(1-p)}{(np(1-p))^2} \\
&\sim \frac{1}{n\sqrt{a_n + 1/n}} \\
&\to 0.
\end{aligned}$$

Now, take for f a histogram whose distribution function satisfies $F(1/n^{na_n}) = \sqrt{a_n + 1/n}$ for all n large enough. □

THEOREM 4. *When f is the uniform distribution on $[0,1]$, we have*

$$\frac{H_n}{\sqrt{\frac{2\log n}{\log\log n}}} \to 1 \text{ in probability.}$$

5. Proof of Theorem 4.

LEMMA 1. *For $t > 0$ and $t \geq c$,*

$$\mathbf{P}\{B(n, c/n) \geq t\} \leq e^{t - c - t\log t + t\log c}.$$

Proof. We write $B = B(n, c/n)$. By Chernoff's bounding method, for $\lambda > 0$,

$$\begin{aligned}
\mathbf{P}\{B \geq t\} &\leq \mathbf{E}\left\{e^{\lambda B - \lambda t}\right\} \\
&= \left(1 + \left(e^\lambda - 1\right)\frac{c}{n}\right)^n e^{-\lambda t} \\
&\leq e^{c\left(e^\lambda - 1\right) - \lambda t}.
\end{aligned}$$

The upper bound is minimized when $e^\lambda = t/c$, yielding the desired inequality. □

LEMMA 2. *If B is a binomial $(n, c/n)$ random variable and $t > 0$, then, for $t \geq c$,*

$$\mathbf{E}\left\{BI_{B \geq t}\right\} \leq ce^{t - c - t\log t + t\log c}.$$

Proof. By simple bounding, we have for $\lambda > 0$,

$$\begin{aligned}
\mathbf{E}\left\{BI_{B \geq t}\right\} &\leq \mathbf{E}\left\{Be^{\lambda B - \lambda t}\right\} \\
&= n\mathbf{E}\left\{\frac{c}{n}e^{\lambda B' - \lambda t}\right\} \\
&= c\mathbf{E}\left\{e^{\lambda B' - \lambda t}\right\},
\end{aligned}$$

where B' is binomial $(n-1, c/n)$. Here we made use of the linearity of expectation. By the Chernoff bound used in Lemma 1, we have

$$\begin{aligned}\mathbf{E}\{BI_{B\geq t}\} &\leq c\left(1+\left(e^{\lambda}-1\right)\frac{c}{n}\right)^{n-1}e^{-\lambda t}\\ &\leq c\left(1+\left(e^{\lambda}-1\right)\frac{c}{n-1}\right)^{n-1}e^{-\lambda t}\\ &\leq ce^{c\left(e^{\lambda}-1\right)-\lambda t}.\end{aligned}$$

The upper bound is minimized when $e^{\lambda}=t/c$. □

We are now ready to prove Theorem 4. For the upper bound, take $\epsilon>0$ and define $k=\left\lceil(1+\epsilon)\sqrt{\frac{2\log n}{\log\log n}}\right\rceil$. The tree is pruned by omitting the root node if its cardinality is less than k, all nodes at depth one with cardinality less than $k-1$, and in general all nodes at depth d of cardinality less than $k-d$. We call this tree the pruned tree. To compute $\mathbf{P}\{H_n\geq k\}$, the pruned tree and the random pebbled hash tree are equivalent as all deleted nodes are roots of subtrees that cannot reach past depth k. The expected number of nodes in the pruned tree at depth one is

$$\begin{aligned}(n-1)\mathbf{P}\{B(n-1,1/(n-1))\geq k-1\} &\leq (n-1)e^{k-1-1-(k-1)\log(k-1)}\\ &= \frac{n-1}{e}\left(\frac{e}{k-1}\right)^{k-1},\end{aligned}$$

where we used Lemma 1. Let the L nodes at depth one have cardinalities $N_1,\ldots,N_L$. Given N_1, the first node spawns an expected number of nodes at depth two equal to

$$\begin{aligned}(N_1-1)\mathbf{P}\{B(N_1-1,1/(N_1-1))\geq k-2\} &\leq (N_1-1)e^{k-2-1-(k-2)\log(k-2)}\\ &= \frac{N_1-1}{e}\left(\frac{e}{k-2}\right)^{k-2}.\end{aligned}$$

Given all the cardinalities, we thus have an expected number of nodes at depth two not exceeding

$$\frac{\sum_{i=1}^{L}(N_i-1)}{e}\left(\frac{e}{k-2}\right)^{k-2}.$$

But

$$\mathbf{E}\left\{\sum_{i=1}^{L}N_i\right\}=\mathbf{E}\left\{\sum_{i=1}^{n-1}M_iI_{M_i\geq k-1}\right\}\leq ne^{k-1-1-(k-1)\log(k-1)},$$

where $M_1,\ldots,M_{n-1}$ are the cardinalities of the $n-1$ intervals in the first partition. Here we used Lemma 2 with $c=1$. Therefore, the expected number of nodes at depth two does not exceed

$$\frac{n}{e^2}\left(\frac{e}{k-1}\right)^{k-1}\left(\frac{e}{k-2}\right)^{k-2}.$$

By induction, the expected number of nodes at depth $k-1$ does not exceed

$$\frac{n}{e^{k-1}}\prod_{i=1}^{k-1}\left(\frac{e}{i}\right)^{i}=ne^{(k-1)(k-2)/2-\sum_{i=1}^{k-1}i\log i}=ne^{-(1/2+o(1))k^2\log k}.$$

Thus,

$$\mathbf{P}\{H_n \geq k\} \leq ne^{-(1/2+o(1))k^2 \log k} \to 0$$

for the given choice of k.

For a matching lower bound, we argue by embedding and prune the tree even further. For $\epsilon \in (0,1)$, define $k = \left\lfloor (1-\epsilon)\sqrt{\frac{2\log n}{\log\log n}} \right\rfloor$. Of all nodes at depth one, we keep only those of exact cardinality k. These nodes spawn children at depth two, of which we keep only the first child and only if it is of cardinality precisely $k-1$. Continuing in this manner, the process either becomes extinct or it survives up to depth k with at least one node of cardinality one. In the latter case, $H_n \geq k$. A node at depth one has progeny that survives to depth k with probability

$$\frac{1}{(k-1)^{k-1}(k-2)^{k-2}\cdots 2^2 1^1} \stackrel{\text{def}}{=} q.$$

Clearly, $q = e^{-(1/2+o(1))k^2 \log k}$. Given that there are L nodes at depth one in the pebbled hash tree, we have

$$\mathbf{P}\{H_n < k | L\} \leq (1-q)^L.$$

Now, L is binomial $(n-1,p)$, where $p = \mathbf{P}\{B = k\}$ and B is binomial $(n-1, 1/(n-1))$. It is easy to verify that for $n \geq 3$,

$$p = \binom{n-1}{k}\frac{1}{(n-1)^k}\left(1-\frac{1}{(n-1)}\right)^{n-1-k} \geq \frac{1}{k!}\left(\frac{n-1-k}{n-1}\right)^k\left(1-\frac{1}{n-1}\right)^{n-1}$$
$$\geq \frac{1}{4\ k!}\left(1-\frac{k}{n-1}\right)^k \geq \frac{1-k^2/(n-1)}{4\ k!} \stackrel{\text{def}}{=} q'.$$

Hence L is stochastically greater than B', a binomial $(n-1,q')$ random variable. Thus,

$$\mathbf{P}\{H_n < k\} \leq \mathbf{E}\left\{(1-q)^L\right\} \leq \mathbf{E}\left\{(1-q)^{B'}\right\}$$
$$= (q'(1-q)+1-q')^{n-1} = (1-qq')^{n-1} \leq e^{-(n-1)qq'} \to 0$$

when $nqq' \to \infty$. This is easily verified for our choice of k. □

6. The size of random hash trees. The second parameter of primary interest is S_n. We will look only at $s_n = \mathbf{E}S_n$. By linearity of expectation, we have

$$s_n = \begin{cases} 1, & 0 \leq n \leq 1, \\ 1 + n\sum_{i=0}^n \mathbf{P}\{B(n,1/n) = i\}s_i, & n \geq 2. \end{cases}$$

Note that we provide storage for empty bins as well. The recurrence given above yields $s_0 = s_1 = 1$, $s_2 = 1+2(3/4+s_2/4)$ so that $s_2 = 5$. Hence we have the following theorem.

THEOREM 5. *For the random hash tree,*

$$\lim_{n\to\infty}\frac{\mathbf{E}S_n}{n} = 2.3020238\ldots.$$

The limiting constant is

$$\frac{1}{e}\sum_{i=0}^{\infty}\frac{s_i}{i!}\ ,$$

where $s_0, s_1, \ldots$ is given by the above recurrence.

Proof. The values s_n can be shown to be approximable by the values t_n, where

$$t_n = \frac{n}{e}\sum_{i=0}^{\infty}\frac{s_i}{i!}.$$

Indeed, note that

$$\frac{s_n - t_n}{n} = \frac{1}{n} + \sum_{i=0}^{n}\left(\mathbf{P}\{B(n,1/n)=i\} - \frac{1}{e\ i!}\right)s_i - \sum_{i>n}\frac{s_i}{e\ i!}.$$

But for $0 \le i \le n$,

$$\begin{aligned}\mathbf{P}\{B(n,1/n)=i\} - \frac{1}{e\ i!} &= \binom{n}{i}\frac{(n-1)^{n-i}}{n^n} - \frac{1}{e\ i!} \le \frac{n^i}{i!}\frac{(n-1)^{n-i}}{n^n} - \frac{1}{e\ i!}\\ &= \frac{1}{i!}\frac{(n-1)^{n-i}}{n^{n-i}} - \frac{1}{e\ i!} \le \frac{e^{\frac{i}{n}-1}}{i!} - \frac{1}{e\ i!} \le \frac{i}{n\ i!},\end{aligned}$$

and for $0 \le i \le n$,

$$\begin{aligned}\mathbf{P}\{B(n,1/n)=i\} - \frac{1}{e\ i!} &\ge \frac{(n-i+1)^i}{i!}\frac{(n-1)^{n-i}}{n^n} - \frac{1}{e\ i!}\\ &\ge \left(1-\frac{i-2}{n-1}\right)^i \frac{e^{-\frac{1}{1-1/n}}}{i!} - \frac{1}{e\ i!}\\ &\ge \frac{e^{-\frac{i^2}{n-i+1}-\frac{1}{1-1/n}}}{i!} - \frac{1}{e\ i!} \ge \frac{1}{e\ i!}\left(e^{-\frac{i^2}{n-i+1}-\frac{1}{n-1}}-1\right)\\ &\ge -\frac{1}{e\ i!}\left(\frac{i^2}{n-i+1}+\frac{1}{n-1}\right)\\ &\ge -\frac{1}{e\ i!}\left(\frac{i^2}{n/2}+i^2 I_{i\ge n/2}+\frac{1}{n-1}\right).\end{aligned}$$

Thus, we have $(s_n - t_n)/n \to 0$ if

$$\sum_{i=0}^{\infty}\frac{i^2 s_i}{i!} < \infty.$$

But that follows from the simple fact that $s_i = O(i)$, something that is easy to verify. Numerical computations show that

$$\frac{1}{e}\sum_{i=0}^{\infty}\frac{s_i}{i!} = 2.3020238\ldots. \qquad \square$$

For the random pebbled hash tree, the recurrences are slightly different. Indeed,

$$s_n = \begin{cases} 1, & 0 \le n \le 1,\\ 1 + (n-1)\sum_{i=0}^{n-1}\mathbf{P}\{B(n-1,1/(n-1))=i\}s_i, & n > 1.\end{cases}$$

The analysis is entirely similar as for Theorem 5, and we thus obtain the following theorem.

THEOREM 6. *For the random pebbled hash tree,*

$$\lim_{n\to\infty} \frac{\mathbf{E}S_n}{n} = 1.4183342\ldots.$$

The limiting constant is

$$\frac{1}{e}\sum_{i=0}^{\infty} \frac{s_i}{i!} ,$$

where $s_0, s_1, \ldots$ is given by the above recurrence.

In particular, note that random pebbled hash trees are smaller on the average than random N-trees.

7. BBST: Bucketing followed by binary search trees. Assume that we bucket n points into n equispaced buckets and that within each bucket we maintain a balanced binary search tree. Then, in view of the results of Gonnet (1981) and Devroye (1985), the expected worst-case search and insert times and indeed the expected value of the height of this hybrid structure is asymptotic to $\log_2 \log n$ for all bounded densities f on $[0,1]$. In fact, the height H_n satisfies $H_n/\log_2 \log n \to 1$ in probability. The expected average search time (if each element is equally likely to be probed for) and the expected unsuccessful search time (for a random element drawn independently from the same density f) are both $O(1)$.

8. Extension: m pebbles. We may extend the analysis to m pebbles, leaving m randomly selected points in every node before bucketing. If there are $m+k$ points, then there are k (with possibly $k=1$) child nodes, each corresponding to a subinterval $1/k$th of the length of the interval of the split node. This leads to random m-pebbled hash trees. A quick analysis not worth repeating here shows that in this case, $H_n \sim \sqrt{(2/m)\log n/\log\log n}$ in probability. However, the worst-case search time is roughly $(m+1)H_n$ when comparisons and bucket operations all take one time unit. Therefore, it seems wasteful to take $m > 1$, and thus the most important member of the family is the pebbled hash tree studied above. A rather obvious but unaesthetic modification, however, will improve matters exponentially. Let us set $m = c\log n/\log\log n$ for some constant c, and pick the m pebbles as follows for a node representing interval $[a,b]$: find the mth order statistic M in time linear in the number of points. The m points on $[a,M]$ are placed in a balanced binary search tree in time $m\log m = O(\log\log n)$. The remaining points on $(M,b]$ are bucketed as in a random hash tree, and the process is repeated. This tree has expected height $O(1)$ so that expected worst-case search times are $O(\log\log n)$. As this method is eclipsed by the simple BBST structure of the previous section, we will not analyze it in this paper.

9. Dynamic data structures. The data structures described above are of course useful in any static setting, in which case we have expected preprocessing time $O(n)$ and expected worst-case search times as given by the expected values of H_n in the analyses. Consider now the standard dictionary operations `insert` and `search`. We may introduce the load factor α, the number of elements stored divided by the branch factor of the root. The objective is to keep α at all times in a fixed range, such as $[1/2, 2]$. As soon as α reaches a boundary, a complete rehash is performed to make $\alpha = 1$. In an amortized sense, these rehash operations take $O(1)$

expected time. Expected worst-case search times remain asymptotically the same as for the static case. However, for a fully dynamic data structure with interspersed `delete` and `insert` operations, additional analysis is required.

Acknowledgment. I would like to thank all referees for their suggestions.

REFERENCES

Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal (1994), *Balanced allocations* (*extended abstract*), in Proc. 26th ACM Symposium on the Theory of Computing, pp. 593–602.

E. G. Coffman and J. Eve (1970), *File structures using hashing functions*, Communications of the ACM, 13, pp. 427–436.

L. Devroye (1985), *The expected length of the longest probe sequence when the distribution is not uniform*, J. Algorithms, 6, pp. 1–9.

L. Devroye (1986), *Lecture Notes on Bucket Algorithms*, Birkhäuser Verlag, Boston.

W. Dobosiewicz (1978), *Sorting by distributive partitioning*, Inform. Process. Lett., 7, pp. 1–6.

G. Ehrlich (1981), *Searching and sorting real numbers*, J. Algorithms, 2, pp. 1–14.

R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong (1979), *Extendible hashing—a fast access method for dynamic files*, ACM Trans. Database Systems, 4, pp. 315–344.

P. Flajolet (1983), *On the performance evaluation of extendible hashing and trie search*, Acta Inform., 20, pp. 345–369.

M. L. Fredman and D. E. Willard (1990), *Blasting through the information theoretic barrier with fusion trees*, in Proc. 22nd Symposium on Theory of Computing, ACM Press, pp. 1–7.

G. H. Gonnet (1981), *Expected length of the longest probe sequence in hash code searching*, J. Assoc. Comput. Mach., 28, pp. 289–304.

G. H. Gonnet and R. Baeza-Yates (1991), *Handbook of Algorithms and Data Structures*, Addison-Wesley, Workingham, UK.

C. McDiarmid (1989), *On the method of bounded differences*, in Surveys in Combinatorics 1989, London Math. Soc. Lecture Note Ser. 141, Cambridge University Press, Cambridge, UK.

B. Pittel (1985), *Asymptotical growth of a class of random trees*, Ann. Probab., 13, pp. 414–427.

M. Tamminen (1983), *Analysis of* N*-trees*, Inform. Process. Lett., 16, pp. 131–137.

M. Tamminen (1985), *Two levels are as good as any*, J. Algorithms, 6, pp. 138–144.

J. S. Vitter and P. Flajolet (1990), *Average-case analysis of algorithms and data structures*, in Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity, J. van Leeuwen, ed., MIT Press, Amsterdam, pp. 431–524.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1225–1231

ON REARRANGEABILITY OF MULTIRATE CLOS NETWORKS*

GUO-HUI LIN†, DING-ZHU DU‡, XIAO-DONG HU§, AND GUOLIANG XUE¶

Abstract. Chung and Ross [*SIAM J. Comput.*, 20 (1991), pp. 726–736] conjectured that the multirate three-stage Clos network $C(n, 2n-1, r)$ is rearrangeable in the general discrete bandwidth case; i.e., each connection has a weight chosen from a given finite set $\{p_1, p_2, \ldots, p_k\}$ where $1 \geq p_1 > p_2 > \cdots > p_k > 0$ and p_i is an integer multiple of p_k, denoted by $p_k \mid p_i$, for $1 \leq i \leq k-1$. In this paper, we prove that multirate three-stage Clos network $C(n, 2n-1, r)$ is rearrangeable when each connection has a weight chosen from a given finite set $\{p_1, p_2, \ldots, p_k\}$ where $1 \geq p_1 > p_2 > \cdots > p_h > 1/2 \geq p_{h+1} > \cdots > p_k > 0$ and $p_{h+2} \mid p_{h+1}$, $p_{h+3} \mid p_{h+2}$, $\ldots$, $p_k \mid p_{k-1}$. We also prove that $C(n, 2n-1, r)$ is two-rate rearrangeable and $C(n, \lceil \frac{7n}{3} \rceil, r)$ is three-rate rearrangeable.

Key words. rearrangeability, multirate Clos networks, minimization of the number of center switches

AMS subject classifications. 94A05, 05C70

PII. S0097539796313921

1. Introduction.

The multirate interconnection network is a research topic in asynchronous transfer mode (ATM) networks with applications in computer networks, telecommunications, and the Internet. The symmetric three-stage Clos network $C(n, m, r)$ has been widely used in the design of telecommunication networks [1]. $C(n, m, r)$ consists of r $n \times m$ crossbars (switches) in the first (or input) stage, m $r \times r$ crossbars in the second (or central) stage, and r $m \times n$ crossbars in the third (or output) stage. Every crossbar in the first stage has an outlet connected to an inlet of every crossbar in the second stage, and every crossbar in the second stage has an outlet connected to an inlet of every crossbar in the third stage (Fig. 1). There are totally rn inlets in the first stage, called *inputs*, and totally rn outlets in the third stage, called *outputs*. A *connection* (*request*, or *call*) in the network is a triple (i, j, w) where i is an input, j is an output, while w is the *weight* of this connection, and it represents the bandwidth required by the connection. A *route* is a path in the network joining an input crossbar (i.e., a crossbar in the first stage) to an output crossbar (i.e., a crossbar in the third stage) and a route r *realizes* a connection (i, j, w) if the input crossbar i and output crossbar j are connected by r with capacity w.

Usually, one assumes that each link has unit capacity. Therefore, the weight of each connection is in the interval $[0, 1]$. A set of connections is *compatible* if, at every

*Received by the editors December 19, 1996; accepted for publication (in revised form) February 5, 1998; published electronically March 22, 1999.
http://www.siam.org/journals/sicomp/28-4/31392.html

†Institute of Applied Mathematics, Chinese Academy of Sciences, Beijing 100080, People's Republic of China. Current address: Department of Computer Science, The University of Vermont, Burlington, VT 05405 (ghlin@emba.uvm.edu). The work of this author was supported in part by the National Natural Science Foundation of China.

‡Department of Computer Science, University of Minnesota, Minneapolis, MN 55455 (dzd@cs.umn.edu). The work of this author was supported in part by National Science Foundation grant CCR-9530306 and Grant-in-Aid Research of The University of Minnesota.

§Institute of Applied Mathematics, Chinese Academy of Sciences, Beijing 100080, People's Republic of China (xdhu@amath3.amt.ac.cn). The work of this author was supported in part by the National Natural Science Foundation of China.

¶Department of Computer Science and Electrical Engineering, The University of Vermont, Burlington, VT 05405 (xue@emba.uvm.edu). The research of this author was supported in part by National Science Foundation grants ASC-9409285 and OSR-9350540.

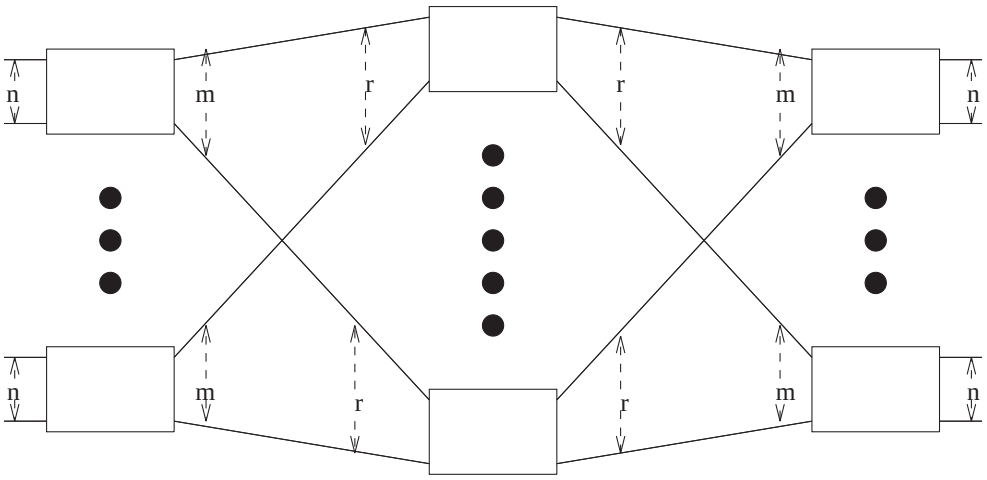


FIG. 1. *Symmetric three-stage Clos network.*

input and output, the sum of weights of all connections is at most one. A *request frame* is a compatible set of connections. A *configuration* is a set of routes, and it is *compatible* if the total weight of routes passing through each link is at most one. A request frame is said to be *realizable* if there exists a compatible configuration which contains routes realizing all connections in the request frame. A multirate network is said to be *(multirate) rearrangeable* if every request frame is realizable.

A connection c is said to be *compatible* with a request frame $\mathcal{F}$ if $\mathcal{F} \cup \{c\}$ is still compatible. A route r is said to be *compatible* with a compatible configuration $\mathcal{C}$ if $\mathcal{C} \cup \{r\}$ is still compatible. A network is said to be *strictly nonblocking* if for every compatible configuration $\mathcal{C}$ realizing a request frame $\mathcal{F}$ and every connection c compatible with $\mathcal{F}$, there exists a route r such that r realizes c and r is compatible with $\mathcal{C}$.

In circuit switching, all connections are assumed to have the same rate one. Namely, a network is said to be rearrangeable in circuit switching if every compatible request frame of connections with weight one is realizable, and it is well known that the symmetric three-stage Clos network $C(n, m, r)$ is rearrangeable in circuit switching if and only if $m \geq n$ [1]. Now, since multirate is involved, we may need more crossbars in the center stage to reach the rearrangeability. Chung and Ross [2] conjectured[1] that if a symmetric three-stage Clos network $C(n, m, r)$ is strictly nonblocking in circuit switching, then it is multirate rearrangeable in the discrete bandwidth case. That is, $C(n, 2n-1, r)$ is multirate rearrangeable if each connection has weight chosen from a given finite set $\{p_1, p_2, \ldots, p_k\}$ where $1 \geq p_1 > p_2 > \cdots > p_k > 0$ and p_i is an integer multiple of p_k, denoted by $p_k \mid p_i$, for $1 \leq i \leq k-1$. They verified their conjecture when $k = 2$ and $\{p_1, p_2, \ldots, p_k\} = \{1, p\}$.

Du et al. [3] recently proved that $C(n, m, r)$ for $m \geq 41n/16$ is multirate rearrangeable in general.

[1] After proving the result [2, Corollary 3] that a strictly nonblocking network for classical circuit switching is also rearrangeable if all connections have weight of either b or 1, Chung and Ross [2] stated that "It would be of interest to show that Corollary 3 holds for the general discrete bandwidth case with K distinct rates." For an easy reference, we call it the Chung–Ross conjecture. In this paper, we consider this conjecture only for three-stage Clos networks.

In this paper, we prove that the symmetric three-stage Clos network $C(n, 2n-1, r)$ is multirate rearrangeable when each connection has a weight chosen from a given finite set $\{p_1, p_2, \ldots, p_k\}$ where $1 \geq p_1 > p_2 > \cdots > p_h > 1/2 \geq p_{h+1} > \cdots > p_k > 0$ and $p_{h+2} \mid p_{h+1}$, $p_{h+3} \mid p_{h+2}, \ldots, p_k \mid p_{k-1}$. We also prove that $C(n, 2n-1, r)$ is two-rate rearrangeable and $C(n, \lceil \frac{7n}{3} \rceil, r)$ is three-rate rearrangeable.

2. Main results. In this section, we prove the following theorem.

THEOREM 2.1. *Symmetric three-stage Clos network $C(n, 2n-1, r)$ is multirate rearrangeable when each connection has a weight chosen from a given finite set $\{p_1, p_2, \ldots, p_k\}$ where $p_{h+2} \mid p_{h+1}$, $p_{h+3} \mid p_{h+2}, \ldots, p_k \mid p_{k-1}$, and $1 \geq p_1 > p_2 > \cdots > p_h > 1/2 \geq p_{h+1} > p_{h+2} > \cdots > p_k > 0$.*

Proof. For $k - h = 0$, since $p_\ell > 1/2$ for all $\ell = 1, 2, \ldots, k$, each link contains at most one call. Thus, we can treat the network as that used in circuit switching. From the result about circuit switching, $C(n, 2n-1, r)$ is nonblocking and hence rearrangeable. Next, we consider $k - h \geq 1$. Suppose α_ℓ for $\ell = 1, 2, \ldots, h$ and β are integers such that $p_\ell + \alpha_\ell p_k \leq 1 < p_\ell + (\alpha_\ell + 1)p_k$ and $\beta p_k \leq 1 < (\beta + 1)p_k$. First, route all calls with weights $p_1, p_2, \ldots, p_{k-1}$ on the $2n-1$ center switches. By the induction assumption, it is possible. Now we route all calls with weight p_k (or, say, p_k-call) in an arbitrary ordering. We will prove that we can always find a space for a p_k-call compatible with previous routed calls. In fact, for contradiction, suppose there exists a p_k-call (i, j, p_k) compatible with previous routed calls, but we cannot find space on a center switch to route this call (i, j, p_k). Let I be the input switch containing input i and J the output switch containing output j. Define the I-load (the J-load) of a center switch as the sum of weights of all calls from input switch I to the center switch (from the center switch to output switch J). Then every center switch has its I-load or its J-load greater than $1 - p_k$. Note that there are $2n-1$ center switches. Therefore, we can find either n center switches that each has I-load greater than $1 - p_k$ or n center switches that each has J-load greater than $1 - p_k$. Without loss of generality, we assume that the former occurs. This means that each of these n center switches has I-load equal to either $p_\ell + \alpha_\ell p_k$ for some $\ell \in \{1, 2, \ldots, h\}$ or βp_k. Note that input switch I has exactly n input, and each input can contain at most one p_ℓ-call for $\ell \in \{1, 2, \ldots, h\}$. Thus, every input in input switch I has a load equal to either $p_\ell + \alpha_\ell p_k$ or βp_k. It follows that the call (i, j, p_k) cannot exist, which is a contradiction. □

Melen and Turner [4] gave a routing algorithm CAP, and with CAP it can be showed that the multirate three-stage Clos network $C(n, 2n-1, r)$ is rearrangeable when each connection has a weight at most $1/2$. The outline of the algorithm CAP is as follows.

Divide all calls in each input/output switch into groups of size m. Then algorithm CAP can arrange m center switches, with flexible link capacity, to route all calls such that each center switch holds at most one call from every group in every input/output switch. Thus, if for each input/output switch the total weight of a set of calls chosen one from each group is never greater than one, then the line capacity can be restricted to be within one. Thus, algorithm CAP actually routes all calls with m center switch in the model considered in this paper.

The following is a corollary of Theorem 2.1.

COROLLARY 2.2. *The symmetric three-stage Clos network $C(n, 2n-1, r)$ is multirate rearrangeable when every connection has a weight chosen from a given set $\{p_1, p_2\}$.*

Proof. If $p_1 \leq 1/2$, then it follows from the result of Melen and Turner [4]. If

$p_1 > 1/2$, it follows from Theorem 2.1. □

The next result is obtained with algorithm CAP, too.

LEMMA 2.3. *Symmetric three-stage Clos network $C(n, 2n-1, r)$ is multirate rearrangeable when each connection has a weight bigger than* $1/3$.

Proof. Since every call has weight bigger than $1/3$, each input/output has at most two calls. Thus, each input/output switch has at most $2n$ calls. If it has fewer than $2n$ calls, we put them all into one group. If it has exactly $2n$ calls, then we place the one with the smallest weight among the $2n$ calls into one group and put the remaining $2n-1$ calls into another group. Note that when an input (output) switch has exactly $2n$ calls, each inlet (outlet) in this switch has exactly two calls. It follows from this fact that among weights of the $2n$ calls, the smallest one plus any other one cannot exceed one. This means that the total weight of any two calls, respectively, chosen from two groups is at most one. Therefore, algorithm CAP can route all calls with $2n-1$ center switches. □

The following is a result about three rates.

THEOREM 2.4. *Symmetric three-stage Clos network $C(n, \lceil \frac{7n}{3} \rceil, r)$ is multirate rearrangeable when every connection has a weight chosen from $\{p_1, p_2, p_3\}$ where $1 \geq p_1 > p_2 > p_3 > 0$.*

Proof. By Theorem 2.1 and Lemma 2.3, we can assume that $p_1 > 1/2 \geq p_2$ and $1/3 \geq p_3$. In the following, we consider two cases.

Case 1. $p_1 > 1/2$ and $1/3 \geq p_2 > p_3$. It was proved in [3] that if all calls with weights bigger than $1/f$ (f is an integer) can be routed with $c(\geq 2n)$ center switches, then at most $\lceil (c-2)/f - c + 2n \rceil$ additional center switches are needed to route all calls with weights at most $1/f$. Now we choose $f = 6$. If $p_3 \leq 1/6$, then all calls with weights bigger than $1/6$ are p_1- or p_2-calls. They can be routed with $2n$ center switches since all calls with the same weight can be routed with n center switches. Therefore, the total number of center switches for routing all calls is at most $2n + \lceil (2n-2)/6 \rceil < \lceil 7n/6 \rceil$. Hence, we may assume $p_3 > 1/6$.

Consider a bipartite graph G with two vertex sets, respectively, consisting of all inputs and all outputs and with all p_3-calls and p_2-calls as edges. Since $p_2 > p_3 > 1/6$, each vertex has degree at most five. By a lemma of de Werra [5], this graph can be decomposed into five edge-disjoint matchings. First, we can route calls in four matchings with $\lceil 4n/3 \rceil$ center switches since $1/3 \geq p_2 > p_3$. In fact, each matching for the bipartite graph G can be decomposed into n edge-disjoint matchings for the bipartite graph H between input switches and output switches, and hence four matchings for G give $4n$ matchings for H. Moreover, each center switch can route three matchings for H. Therefore, we need only $\lceil 4n/3 \rceil$ center switches to route four matchings for G.

Next, we consider calls in the fifth matching together with all p_1-calls. We will route them with n center switches in the following way.

If $p_1 + p_3 > 1$, then there are at most n considered calls in each input/output switch. Therefore, n center switches are enough to route them by classic routing algorithm.

If $p_1 + p_3 \leq 1$ and $p_1 + p_2 > 1$, then each input/output switch has at most $2n$ calls in which there are at most n p_1- or p_2-calls. Thus, we can divide them into two groups of at most size n such that one group contains only p_3-calls and the other one contains the remainders. Now, with the routing algorithm CAP of Melen and Turner [4], n center switches are enough to route all considered calls.

If $p_1 + p_2 \leq 1$, then each input/output switch has at most $2n$ calls in which there

are at most n p_1-calls. Thus, we can divide them into two groups of size at most n such that one group contains only p_2- or p_3-calls and the other one contains the remainders. Now, the routing algorithm CAP of Melen and Turner [4] can also use n center switches to route all considered calls.

Case 2. $p_1 > 1/2 \geq p_2 > 1/3 \geq p_3$. Furthermore, if $p_1 + p_2 \leq 1$, then each input switch has at most $2n$ p_1- or p_2-calls. These at most $2n$ calls can be divided into two groups such that each group contains at most n calls and only one group contains p_1-calls since each input switch has at most n p_1-calls. Since $p_1 + p_2 \leq 1$, the sum of two elements chosen, respectively, from the two group is at most one. Thus, we can use n center switches to route all p_1-calls and p_2-calls by the routing algorithm CAP of Melen and Turner [4]. It is shown in [3] that a network which is rearrangeable for the classical circuit switching is multirate rearrangeable if all weights are in the interval $[b, 1/\lfloor 1/b \rfloor]$ for some $0 < b \leq 1$. According to this result, n center switches are enough to route all p_3-calls. Therefore, totally, $2n$ center switches are enough when $p_1 + p_2 \leq 1$. Next, we may also assume $p_1 + p_2 > 1$. An argument similar to that in the proof of Theorem 2.1 will be employed.

First, route all calls with weights p_1 and p_2 on the $2n - 1$ center switches. By Corollary 2.2, it is possible. Now we route all calls with weight p_3 (or, say, p_3-call) in an arbitrary ordering. We will prove that we can always find a space for a p_3-call compatible with previous routed calls. In fact, for contradiction, suppose there exists a p_3-call (i, j, p_k) compatible with previous routed calls, but we cannot find space on a center switch to route this call (i, j, p_k). Let I be the input switch containing input i and J the output switch containing output j. Define the I-load (the J-load) of a center switch as the sum of weights of all calls from input switch I to the center switch (from the center switch to output switch J). Then every center switch has its I-load or its J-load greater than $1 - p_3$. Note that there are $\lceil 7n/3 \rceil$ center switches. Therefore, we can find either $\lceil 7n/6 \rceil$ center switches that each has I-load greater than $1 - p_3$ or $\lceil 7n/6 \rceil$ center switches that each has J-load greater than $1 - p_3$. Without loss of generality, we assume that the former occurs. Since $p_1 + p_2 > 1$, every I-load greater than $1 - p_3$ must be in the following forms:

$$p_1 + k_1 p_3 \left(k_1 = \left\lfloor \frac{1 - p_1}{p_3} \right\rfloor \right),$$

$$p_2 + k_2 p_3 \left(k_2 = \left\lfloor \frac{1 - p_2}{p_3} \right\rfloor \right),$$

$$2p_2 + k_3 p_3 \left(k_3 = \left\lfloor \frac{1 - 2p_2}{p_3} \right\rfloor \right),$$

$$k_4 p_3 \left(k_4 = \left\lfloor \frac{1}{p_3} \right\rfloor \right).$$

Suppose that there are x_1 center switches with I-load equal to $p_1 + k_1 p_3$, x_2 center switches with I-load equal to $p_2 + k_2 p_3$, x_3 center switches with I-load equal to $2p_2 + k_3 p_3$, and x_4 center switches with I-load equal to $k_4 p_3$. Then we have

$$x_1 + x_2 + x_3 + x_4 \geq \lceil 7n/6 \rceil.$$

Without loss of generality, assume

$$x_1 + x_2 + x_3 + x_4 = \lceil 7n/6 \rceil.$$

(If $x_1+x_2+x_3+x_4 > \lceil 7n/6 \rceil$, we delete some center switches from our consideration.) Now we consider p_1-calls and p_2-calls only in the I-loads of $\lceil 7n/6 \rceil$ center switches. Suppose that among n inputs of input switch I, there are y_1 inputs each containing such a p_1-call, y_2 ones each containing one such p_2-call, y_3 ones each containing two such p_2-calls, and y_4 containing only p_3-calls. Note that the number of considered p_1-calls and the number of considered p_2-calls does not change and the total number of p_3-calls in I-loads must be smaller than the maximum number of p_3-calls which can be put in the inputs. Thus, we have

$$\begin{aligned} x_1 &= y_1, \\ x_2 + 2x_3 &= y_2 + 2y_3, \\ k_1x_1 + k_2x_2 + k_3x_3 + k_4x_4 &< k_1y_1 + k_2y_2 + k_3y_3 + k_4y_4. \end{aligned}$$

That is,

$$\begin{aligned} (x_1 - y_1) &= 0, \\ (x_2 - y_2) + 2(x_3 - y_3) &= 0, \\ k_1(x_1 - y_1) + k_2(x_2 - y_2) + k_3(x_3 - y_3) + k_4(x_4 - y_4) &< 0. \end{aligned}$$

Therefore,

$$\begin{aligned} &(x_1 - y_1) + (x_2 - y_2) + (x_3 - y_3) + (x_4 - y_4) \\ <&\left(1 - \frac{k_1}{k_4}\right)(x_1 - y_1) + \left(1 - \frac{k_2}{k_4}\right)(x_2 - y_2) + \left(1 - \frac{k_3}{k_4}\right)(x_3 - y_3) \\ =&\frac{2k_2 - k_3 - k_4}{k_4} \cdot (x_3 - y_3). \end{aligned}$$

Note that

$$0 \le \left\lfloor \frac{2(1-p_2)}{p_3} \right\rfloor - (k_3 + k_4) \le 1,$$

$$0 \le \left\lfloor \frac{2(1-p_2)}{p_3} \right\rfloor - 2k_2 \le 1.$$

It follows that $|2k_2 - k_3 - k_4| \le 1$. Now, we consider two subcases.

Subcase 2.1. $p_3 \le 1/4$. In this case, we have $k_4 \ge 4$ and $|x_3 - y_3| = |y_2 - x_2|/2 \le \lceil 7n/6 \rceil/2$. Thus, we have

$$(x_1 - y_1) + (x_2 - y_2) + (x_3 - y_3) + (x_4 - y_4) < \lceil 7n/6 \rceil/8.$$

Therefore,

$$x_1 + x_2 + x_3 + x_4 < n + \lceil 7n/6 \rceil/8 \le \lceil 7n/6 \rceil(6/7 + 1/8) < \lceil 7n/6 \rceil,$$

which is a contradiction.

Subcase 2.2. $p_3 > 1/4$. In this case, we have $1 \le k_2 \le 2$, $0 \le k_3 \le 1$, and $k_4 = 3$. If $k_3 = 1$, then $2p_2 + p_3 \le 1$. Then we must have $k_2 = 2$. Thus, $2k_2 - k_3 - k_4 = 0$. Therefore,

$$x_1 + x_2 + x_3 + x_4 < y_1 + y_2 + y_3 + y_4 = n,$$

which is a contradiction. (Note: The proof here shows that $2n - 1$ center switches are enough in this special situation.) Next, we assume $k_3 = 0$, i.e., $2p_2 + p_3 > 1$. Consider a bipartite graph with two vertex sets, respectively, consisting of all inputs and all outputs and with all p_3-calls and p_2-calls as edges. Since $p_2 > p_3 > 1/4$, each vertex has degree at most three. By a lemma of de Werra [5], this graph can be decomposed into three edge-disjoint matchings. Clearly, we can route two matchings with n center switches since $1/2 \ge p_2 > p_3$. Put calls in the third matching together with all p_1-calls. We now consider p_1-calls and those calls in the third matching. If $p_1 + p_3 > 1$, then each input/output switch has at most n considered calls. They can be routed with n center switches. If $p_1 + p_3 \le 1$, then each input/output switch has at most $2n$ such calls in which there exist at most n p_1- or p_2-calls. Thus, we can route them with n center switches by the routing algorithm CAP of Melen and Turner [4]. Thus, totally, $2n$ center switches are enough in this case. □

Note that in Subcase 2.2, only $2n$ center switches are required. Moreover, if $1 \ge p_1 > 1/2$ and $1/3 \ge p_2 > p_3 > 1/4$, then we can route all p_1-calls with n center switches and route all p_2-calls and p_3-calls with n center switches. Therefore, we have the following corollary.

COROLLARY 2.5. *The symmetric three-stage Clos network $C(n, 2n, r)$ is multirate rearrangeable when every connection has a weight chosen from $\{p_1, p_2, p_3\}$ where $1 \ge p_1 > p_2 > p_3 > 1/4$.*

3. Discussion. The conjecture of Chung and Ross [2] on rearrangeability of multirate Clos networks seems true not only in the discrete bandwidth case but also in arbitrary rates. This is equivalent to the following conjecture: consider any double stochastic square matrix of order nr. Divide it into r^2 blocks, each of which is an $n \times n$ submatrix. Now we color all cells of the matrix such that the total value in the same color and in the same block-row is at most one and the total value in the same color and in the same block-column is at most one. The conjecture says that $2n - 1$ colors are enough. In this paper, we proved it in several special cases. Finally, we would like to mention that by an argument similar to the proof of Lemma 2.3, we can also prove that the conjecture is true for $r = 2$.

Acknowledgments. The authors wish to thank the referees for their insightful comments.

REFERENCES

[1] V. E. BENES, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, 1965.

[2] S.-P. CHUNG AND K. W. ROSS, *On nonblocking multirate interconnection networks*, SIAM J. Comput., 20 (1991), pp. 726–736.

[3] D. Z. DU, B. GAO, F. K. HWANG, AND J. H. KIM, *On multirate rearrangeable Clos networks*, SIAM J. Comput., 28 (1999), pp. 463–470.

[4] R. MELEN AND J. S. TURNER, *Nonblocking multirate networks*, SIAM J. Comput., 18 (1989), pp. 301–313.

[5] D. DE WERRA, *Balanced schedules*, Inform. J., 9 (1971), pp. 453–465.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1232–1246

DESIGN OF ON-LINE ALGORITHMS USING HITTING TIMES*

PRASAD TETALI†

Abstract. Random walks are well known for playing a crucial role in the design of randomized off-line as well as on-line algorithms. In this work we prove some basic identities for ergodic Markov chains (e.g., an interesting characterization of *reversibility* in Markov chains is obtained in terms of first passage times). Besides providing new insight into random walks on weighted graphs, we show how these identities give us a way of designing competitive randomized on-line algorithms for certain well-known problems.

Key words. random walk, graph, Markov chain, reversibility, competitive ratio, first passage time, M-matrices

AMS subject classifications. 68Q25, 60C05, 60J10, 60J15, 60J20, 15A51

PII. S0097539798335511

1. Introduction.

In a recent paper, Coppersmith et al. [8] made clever use of results from synthesis of electrical networks to design reversible random walks useful for certain randomized on-line algorithms.

At the heart of their methods lies the following problem on random walks. We are given a weighted (undirected) graph with n vertices, and with weight C_{ij} on edge $\{i, j\}$, for $1 \leq i, j \leq n$. Assume that the weights are symmetric and that they satisfy the triangle inequality. Every traversal of an edge $\{i, j\}$ costs C_{ij}.

Recall that a random walk on an undirected graph is the Markov chain whose state space is the vertex set of the graph, whose behavior is given by the rule that when the chain is at any given vertex the next transition is along an edge incident to that vertex that is chosen at random, according to some probability distribution. (The probability distribution could depend on the vertex, and the case of uniform distribution over the incident edges is often called a *simple* random walk.) We say that a random walk on G has *stretch* s (with respect to the cost matrix C) if, for any sequence of vertices $v_0, \ldots, v_k$, the expected cost of the random walk to traverse these nodes in the prescribed order is at most s times the optimal cost, up to an additive constant. Let e_{uv} denote the expected cost of the walk to go from vertex u to vertex v. If for every $v_0, \ldots, v_k$, $\sum_{i=1}^{k} e_{v_{i-1}v_i} \leq s \sum_{i=1}^{k} C_{v_{i-1}v_i} + a$, where $a \geq 0$, then the walk is said to have stretch s. Given the cost matrix C, the problem is to design a random walk with as low a stretch as possible.

Coppersmith et al. proved the following tight result for all symmetric cost matrices: Any random walk on a weighted (undirected) graph with n vertices has stretch $\geq (n-1)$, and every weighted (undirected) graph has a random walk with stretch $\leq (n-1)$.

Coppersmith et al. justified the relevance of this problem by providing bounds for the *cat and mouse* game, which they showed was central to the analysis of on-line

*Received by the editors March 10, 1998; accepted for publication March 20, 1998; published electronically March 22, 1999. A preliminary version of this paper appeared in *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, PA, 1994, pp. 402–411. The work of this author was done while at AT&T Bell Labs, Murray Hill, NJ 07974.

http://www.siam.org/journals/sicomp/28-4/33551.html

†School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332-0160 (tetali@math.gatech.edu).

algorithms for well-known problems such as the metrical task system problem and the k-server problem. (We shall define these shortly for the nonspecialist.)

Symmetry of the costs is crucial to the basic technique used in [8] in designing the appropriate random walk, and thus an interesting question is left open on stretch under asymmetric costs—equivalently, the stretch of random walks on directed graphs. In this work we prove close-to-optimal lower and upper bounds on a stretch for a class of matrices, without the symmetry assumption. Moreover, under symmetry our results imply those of [8].

In a nutshell, Coppersmith et al. interpret a given cost matrix as an effective resistance matrix of a resistive network, and then they synthesize an actual network that has the desired properties. Classical analogues between resistive networks and reversible Markov chains yield a corresponding random walk that achieves the optimal stretch. We use a different way of synthesizing a random walk, the advantage being that we do not need reversibility of the walk (i.e., symmetry of the costs) for our techniques to work. We interpret the cost matrix as the hitting time (first passage time) matrix of an ergodic (not necessarily reversible) Markov chain and then describe how to find the unique chain that yields the desired hitting times, i.e., design the transition probabilities which yield the desired hitting times. While such a synthesis of an ergodic chain from a valid hitting time matrix is unique and efficient, it is not true that an arbitrary cost matrix is always a hitting time matrix.[1] (The consolation, however, is that we can check for the latter condition with essentially one matrix inversion.) We call a cost matrix *ergodic* if it can be interpreted as a certain hitting time matrix, and we call the corresponding random walk an *ergodic* walk. We show that this walk achieves optimal stretch when the costs are symmetric and is close to optimal when the costs are asymmetric. We show that our techniques also work when the cost matrix is essentially an effective resistance matrix, thus extending several results proved in [8].

Given a weighted (directed) graph with a weight (or *cost*) matrix $C = \{C_{ij}\}$, define the *cycle offset ratio* $\Psi(C)$ as follows. $\Psi(C)$ is the maximum over all sequences

$$v_0, \ldots, v_k = v_0 \text{ of } \frac{\sum_{i=1}^{k} C_{v_{i-1},v_i}}{\sum_{i=1}^{k} C_{v_i,v_{i-1}}}.$$

(Note that $1 \leq \Psi(C) \leq (n-1)$, since the costs satisfy the triangle inequality.)

We prove the following result.

Any random walk on a weighted graph with n vertices has stretch $\geq (n-1)/\Psi(C)$, and the ergodic walk has stretch $\leq (n-1)$, with equality under a symmetric C.

While we show examples that achieve equality in the lower bound, any tightening of the upper bound seems quite hard. However, it is interesting that the stretch of random walks on directed graphs can be brought down below $n-1$, while the counterpart on undirected graphs has an optimal bound of $n-1$.

The first application, for the *cat and mouse* game (see section 3), follows immediately from the above. It was mentioned in [8] that the cat and mouse game is at the core of several on-line algorithms. We show that for any $n \times n$ cost matrix C and any "blind" cat strategy (i.e., a random walk strategy), there is a mouse strategy that forces the competitiveness of the cat to be at least $(n-1)/\Psi(C)$, and the ergodic walk by the cat achieves a competitive ratio $\leq (n-1)$, on ergodic C.

[1] Similarly, an arbitrary cost matrix is not necessarily an effective resistance matrix for the techniques of [8] to work.

The second and more interesting application is for the notoriously hard k-server problem (see [11], [16], [20], [17]). The k-server problem (defined in [20]) is as follows. There are k mobile servers located on k vertices of a graph G with positive, real costs on the edges. (The costs can be thought of as distances between the positions.) An on-line algorithm manages the servers in such a way as to satisfy an on-line sequence of requests for service at vertices v_i, $i = 1, 2, \ldots$; i.e., servicing a request corresponds to moving a server to the requested vertex whenever a server isn't there. The algorithm pays a cost equal to the cost on the edge traversed by the server. The competitiveness of the algorithm is measured with respect to the cost an adversary pays, wherein the adversary moves the servers but also gets to choose the request sequence.

Due to the hardness of the k-server problem, it is significant to prove competitive ratios even for special cases such as special classes of graphs (e.g., [4], [8], [6]). Coppersmith et al. [8] provide one such example class. They use random walks to design optimal randomized k-competitive server algorithms when the cost matrix has a resistive inverse. We extend this class by allowing asymmetric costs, or equivalently, weighted *directed* graphs, with ergodic cost matrices.

Define *edge offset ratio* $\Psi'(C)$ to be $\max_{ij} \frac{C_{ij}}{C_{ji}}$. (Note that $\Psi(C) \leq \Psi'(C)$, and that $\Psi(C) = \Psi'(C) = 1$, for symmetric C.) We prove the following result for the *asymmetric* k-server problem. (This implies the result of [8].)

Let C be a cost matrix on n nodes. If every submatrix on $k+1$-nodes is ergodic, then we have a randomized $k\Psi'(C)$-competitive strategy for the k-server problem on C.

The final application is to the task system problem, defined as follows. We have a task system (S, C) for processing sequences of tasks wherein S is a set of states, and C is a cost matrix, describing the cost of changing from state i to state j. We assume that the costs satisfy the triangle inequality and that there is no cost of staying in the same state ($C_{ii} = 0$). Furthermore, when the costs are symmetric we refer to the task system as a *metrical* task system (MTS). Each task T has a cost vector v_T, where $v_T(i)$ is the cost of processing T in state i. A schedule for a given sequence of tasks $T_1, \ldots, T_k$ is a sequence of states $s_1, \ldots, s_k$, where s_i is the state in which T_i is processed. The *task system problem* is to design an *on-line* schedule (choose s_i only knowing $T_1, \ldots, T_i$) so that the algorithm is w-competitive—on any input sequence of tasks, the cost of the on-line algorithm is, barring an additive constant, at most w times that of the optimal off-line algorithm.

Borodin et al. [5] designed a deterministic algorithm with a competitive ratio of at most $(2n-1)\Psi(C)$ for the task system problem with asymmetric costs. If the costs are symmetric (i.e., an MTS), they prove a matching lower bound of $(2n-1)$. It is straightforward to extend their lower bound proof for the asymmetric case to get a lower bound of $(2n-1)/\Psi(C)$. Coppersmith et al. provided a simpler, *memoryless*, $(2n-1)$-competitive, randomized algorithm for any MTS, and also showed that no randomized algorithm can do better against an adaptive on-line adversary. We extend the results of [8] to the task system with ergodic cost matrices. In particular, we prove a lower bound of $(2n-1)/\Psi(C)$ for any randomized on-line scheduler. We also provide a randomized on-line scheduler with a competitive ratio of at most $(2n-1)$. While $(2n-1)$ is the best possible for symmetric cost matrices, our randomized scheduler achieves a competitive ratio of strictly less than $(2n-1)$, whenever the (ergodic) cost matrix is asymmetric. This shows that it is possible to design random walks with lower stretch on directed graphs than on undirected graphs. This also suggests that the deterministic algorithm of [5] (or a variant thereof) may have a competitive ratio

of at most $(2n-1)$ even under asymmetric costs.

Thus the novel technique of using the synthesis of random walks from hitting times for the design of on-line algorithms, while yielding the results of [8] for undirected graphs in a natural and simpler way, also yields results for directed graphs. We conclude this work with an interesting question on "approximating" an ergodic Markov chain, which will have useful implications in terms of extending our results to all cost matrices.

2. Results on ergodic Markov chains. In this section we prove two identities involving the first passage times and the transition probabilities. These results are crucial (later) to the analysis of our on-line algorithms. Lemma 2.1 and Theorem 2.2 below appeared in an earlier paper of this author [23]; however, the proofs are included here to keep the presentation self-contained.

Consider an electrical network on n nodes with resistors r_{ij} between nodes i and j. Let R_{ij} denote the *effective resistance* between the nodes. Then Foster's theorem asserts that

$$\sum_{i\sim j} \frac{R_{ij}}{r_{ij}} = n-1,$$

where $i \sim j$ denotes that i and j are connected by a finite r_{ij}. The proof appears in [13]. Also, [22] shows an alternative way (using random walks) of proving the same. In this section we prove an elementary identity for ergodic Markov chains which yields Foster's theorem when the chain is time-reversible.

Let P denote the transition probability matrix (size $n \times n$) of an ergodic Markov chain with stationary distribution π. Let $P_{ii} = 0\ \forall i$. Furthermore, let H denote the expected first-passage matrix (also size $n \times n$) of the above chain; i.e., H_{ij} denotes the expected time to reach state j starting from state i. We call these the *hitting times*. Then we have the following lemma.

LEMMA 2.1. $\sum_{i,j} \pi_j P_{ji} H_{ij} = n-1$.

Proof.

$$\sum_{i,j} \pi_j P_{ji} H_{ij} = \sum_j \pi_j \left(\sum_i P_{ji} H_{ij} \right) = \sum_j \pi_j [H_{jj} - 1]$$
$$= \sum_j \pi_j [1/\pi_j - 1] = n-1,$$

since $H_{jj} = 1/\pi_j$. □

We prove a stronger statement than Lemma 2.1 in the form of Corollary 2.7 below. Both Lemma 2.1 and Corollary 2.7 were formulated while attempting to interpret Foster's theorem in terms of ergodic Markov chains. For a proof that Lemma 2.1 implies Foster's theorem, see [23].

2.1. Synthesis of an ergodic walk. In the following we describe the construction (whenever one such exists) of an ergodic walk given an all-pairs hitting times matrix H. Given P as above, we define $\bar{P}$ to be the following $(n-1)\times(n-1)$ matrix. Let

$$\bar{P}_{ii} = \pi_i \left(= \sum_{\substack{j=1 \\ j\neq i}}^{n} \pi_i P_{ij} \right),$$

and $\bar{P}_{ij} = -\pi_i P_{ij}$, for $1 \le i, j \le n-1$.

Furthermore, let $\bar{H}_{jj} = H_{jn} + H_{nj}$, and $\bar{H}_{jk} = H_{jn} + H_{nk} - H_{jk}$, for $1 \le j, k \le n-1$. We use the standard notation of I_n for an identity matrix of size $n \times n$, and δ_{ij} to denote the entries of I. The following theorem is a generalization of the *resistive inverse identity* (well known in electrical network theory) used in [8].

THEOREM 2.2.

$$\bar{P}\bar{H} = I_{n-1}.$$

Proof. The basic identity we use is the triangle inequality for the hitting times. Using a "renewal type" theorem (see section 2.3 of [2] or Proposition 9-58 of [19]) one can show the following:

$$H_{xz} + H_{zy} - H_{xy} = \frac{N_y^{xz}}{\pi_y}, \tag{2.1}$$

$$H_{xz} + H_{zx} = \frac{N_x^{xz}}{\pi_x}. \tag{2.2}$$

(Recall that N_y^{xz} denotes the expected number of visits to y in a random walk from x to z.) From (2.1) and (2.2) we have

$$\bar{H}_{jk} = \frac{N_k^{jn}}{\pi_k} \quad \forall j, k.$$

Consider

$$\begin{aligned}
\sum_{j=1}^{n-1} \bar{P}_{ij}\bar{H}_{jk} &= \bar{P}_{ii}\bar{H}_{ik} + \sum_{\substack{j=1 \\ j\neq i}}^{n-1} \bar{P}_{ij}\bar{H}_{jk} \\
&= \pi_i \frac{N_k^{in}}{\pi_k} - \sum_{\substack{j=1 \\ j\neq i}}^{n-1} \pi_i P_{ij} \frac{N_k^{jn}}{\pi_k} \\
&= \frac{\pi_i}{\pi_k} \left[N_k^{in} - \sum_{\substack{j=1 \\ j\neq i}}^{n-1} P_{ij} N_k^{jn} \right] \\
&= \frac{\pi_i}{\pi_k} [\delta_{ik}] \qquad \text{(taking conditional means, given the first outcome)} \\
&= \delta_{ik}. \qquad \square
\end{aligned}$$

REMARK 1. *We have defined $\bar{P}$ and $\bar{H}$ by treating n as a special state of the chain. Clearly, we could have chosen any other state j and carried out a similar analysis.*

REMARK 2. *For reversible chains, we have $\bar{H}_{jk} = \bar{H}_{kj}$. This is because*

$$H_{jn} + H_{nk} + H_{kj} = H_{jk} + H_{kn} + H_{nj} \quad \forall i, j \quad (**).$$

The proof of this can be found in [9], or it can be verified directly by using the formula for the hitting times in terms of either resistances (see [22]) or the *fundamental*

matrix (see [18]). Thus the proof of Theorem 2.2 becomes simpler for the reversible case. (In particular, we do not need to use (2.1) and (2.2).)

An interesting consequence of Theorem 2.2 is that the property in (**) is sufficient (not only necessary) to imply reversibility. For (**) implies that $\bar{H}$ is symmetric which in turn implies that $\bar{P}$ is symmetric, i.e., $\pi_i P_{ij} = \pi_j P_{ji}\ \forall i, j$. This alternative characterization of reversibility is interesting for yet another reason: Interpreted in the electrical world, it can be shown (see [23]) to be equivalent to what is known as the *reciprocity theorem* (see [22]).

COROLLARY 2.3. *Given the hitting times, the chain can be tested for reversibility in $O(n^2)$ time.*

Proof. First we designate an arbitrary state as state n and then verify (**) for all pairs of vertices in $O(n^2)$ time. (Here we abuse notation by using n for both the number of states and the name of a particular state.) □

COROLLARY 2.4. *Given P and π, the hitting times (H_{ij}) can be computed with a single matrix inversion, and conversely, given the hitting times, P and π can be computed with a single matrix inversion.*

Proof. In view of Theorem 2.2, we need to show only (a) how to compute H from $\bar{H}$, and (b) how to compute P from $\bar{P}$.

(a) For $1 \le i, j \le n-1$, we have

$$H_{in} = \sum_k N_k^{in} = \sum_k \pi_k \bar{H}_{ik},$$

$$H_{ni} = \bar{H}_{ii} - H_{in},$$

$$H_{ij} = H_{in} + H_{nj} - \bar{H}_{ij}.$$

Thus we can first compute H_{in} and $H_{ni}\ \forall i < n$, and then compute H_{ij} for $1 \le i, j \le n-1$.

(b) We need to compute π_n and P_{ni}, since the rest of the information is available in $\bar{P}$. Since π is stochastic, and $\pi P = \pi$, we have

$$\pi_n = 1 - \sum_{i<n} \pi_i = 1 - \sum_{i<n} \bar{P}_{ii},$$

$$\pi_n P_{ni} = \pi_i - \sum_{j \ne i,n} \pi_j P_{ji} = \sum_{j \ne n} \bar{P}_{ji}. \qquad \square$$

We observe that Theorem 4.4.12 of [18] gives an alternative way of computing the chain, given all-pairs hitting times. However, the method outlined above seems simpler, since the solution can be written in essentially one equation; see Theorem 2.2.

2.2. A trace inequality. Based on some empirical results and Lemma 2.1 above, we conjectured that $\sum_{i,j} \pi_i P_{ij} H_{ij} \le n-1$, with equality under reversibility of the chain. This plays a crucial role in all our applications below, besides having an intrinsic importance. Recently, Aldous [1] proved this conjecture using a result due to Fiedler et al. [12]. We provide a slightly different proof using Theorem 2.2 and the main theorem in [12].

DEFINITION 2.5. *An M-matrix is an $n \times n$ matrix A of the form $A = \alpha I - P$ in which P is nonnegative and α is at least as big as the largest eigenvalue of P.*

An alternative characterization of nonsingular M-matrices (see [21]*) is that a nonsingular matrix A with nonpositive off-diagonal entries is an M-matrix iff $A^{-1} \geq 0$, meaning that all the nonzero entries are positive. From this, it is clear that the matrix $\bar{P}$ defined above is an M-matrix. The following theorem of Fiedler et al. is an interesting trace-inequality.*

THEOREM 2.6 (Fiedler et al. [12]). *For a nonsingular M-matrix A (size $n \times n$), $\mathrm{tr}(A^{-1}A^T) \leq n$, with equality holding iff A is symmetric.*

Now we are ready to state and prove the generalization of Lemma 2.1.

COROLLARY 2.7. *$\sum_{i,j=1}^{n} \pi_i P_{ij} H_{ij} \leq n-1$, with equality holding iff $P(,)$ is a reversible chain.*

Proof. Using Theorem 2.6 with $\bar{P}$ in place of A, we have $\mathrm{tr}(\bar{H}\bar{P}^T) \leq n-1$. We are done by noticing that

$$\begin{aligned}
&\mathrm{tr}(\bar{H}\bar{P}^T)\\
&= \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} \bar{H}_{ij}\bar{P}_{ij}\\
&= \sum_i \bar{H}_{ii}\pi_i - \sum_{i\neq j} \bar{H}_{ij}\pi_i P_{ij}\\
&= \sum_i [H_{in}+H_{ni}]\pi_i - \sum_{i\neq j}[H_{in}+H_{nj}-H_{ij}]\pi_i P_{ij}\\
&= \sum_i [H_{in}+H_{ni}]\pi_i - \sum_i H_{in}\pi_i(1-P_{in})\\
&\quad - \sum_j H_{nj}(\pi_j - \pi_n P_{nj}) + \sum_{i,j=1}^{n-1} \pi_i P_{ij} H_{ij}\\
&= \sum_{i,j=1}^{n} \pi_i P_{ij} H_{ij}. \qquad \square
\end{aligned}$$

3. Lower and upper bounds on stretch. We recall the definition of stretch of a random walk from the introduction: a random walk is said to have stretch s if there exists an $a > 0$ such that, for every $v_0, \ldots, v_k$, $\sum_{i=1}^{k} e_{v_{i-1}v_i} \leq s \sum_{i=1}^{k} C_{v_{i-1}v_i} + a$. The following facts follow easily from the definition of stretch.

FACT 1. *If a random walk has stretch s on cost matrix $C = \{C_{ij}\}$, then the walk has the same stretch on $C' = \{\beta C_{ij}\}$, where β is any positive constant.*

FACT 2. *In computing the stretch of a random walk, it suffices to consider sequences of vertices, $v_0, v_1, \ldots, v_k = v_0$, that form simple cycles in the graph G.*

Note that if a random walk has a stretch of c on simple cycles, then since any cycle can be decomposed as the union of disjoint simple cycles, the random walk will have stretch c on arbitrary closed paths. Now, as shown on page 426 of [8], Fact 2 follows from the fact that we gave ourselves room by allowing for an additive constant in the definition of stretch.

Let $C = \{C_{ij}\}$ be the given cost matrix of size $n \times n$.

DEFINITION 3.1. *Let $\Psi(C)$ be defined as the maximum over all cycles $(v_0, \ldots, v_k = v_0)$ of the ratio*

$$\frac{\sum_{i=0}^{k-1} C_{v_i, v_{i+1}}}{\sum_{i=0}^{k-1} C_{v_{i+1}, v_i}}.$$

If we assume that the costs satisfy the triangle inequality, then it is easy to see that $\Psi(C) \leq n-1$. *(Note that* Ψ *is defined in* [5] *and is termed the cycle offset ratio.)*

THEOREM 3.2. *Any random walk over a directed weighted graph has stretch at least* $(n-1)/\Psi(C)$, *where* $C = \{C_{ij}\}$ *is an* $n \times n$ *matrix specifying weight* C_{ij} *on edge* (i,j).

Proof. Note that the lower bound is $n-1$, when C is symmetric, since $\Psi(C) = 1$. In fact, the proof is identical to Theorem 1 of [8], wherein the symmetric case was dealt with. □

We now introduce the notion of *ergodic* cost matrices, which subsumes the class of resistive cost matrices, introduced in [8]. First, we describe two types of cost matrices for which there exist random walks with stretch at most $n-1$.

Type I. Let $C_{ij} = H_{ij}$ $\forall i,j$, $i \neq j$, where H_{ij} denote the hitting times of an n-state ergodic Markov chain.

CLAIM 1. *Every cost matrix of Type* I *has a random walk with stretch at most* $n-1$.

Proof. Note that, in view of Fact 2, it suffices to bound stretch over all simple cycles; this can then be extended to all paths, with an additive constant such as $\max_{i,j} C_{ij}$. □

Consider the walk with H_{ij} as the hitting times. The expected cost per move is

$$E = \sum_{i,j} \pi_i P_{ij} C_{ij} = \sum_{i,j} \pi_i P_{ij} H_{ij} \leq (n-1)$$

by Corollary 2.7. The claim now follows by noticing that the expected cost of a traversal over any sequence $v_0, \ldots, v_k$ of vertices equals $E \times \sum_{i=0}^{k-1} H_{v_i v_{i+1}} = E \times \sum_{i=0}^{k-1} C_{v_i v_{i+1}}$.

Type II. Let $C_{ij} = \frac{1}{2}[H_{ij} + H_{ji}]$ $\forall i,j$, $i \neq j$, where H_{ij} now denote the hitting times of any *reversible* Markov chain.

CLAIM 2. *Every cost matrix of Type* II *has a random walk with stretch at most* $n-1$.

Proof. As in the proof of Claim 1, without loss of generality, it suffices to bound stretch over all simple cycles. □

The expected cost per move is

$$E = \sum_{i,j} \pi_i P_{ij} C_{ij} = \frac{1}{2} \sum_{i,j} \pi_i P_{ij} [H_{ij} + H_{ji}] = n-1$$

by Lemma 2.1 and reversibility of P. Moreover, the expected cost of a traversal over any (cyclic) sequence $v_0, \ldots, v_k = v_0$ of vertices is, as before, equal to

$$\begin{aligned}
& E \times \sum_{i=0}^{k-1} H_{v_i v_{i+1}} \\
&= E \times \frac{1}{2} \left[\sum_{i=0}^{k-1} H_{v_i v_{i+1}} + \sum_{i=0}^{k-1} H_{v_{i+1} v_i} \right] \quad \text{by (**)} \\
&= (n-1) \sum_{i=0}^{k-1} \frac{1}{2} [H_{v_i v_{i+1}} + H_{v_{i+1} v_i}] \\
&= (n-1) \times \sum_{i=0}^{k-1} C_{v_i v_{i+1}}.
\end{aligned}$$

Hence we have the claim.

REMARK 3. *Any quantitative sharpening of the trace inequality (Theorem* 2.6*) immediately gives an improved upper bound on stretch for cost matrices of Type* I.

DEFINITION 3.3. *A cost matrix is ergodic if it is either the hitting time matrix of an ergodic chain (Type* I*) or the commute time matrix of a reversible chain (Type* II*).*

Note that Theorem 2.2 *guarantees that we can test if a matrix is ergodic or not with essentially a single matrix inversion.*

THEOREM 3.4. *Every graph with an ergodic cost matrix has a random walk with stretch* $\leq n-1$*. Moreover, this walk (termed ergodic walk) can be designed with a single matrix inversion.*

Proof. From Claims 1 and 2, the first part of the theorem follows. We now show how Theorem 2.2 can be used to design the desired random walk. Let G be a graph with the ergodic cost matrix C. In view of (**) it is easy to see that $C_{in}+C_{nj}-C_{ij} = H_{in}+H_{nj}-H_{ij}$, regardless of whether C is of Type I or II. Define $\bar{H}_{ij} = C_{in}+C_{nj}-C_{ij}$, for $1 \leq i,j \leq n-1$. The rest should be obvious: We construct P, the transition probability matrix of the desired walk, by first computing $\bar{P}$ using Theorem 2.2. □

REMARK 4. *Recall that a cost matrix is resistive if the* C_{ij} *can be interpreted as effective resistance* R_{ij} $\forall i,j$*. Note that any resistive cost matrix is of Type* II *(modulo a constant factor), since effective resistance* R_{ij} *is essentially the commute time* $H_{ij}+H_{ji}$ *(modulo the same constant factor) of a reversible chain. This, together with Fact* 1*, shows that our results imply those of* [8].

The lower and upper bounds are obviously tight under symmetry, since $\Psi(C) = 1$. The following example shows that the lower bound is, in general, tight.

Example. Consider a directed cycle $1, 2, \ldots, n, 1$ with cost 1 on each directed edge $(i, i+1)$. Now put in all other edges to make a directed K_n and assign the distance along the original cycle to be the cost of each edge. Thus the cost matrix has $\Psi = n-1$. The optimal random walk (with stretch 1) is, in fact, the deterministic walk always going around the cycle. □

With each undirected cycle, we associate the following notion of a (strongly connected) "bicycle." A bicycle is a sequence of nodes $v_0, v_1, \ldots, v_{k-1}, v_0, v_{k-1}, \ldots, v_1, v_0$, i.e., the undirected cycle traversed once in either direction. The following asserts that our random walk is optimal over traversals of 1.

COROLLARY 3.5. *The stretch over any bicycle of any random walk is* $\geq (n-1)$*, and the ergodic walk achieves the equality.*

Proof. The equality is obvious in view of the preceding theorem. The proof of the lower bound is essentially the proof of Theorem 1 of [8]. □

The cat and mouse game. As mentioned in the introduction this game is a convenient tool in analyzing more complicated on-line strategies. For completeness, we describe the game here, but we refer the reader to [8] for further details and related interesting references. This is a game played for a fixed number of rounds between a cat and a mouse on a graph. Each round begins with both the cat and the mouse on the same vertex; the mouse moves once (and just once) at the beginning of the round to some (carefully chosen) vertex unknown to the cat. The rest of the round consists of the cat's moves (which could be deterministic or randomized) on the edges of the graph until the cat reaches the vertex that the mouse is at. Each move of the mouse can use information about all previous moves by the cat. A strategy for the cat is c-competitive if there exists an (additive) constant $a \geq 0$ such that for any number of rounds and any strategy of the mouse, the cat's expected cost is at most c times the

mouse's cost $+a$.

THEOREM 3.6. *For any $n \times n$ ergodic cost matrix C and for any random walk strategy by the cat, there is a mouse strategy that forces the competitiveness of the cat to be at least $(n-1)/\Psi(C)$, and the ergodic walk by the cat achieves a competitive ratio $\leq (n-1)$.*

Proof. It was pointed out in [8] that a random walk with stretch c defines a *memoryless* c-competitive strategy for the cat: in each round the cat, without recourse to its previous moves, executes a random walk with stretch c. Thus the upper bound is immediate from Theorem 3.4 above. For the lower bound, we use a standard argument (used, e.g., in [8] and [20]). Consider $(n-1)$ mice, one on each node except where the cat is. Whenever a cat moves from i to j, the mouse on j moves to i. Thus the mice together pay a cost of $\sum_{i=0}^{k-1} C_{v_{i+1},v_i}$, whenever the cat takes a walk $v_0, \ldots, v_k$ incurring a cost of $\sum_{i=0}^{k-1} C_{v_i,v_{i+1}}$. The single mouse strategy is going to be (just as in [8]) that we choose one of the $(n-1)$ strategies uniformly at random. By the definition of $\Psi(C)$, the mouse can always make the competitive ratio to be $\geq (n-1)/\Psi(C)$. □

4. *k*-servers with asymmetric costs. Consider the usual k-server problem with the triangle inequality constraint on the costs. An *adaptive on-line* adversary (provably different from the *oblivious* and the *adaptive off-line* ones) chooses the next request at each step, knowing the current position of the on-line algorithm and, if 1, moves one of its servers to satisfy the request. We describe here a randomized on-line algorithm that works well against such an adaptive on-line adversary. (The significance of results proved against such an adversary is as follows. It was shown in [3] that a c-competitive randomized on-line algorithm against an adaptive on-line adversary implies the existence of a c^2-competitive deterministic algorithm.) Let us assume that all the costs are positive and bounded. Let us, however, *not* make the assumption that the cost C_{ij} of moving a server from position i to j be the same as that of moving a server from j to i, C_{ji}. We also make the strong assumption that every $k \times k$ submatrix is *ergodic* in the sense defined above.

DEFINITION 4.1. *Let the edge offset ratio $\Psi'(C)$ be $\max_{i,j}(C_{ij}/C_{ji})$.*

We may simply write Ψ' to denote $\Psi'(C)$ as long as there is no confusion as to the underlying C. Note that $\Psi(C) \leq \Psi'(C)$, and when C is symmetric $\Psi(C) = \Psi'(C) = 1$.

We are now ready to state the main theorem of this section.

THEOREM 4.2. *Let C be a cost matrix on n nodes. If every submatrix on $k+1$-nodes is ergodic, then we have a randomized $k\Psi'(C)$ competitive strategy (against an adaptive on-line adversary) for the k-server problem on C.*

Proof. For convenience, we refer to the k-servers under our randomized on-line strategy as *randomized servers*. The strategy is as follows. Suppose the randomized servers are on positions 1 through k. Let the current request be on position x. We find the (unique) ergodic walk that corresponds to these $k+1$ vertices by interpreting the costs as the hitting times. (Note that this computation, for all choices of $k+1$ vertices, can be done once and for all at the beginning.) Let p_{ij} denote the transition probabilities of this walk. Then the strategy is to move the (randomized) server at j (for $j = 1, \ldots, k$) to x with probability $(p_{xj}/\sum_j p_{xj})$. (The probabilities clearly sum to 1, over all j.)

We can model the situation as a game between the randomized server and an adversary. The adversary controls k "off-line" servers. In each round of the game, the adversary picks the next request (vertex) and moves one of his servers to that vertex.

Then the randomized server uses her strategy to move one of her servers to the request. We want to show that the expected cost of the randomized server is within a constant factor of k times the cost incurred by the off-line server. Toward this, let us denote the positions of the randomized servers and the adversary's servers by $\mathbf{a} = \{a_1, \ldots, a_k\}$ and $\mathbf{b} = \{b_1, \ldots, b_k\}$, respectively. We define the following "potential function" Φ:

$$\Phi(\mathbf{a}, \mathbf{b}) = \sum_{i,j=1}^{n} C_{ij} - \sum_{x} \sum_{j} C_{xa_j} + k \min_{\sigma} \sum_{i} C_{b_i a_{\sigma(i)}},$$

where x ranges over nodes where there is currently no randomized server, and σ ranges over (directed) matchings between the adversary's positions and the randomized servers' positions. (The first term in the definition of Φ is introduced simply to make Φ positive and is thus not essential to the proof.) Let $Cost_R$ and $Cost_A$ denote the accumulated costs of the randomized servers and the adversary's servers, respectively.

Furthermore, let

$$\Delta = \Phi + Cost_R - k\psi' \times Cost_A.$$

We would like to show that Δ is always nonincreasing with every move of either the randomized servers or the adversary.

Adversary's move. Consider a move by the adversary's server from node b_j to $b_{j'}$. The adversary clearly pays cost $C_{b_j b_{j'}}$. We want to show that $\Delta(new) - \Delta(old) = \Phi(new) - \Phi(old) - k\psi' C_{b_j b_{j'}}$ is ≤ 0. Let the minimum matching in $\Phi(old)$ be $b_i \to a_i \forall i$. Then define a new matching as follows. Match b_i with a_i for $i \neq j'$, and match $b_{j'}$ with a_j. Clearly,

$$\begin{aligned}
&\Phi(new) - \Phi(old) - k\psi' C_{b_j b_{j'}} \\
&\leq kC_{b_{j'} a_j} - kC_{b_j a_j} - k\psi' C_{b_j b_{j'}} \\
&\leq kC_{b_{j'} a_j} - kC_{b_j a_j} - kC_{b_{j'} b_j} \qquad \text{(by the definition of } \psi') \\
&\leq 0, \qquad \text{since} \quad C_{b_{j'} b_j} + C_{b_j a_j} \geq C_{b_{j'} a_j}.
\end{aligned}$$

Randomized servers' move. For this half of the proof, it was shown in [8] that (restrictive as it may sound) it suffices to prove the case when the randomized servers and the adversary agree on (i.e., share the same vertices) $k-1$ positions and differ in only one position. The same justification applies in our situation as well, and we omit the straightforward proof.

Thus let us assume without loss of generality that the adversary occupies positions 1 through $m - 1 = k$, and the randomized servers occupy positions 2 through m. Let the request be on 1. So, the randomized server moves from j to 1 with probability $p_{1j}/(\sum_j p_{1j})$, paying a cost of C_{j1}. Here and in the claim below, $\sum_j$ represents $\sum_{j=2}^{m}$. Note that $\sum_j p_{1j}$ is equal to 1, if $p_{11} = 0$, but in general, $\sum_j p_{1j} = 1 - p_{11}$.

Claim.

$$\sum_{j} \frac{p_{1j}}{\sum_j p_{1j}} [\Phi(new) - \Phi(old) + C_{j1}] = 0.$$

Proof. Clearly, the minimum matching before the move has cost C_{1m}, since the minimium matching consists of the (directed) edge $(1, m)$. Similarly, after the move

(of the server from j to 1), the minimum matching has cost C_{jm} (match j with m), since the adversary and the randomized servers are identical everywhere else. Thus

$$\begin{aligned}
&[\Phi(new) - \Phi(old) + C_{j1}] \\
&= -\sum_{i=1}^{m} C_{ji} + \sum_{i=2}^{m} C_{1i} + kC_{jm} - kC_{1m} + C_{j1} \\
&= -\sum_{i=2}^{m} C_{ji} + \sum_{i=2}^{m} C_{1i} + kC_{jm} - kC_{1m} \\
&= -\sum_{i=2}^{m} H_{ji} + \sum_{i=2}^{m} H_{1i} + kH_{jm} - kH_{1m}.
\end{aligned}$$

Thus

$$\begin{aligned}
&\sum_j \frac{p_{1j}}{\sum_j p_{1j}} [\Phi(new) - \Phi(old) + C_{j1}] \\
&= \frac{1}{\sum_j p_{1j}} \left[-\sum_{i=2}^{m} \sum_j p_{1j} H_{ji} + \sum_{i=2}^{m} H_{1i} \sum_j p_{1j} \right] + \frac{1}{\sum_j p_{1j}} \left[k \sum_j p_{1j} H_{jm} - kH_{1m} \sum_j p_{1j} \right] \\
&= \frac{1}{\sum_j p_{1j}} \left[\sum_{i=2}^{m} \left(-\sum_j p_{1j} H_{ji} + H_{1i}(1 - p_{11}) \right) \right] \\
&+ \frac{1}{\sum_j p_{1j}} \left[k \left(\sum_j p_{1j} H_{jm} - H_{1m}(1 - p_{11}) \right) \right] \\
&= \frac{1}{\sum_j p_{1j}} \left[\left(\sum_{i=2}^{m} 1 \right) - k \right] \qquad (\text{since } H_{1i} = 1 + p_{11} H_{1i} + \textstyle\sum_{j=2}^{m} p_{1j} H_{ji}) \\
&= 0.
\end{aligned}$$

We showed that Δ is always nonincreasing. This concludes the proof that the randomized strategy is $k\Psi'$-competitive against an adaptive on-line adversary. □

5. Task systems. Recall the description of a task system from the introduction: We have a task system (S, C) for processing sequences of tasks wherein S is a set of states, and C is a cost matrix, describing the cost of changing from state i to state j. The *task system problem* is to design an *on-line* schedule (choose s_i only knowing $T_1, \ldots, T_i$) so that the algorithm is w-competitive—on any input sequence of tasks, the cost of the on-line algorithm is, barring an additive constant, at most w times that of the optimal off-line algorithm. In [5] it was shown that $w(S, C) = 2|S| - 1$ for every *metrical* task system (MTS), and $w(S, C) \leq (2|S| - 1)\Psi(C) = O(|S|^2)$, for every task system. Subsequently, [8] gave a $(2|S| - 1)$-competitive *randomized* on-line algorithm for every MTS. It is to be noted that although this is a weaker result in light of the deterministic algorithm of [5], the randomized algorithm is conceptually and otherwise much simpler and is, moreover, *memoryless*.

Our contribution is as follows. We prove the analogous simplification for the (nonmetrical) task systems using randomization. We prove a lower bound of $(2|S| - 1)/\Psi(C)$ on the competitive ratio of any randomized on-line algorithm and provide a randomized on-line scheduler with $w(S, C) \leq (2|S| - 1)$. Note that our results imply

those of [8] in the case of metrical task systems. In the asymmetric case, we provide improved bounds to those of [5] with simpler memoryless randomized schedulers. However, we do have the restriction that the cost matrix should essentially be a hitting time matrix of an ergodic chain. Thus the random walk we refer to below is the ergodic walk designed by interpreting the cost matrix as the hitting time matrix as explained in sections 2 and 3.

5.1. Lower and upper bounds. The following lower bound on the competitiveness of any deterministic or randomized algorithm for the task system problem is straightforward to prove from the proofs in [5] and [8] for the lower bounds of the MTS problem.

THEOREM 5.1. *Any deterministic or randomized on-line algorithm for the task system problem has a competitive ratio of at least* $(2n-1)/\Psi(C)$ *against an adaptive on-line adversary.*

Proof. The proof is similar to the proofs for the symmetric case. For the deterministic part, follow the proof of Theorem 2.2 of [5]. For the randomized part, the proof is essentially that of Theorem 11 of [8]. □

For the upper bound we use the basic traversal algorithm first described in [5] and also used in [8] in the following modified form:

(i) The positions are visited in a sequence, that is, prescribed by a random walk, but independent of the input task sequence $T_1, T_2, \ldots$.

(ii) There is a sequence of positive *threshold* costs $\beta_1, \beta_2, \ldots$ such that the transition from s_i to s_{i+1} occurs when the total task processing cost incurred since entering s_i reaches β_i.

Please refer to both [5] and [8] for a full account on such traversal algorithms, especially for the technical difficulties involved. The only originality on our part is in choosing an appropriate value for β_i. We simply choose β_i to be the return time $H_{s_i s_i}$; i.e., when in state j, the random scheduler leaves state j once the task processing cost incurred since reaching j equals the expected time for a random walk beginning from j to return to j for the first time. (This has an intuitive appeal!) Once again the analysis in our case turns out to be quite simple.

THEOREM 5.2. *There exists a randomized traversal algorithm for task systems (with ergodic cost matrices) which is* $(2n-1)$*-competitive.*

Proof. By *total cost* we mean the sum of the task processing costs and the moving costs. We can assume without loss of generality (see [8]) that the adversary is a "cruel taskmaster," i.e., changes position only at the time the randomized on-line algorithm reaches its current position. We further distinguish *moving phases*, where the adversary changes positions, and *staying phases*, where the adversary stays in the same position but the randomized server moves. Let the current position be i. We first consider the cost incurred by the (randomized) on-line algorithm. Recall that we set $\beta_i = H_{ii}$. Also, from our results in section 3, recall that the expected cost per move is $E \leq (n-1)$. The expected task processing cost per move is $\sum_i \pi_i \beta_i = \sum_i \pi_i H_{ii} = n$, since π_i is the steady state probability of being at i. So, the ratio of the expected total cost to expected cost per move (of the on-line algorithm) is $[n+E]/E$. Note that it suffices to show that the expected moving cost of the on-line algorithm is at most E times the cost of the adversary—we would then have a competitive ratio of $[n+E] \leq (2n-1)$. We show this in the following two phases, depending on when the adversary is moving (the moving phase) and when the adversary is staying (the staying phase):

1. *The moving phase.* The average cost in this phase can be analyzed as a cat

and mouse game, with the adversary playing the mouse's role, yielding a competitive ratio of E.

2. *The staying phase.* The cost of the adversary starting (and ending) at node i is β_i, whereas the expected moving cost of the on-line algorithm in that phase is $E \times H_{ii} = E\beta_i$. □

5.2. Memoryless counterpart. The above algorithm needs to store a current virtual position and a counter for the accumulated task processing cost at that position. However, this can also be made *memoryless* in the spirit of [8], without sacrificing the competitive ratio with the idea of using a *probabilistic counter*. We omit further discussion since the details are the same as those of [8], modulo the following aspect. We need to bound the stretch of a random walk while allowing for positive costs on self-loops. We merely state the requisite lemma and outline the proof idea.

LEMMA 5.3. *The ergodic walk has a stretch of at most* $(2n-1)$*, on a graph with cost matrix* C*, where* C_{ii} *are not necessarily zero.*

Proof idea. We adopt the procedure that was described in detail in [8]. We merely suggest the solution and omit the proof, since it is similar to that of Theorem 7 of [8]. Intuitively, the idea is to place a special vertex on each self-loop with the appropriate transition probabilities and appeal to the case of no self-loops. Since the number of vertices is doubled (in the worst case), the stretch is at most $2n-1$ rather than $n-1$. □

6. Loose ends. The most important issue here is in extending our results to *all* cost matrices (say, those that satisfy the triangle inequality). In particular, tighter upper and lower bounds on a stretch of random walks on directed graphs would constitute significant progress. Coppersmith et al. [8] extend their results from *resistive* cost matrices to all cost matrices (at least existentially, if not with an efficient construction); i.e., they show that given any symmetric cost matrix $\{C_{ij}\}$, satisfying the triangle inequality, there exists a resistive (approximation) network (with conductances c_{ij}) such that the effective resistance $R_{ij} \leq C_{ij}$ whenever $c_{ij} \geq 0$, with $R_{ij} = C_{ij}$ whenever $c_{ij} > 0$. Much in the same spirit we would like to aim for an ergodic Markov chain with the property that $H_{ij} \leq C_{ij}$, with equality whenever $p_{ij} > 0$. The existence of such an "approximate chain" is clearly implied by the result of [8] when we are seeking a reversible chain. The main hurdle in mimicking the approach taken in [8] for the nonreversible case is the following. Consider the space of all $n-1 \times n-1$ matrices, $\mathcal{P} = \{\bar{P}\}$, where $\bar{P}$ corresponds (as in Theorem 2.2) to an ergodic chain on n states. It is easy to see that for $0 \leq \alpha \leq 1$,

$$\bar{P}_1, \bar{P}_2 \in \mathcal{P} \quad \Rightarrow \quad \alpha\bar{P}_1 + (1-\alpha)\bar{P}_2 \in \mathcal{P}.$$

Let's even assume that $\mathcal{P}$ is a space of positive definite matrices. Now the function, log of the determinant, is concave over the space of positive definite *symmetric* (or Hermitian) matrices, and symmetry is crucial here. We have examples of asymmetric $\bar{P}$ (i.e., nonreversible chains) which show that the log of the determinant is neither concave nor convex over $\mathcal{P}$. This makes the analogous result for the nonreversible case much harder. On the other hand, Coppersmith et al. benefit from convexity as follows. They formulate the approximation as an appropriate convex programming problem; the existence of the approximate chain (resistive network) is then guaranteed by simply appealing to the "Kuhn–Tucker" (necessary and sufficient) conditions arising in the solution of the convex programming problem. It is still conceivable that a somewhat different approach works for the nonreversible case.

Acknowledgments. The author thanks Steve Phillips, Prabhakar Raghavan, Nick Reingold, and Emre Telatar for many useful discussions. The author would also like to thank Mike Saks (the no-longer-anonymous referee) for a careful reading and constructive criticism of the manuscript.

REFERENCES

[1] D. Aldous, *Personal communication,* 1993.
[2] D. Aldous and J. Fill, *Reversible Markov Chains and Random Walks on Graphs*, draft of book in preparation.
[3] S. Ben-David, A. Borodin, R. M. Karp, G. Tárdos, and A. Wigderson, *On the power of randomization in on-line algorithms*, Algorithmica, 11 (1994), pp. 2–14.
[4] P. Berman, H. J. Karloff, and G. Tárdos, *A competitive* 3*-server algorithm*, in Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms, SIAM, Philadelphia, PA, 1990, pp. 280–290.
[5] A. Borodin, N. Linial, and M. Saks, *An Optimal on-line algorithm for metrical task system*, J. Amer. Math. Soc., 39 (1992), pp. 745–763.
[6] M. Chrobak and L. L. Larmore, *An optimal on-line algorithm for k-servers on trees*, SIAM J. Comput., 20 (1991), pp. 144–148.
[7] A. K. Chandra, P. Raghavan, W. L. Ruzzo, R. Smolensky, and P. Tiwari, *The electrical resistance of a graph captures its commute and cover times*, Comput. Complexity, 6 (1996/97), pp. 312–340.
[8] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir, *Random walks on weighted graphs, and applications to on-line algorithms*, J. Amer. Math. Soc., 40 (1993), pp. 421–453.
[9] D. Coppersmith, P. Tetali, and P. Winkler, *Collisions among random walks on a graph*, SIAM J. Discrete Math., 6 (1993), pp. 363–374.
[10] P. G. Doyle and J. L. Snell, *Random Walks and Electric Networks*, Carus Mathematical Monographs 22, Mathematical Association of America, Washington, DC, 1984.
[11] A. Fiat, Y. Rabani, and Y. Ravid, *Competitive k-server algorithms*, J. Comput. System Sci., 48 (1994), pp. 410–428.
[12] M. Fiedler, C. R. Johnson, T. L. Markham, and M. Neumann, *A trace inequality for M-matrices and the symmetrizability of a real matrix by a positive diagonal matrix*, Linear Algebra Appl., 71 (1985), pp. 81–94.
[13] R. M. Foster, *The average impedance of an electrical network*, in Contributions to Applied Mechanics (Reissner Anniversary Volume), Edwards Bros., Ann Arbor, MI, 1949, pp. 333–340.
[14] R. M. Foster, *An extension of a network theorem*, IRE Trans. Circuit Theory, 8 (1961), pp. 75–76.
[15] F. Gobel and A. A. Jagers, *Random walks on graphs*, Stochastic Process. Appl., 2 (1974), pp. 311–336.
[16] E. Grove, *The harmonic online K-server algorithm is competitive*, in On-Line Algorithms, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., 7, AMS, Providence, RI, 1992, pp. 65–75.
[17] E. Koutsoupias and C. H. Papadimitriou, *On the k-server conjecture*, J. Amer. Math. Soc., 42 (1995), pp. 971–983.
[18] J. G. Kemeny and J. L. Snell, *Finite Markov Chains*, Springer-Verlag, New York, 1983.
[19] J. G. Kemeny, J. L. Snell, and A. W. Knapp, *Denumerable Markov Chains*, Springer-Verlag, New York, 1976.
[20] M. S. Manasse, L. A. McGeoch, and D. D. Sleator, *Competitive algorithms for server problems*, J. Algorithms, 11 (1990), pp. 208–230.
[21] H. Minc, *Nonnegative Matrices*, Wiley-Interscience, New York, 1988.
[22] P. Tetali, *Random walks and the effective resistance of networks*, J. Theoret. Probab., 4 (1991), pp. 101–109.
[23] P. Tetali, *An extension of Foster's network theorem*, Combin. Probab. Comput. (special issue in honor of Paul Erdős's 80th birthday), 3 (1994), pp. 421–427.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1247–1263

EFFICIENT GENERATION OF MINIMAL LENGTH ADDITION CHAINS*

EDWARD G. THURBER†

Abstract. An addition chain for a positive integer n is a set $1 = a_0 < a_1 < \cdots < a_r = n$ of integers such that for each $i \geq 1$, $a_i = a_j + a_k$ for some $k \leq j < i$. This paper is concerned with some of the computational aspects of generating minimal length addition chains for an integer n. Particular attention is paid to various pruning techniques that cut down the search time for such chains. Certain of these techniques are influenced by the multiplicative structure of n. Later sections of the paper present some results that have been uncovered by searching for minimal length addition chains.

Key words. addition chain, search tree, pruning bounds, backtracking, branch and bound

AMS subject classifications. 11Y16, 11Y55, 68Q25

PII. S0097539795295663

1. Introduction.

Dellac [6] asks what is the minimum number of multiplications required to raise a number A to the power m. Since exponents are added when powers of the same base are multiplied, this gives rise to the concept of an addition chain. This topic has been studied somewhat extensively over the intervening 100 years. Many questions have been posed concerning this deceptively simple problem. While some of these have been answered, others remain tantalizingly open.

This paper explores some of the computational aspects of generating minimal length addition chains for an integer n. The emphasis on computation is of interest in light of the surprising discoveries that the generation of minimal length addition chains has uncovered, some of which led Knuth [12] to say that perhaps no conjecture concerning addition chains is safe. Some of these discoveries will be mentioned in section 10.

The algorithm to be explored for generating minimal length addition chains is a backtracking algorithm that uses branch and bound methods to prune the search tree. Since the algorithm is exponential in nature, particular attention will be paid to pruning the search tree. The pruning bounds dramatically increase the efficiency of the search and increase the feasibility of pursuing a variety of questions concerning addition chains. For example, the determination of $c(r)$, the first integer requiring r steps in an addition chain of minimal length, is greatly facilitated by these methods.

Of course, the trick when pruning a search tree is not to cut off too much. Establishing the validity of the pruning bounds used in the backtracking algorithm will be a primary focus of what follows. It will be seen that certain classes of pruning bounds are affected by the multiplicative structure of n.

The paper is organized as follows. Section 2 includes some notation and prior results. Section 3 develops the notion of the search tree for addition chains. In section 4, an algorithm is discussed which will find all the minimal addition chains for the integer n. The algorithm is adapted easily to find just one such chain. In section 5, the pruning bounds that come most readily to mind are developed. In

*Received by the editors December 6, 1995; accepted for publication (in revised form) September 16, 1997; published electronically March 22, 1999.
http://www.siam.org/journals/sicomp/28-4/29566.html
†Department of Math and Computer Science, Biola University, La Mirada, CA 90639 (ed@irvine.com).

section 6, improvements are made to class 1 bounds. Section 7 discusses pruning bounds that involve more than one member of an addition chain. In section 8, class 1 and class 2 bounds are applied to n and their effectiveness is explored. In section 9, class 1 pruning bounds are refined in a way that is influenced by the multiplicative nature of n, and a summary is given of the pruning bounds developed in the paper. Section 10 further explores the efficiency of the various classes of pruning bounds and presents some computational results found concerning the addition chain problem. Finally, in section 11 concluding remarks are made with some reference to future directions.

2. Preliminaries. An addition chain for a positive integer n is a set $1 = a_0 < a_1 < \cdots < a_r = n$ of integers such that for every $i \geq 1$, $a_i = a_j + a_k$ for some $k \leq j < i$. The minimal length, r, of an addition chain for n is denoted by $l(n)$. As in Knuth [12], $\lambda(n) = \lfloor \log_2 n \rfloor$, and $\nu(n)$ will denote the number of ones in the binary representation of n. The function NMC(n) was introduced by the author [19] and denotes the number of minimal addition chains for n. For $n = 29$, a minimal addition chain is $1, 2, 4, 8, 9, 13, 16, 29$. In base 2, $29 = 11101_2$. Thus, $\nu(29) = 4, \lambda(29) = \lfloor \log_2 29 \rfloor = 4$, $l(29) = 7$, and, as it turns out, NMC(29) = 132.

As Knuth observed, either $\lambda(a_i) = \lambda(a_{i-1})$ or $\lambda(a_i) = \lambda(a_{i-1}) + 1$. In the former case, step i is called a *small step* and is called a *big step* otherwise. There are exactly $\lambda(n)$ big steps in any chain for n. The number of steps, r, in an addition chain for n can be expressed as $r = \lambda(n) + N(n)$, where $N(n)$ denotes the number of small steps in the chain. It should be noted that $N(n)$ is chain dependent. Minimizing $N(n)$ will result in a minimal length addition chain for n. If $j = i - 1$, then step i is called a *star step*. An addition chain that consists entirely of star steps is called a *star chain*. If $j = k = i - 1$, then step i is called a *doubling*.

Theoretically developed lower bounds for $l(n)$ provide starting values from which to start looking for minimal length addition chains. It is conjectured [12] that $l(n) \geq \lambda(n) + \lceil \log_2 \nu(n) \rceil$. If $\nu(n) \geq 2^m + 1$, the conjecture states that $l(n) \geq \lambda(n) + m + 1$. The conjecture has been established for $m = 0, 1, 2, 3$ [17], and it is known that the conjecture holds for all $n \leq$ 327,678. Schönhage [14] has shown that $l(n) \geq \log_2 n + \log_2 \nu(n) - 2.13$. Thus, $l(n) \geq \lceil \log_2 n + \log_2 \nu(n) - 2.13 \rceil$ in any event.

3. The search tree. A tree organization for the solution space for finding addition chains for n is as follows:

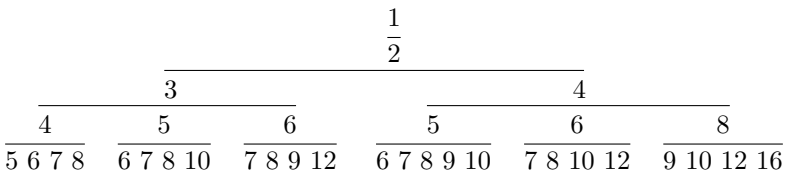


FIG. 3.1.

This tree organization is found in a paper by Chin and Tsai [5] in which they develop algorithms for finding minimal or near minimal addition chains for integers n. The tree shall be referred to as the search tree. The algorithm in this paper traverses the search tree in a fashion similar to that found in [5]. The search is depth first and considers larger numbers first (i.e., all paths that start $1, 2, 4$ are taken before any path that starts $1, 2, 3$). In what follows the emphasis is on developing and establishing the

validity of pruning bounds used in the branch and bound method to greatly speed up the search for minimal addition chains.

If $n = 1$ is at *level* 0, then 2 is at *level* 1, 3 and 4 are at *level* 2, etc. The children of a given node a_i at level i are all numbers $a_{i+1} > a_i$ formed as sums $a_j + a_k$, $k \le j \le i$ of numbers in the path from 1 to a_i. Any addition chain for an integer n can be found by taking the path from 1 down to the appropriate occurrence of n in the search tree. The tree grows exponentially as can be seen by the fact that each integer at level k in the tree has at least $k + 1$ children. This means that the number of paths to level k is at least $k!$ To find $l(n)$ and NMC(n) is to find on what level n appears first and how many times it appears on this level. If $1 = a_0, a_1, \ldots, a_i$ is a partial chain, then br$[a_i]$ is the branch of the search tree that has a_i for its root.

4. Algorithm for finding all minimal addition chains. The backtracking algorithm for finding all minimal addition chains for an integer n starts by setting $r = \lambda(n) + \lceil \log_2 \nu(n) \rceil$ if $n \le 327{,}678$ or $\nu(n) \le 16$. Otherwise, it sets $r = \lceil \log_2 n + \log_2 \nu(n) - 2.13 \rceil$. If no chain of length r is found, then set r to $r + 1$ and repeat. The process is continued until a length is tried for which chains for n exist. This length will be $l(n)$. By the binary chain method [12], $l(n) \le \lambda(n) + \nu(n) - 1$.

Starting with minimum estimates for $l(n)$ and working up is preferable to over estimating $l(n)$ and working down since it minimizes the number of small steps in the addition chains being generated. It is the increase in small steps that adds significantly to the search time.

In what follows, *lb*, which stands for lower bound, will be used in place of r. The search commences with a lower bound *lb* for $l(n)$ and increases this value incrementally if necessary. The following algorithm traverses the search tree and develops partial addition chains $1 = a_0, a_1, \ldots, a_i$ into minimal length addition chains for n. A stack is maintained which holds the possible children of a_i at each stage. The children of a_i constitute a stack segment.

Algorithm find_minimum_chains(n).
begin
  if $n \le 327{,}678$ **or** $\nu(n) \le 16$ **then**
    $lb = \lambda(n) + \lceil \log_2 \nu(n) \rceil$
  else
    $lb = \lceil \log_2 n + \log_2 \nu(n) - 2.13 \rceil$
  end if
  $a_0 = 1$, $a_1 = 2$ (first two elements of an addition chain)
  loop *lb*_value
    determine pruning bounds
    $i = 1$
    loop find_chains
      if $i < lb$ **then**
        determine whether to retain a_i
        if a_i is retained **then**
          stack the possibilities for a_{i+1} in increasing order in next stack
            segment (all sums $a_j + a_k > a_i, k \le j < i + 1$)
          increment i by 1
          let a_i be the element on top of the stack
          if $a_i = n$ **then**
            chain is found

```
            back up in the search tree until a node is found that needs further
              expanding (i.e., take the next element off the stack that is not
              in the stack segment of a_i)
          end if
        else
          back up in the search tree until a node is found that needs further
            expanding (i.e., take the next element off of the stack)
          let a_i be the element on top of the stack
        end if
      else
        back up in the search tree until a node is found that needs further
          expanding (i.e., take the next element off of the stack that is not
          in the stack segment of a_i)
        let a_i be the element on top of the stack
      end if
    end loop find_chains
    if no chains found then
      increase lb by 1
    else
      exit loop lb_value
    end if
end loop lb_value.
```

When an element a_i is taken off of the top of the stack, it is either n, in which case a minimal addition chain for n has been found, or it becomes part of a partial chain $1 = a_0, a_1, \ldots, a_i$ provided that it is not pruned from the search tree. The possible children a_{i+1} of a_i are put on the stack in increasing order. Some pruning can take place at this point, also. Backing up in the search tree is accomplished by taking elements off the stack. The stack is popped until an element is found that cannot be pruned. If, for instance, it is determined that a_i is to be pruned and that this exhausts all possibilities from a_{i-1}, then the next element on the stack is the next child of a_{i-2} in decreasing order (if children of a_{i-2} still remain; otherwise, backtrack further). The next child of a_{i-2} (the new a_{i-1}) is added to the partial chain $1 = a_0, a_1, \ldots, a_{i-2}$ provided that it is not pruned, and its children are added to the stack. When the algorithm backs up to $a_1 = 2$, it will terminate since the $1, 2, 4$ and $1, 2, 3$ branches (br[4] and br[3]) will have been traversed, and these are the only possibilities from level 2 in the search tree. At this point, either all the minimal addition chains for n will have been found or none will have been found, in which case lb is increased by 1 and the process is repeated. The algorithm will exit loop find_chains when the stack is empty. A slight variation of the algorithm will terminate once the first addition chain for n is found.

It should be mentioned that this algorithm is particularly well suited for finding all the minimal length addition chains for n. If one were interested in finding only one such chain, other strategies might be employed such as limiting the search to star chains at first and checking only for nonstar chains if no star chains are found. The pruning bounds, however, apply to whatever strategy is employed for traversing the search tree.

Two classes of bounds will be discussed. The first class (class 1 bounds) concerns bounds for a_i, while the second class (class 2 bounds) concerns bounds for $a_i + a_{i-1}$. Class 1 bounds will be refined to further improve the efficiency of the search. Finally,

the somewhat curious phenomenon will be explored of how class 1 bounds can be improved for integers not divisible by integers of the form $2^i + 1$.

5. Pruning bounds (class 1). Since $a_i \le 2a_{i-1}$ for $1 \le i \le r$, it follows that if a_j and a_i are two members of an addition chain such that $a_i > 2^m a_j$, then it will require more than m steps to get from a_j to a_i no matter how the chain is constructed. This can be used as follows to prune the search tree.

The subscript, i, of an integer a_i in an addition chain for n represents the number of steps that have been taken to reach a_i. If a chain of a certain length, $lb = i + m$, is being sought for an integer n, and if $2^m a_i < n$, then it will be impossible to get to n from a_i in $m = lb - i$ steps. The branch br$[a_i]$ of the search tree emanating from a_i can be eliminated. If a chain of length 4 is being sought for $n = 16$, then the partial chain $1, 2, 3$ cannot lead to a 4-step chain for 16 since $2^2 3 < 16$. This eliminates br[3] which cuts out 12 possible paths to level 4 (or roughly half of the 25 paths to this level).

If lb has been set, then there will be $lb - i$ steps in the chain from a_i to n if such a chain exists. If $2^{lb-i} a_i < n$, then no chain for n which includes a_i exists of length lb, and the branch br$[a_i]$ can be pruned from the search tree. It follows that if $a_i < n/2^{lb-i}$, then br$[a_i]$ is pruned from the tree. This eliminates at least $lb!/i!$ paths to search.

If $a_i < \lceil n/2^{lb-i} \rceil$, there are two cases for the integer a_i. If $n/2^{lb-i}$ is an integer, then $a_i < n/2^{lb-i}$, and if it is not an integer, then $a_i < \lceil n/2^{lb-i} \rceil$ implies that $a_i \le \lfloor n/2^{lb-i} \rfloor$. Since $n/2^{lb-i}$ is not an integer, it follows that $a_i < n/2^{lb-i}$. In any event, if $a_i < \lceil n/2^{lb-i} \rceil$, then br$[a_i]$ can be pruned from the tree. This leads to the set of class 1 bounds

$$b_i = \lceil n/2^{lb-i} \rceil, \qquad i = 0, \ldots, lb. \tag{A}$$

We have the following theorem.

THEOREM 1. *Let $\{b_i\}$ denote bounding sequence* (A) *for a positive integer n. If for some step i in an addition chain for n, $a_i < b_i$, then the partial chain $a_0, a_1, \ldots, a_i$ cannot lead to a chain of length lb for n, and br$[a_i]$ can be pruned from the search tree.*

For example, suppose $n = 39 = 100111_2$. Then $lb = \lambda(39) + \lceil \log_2 \nu(39) \rceil = 5 + 2 = 7$. The set of bounds $\{\lceil 39/2^{7-i} \rceil\} = \{1, 1, 2, 3, 5, 10, 20, 39\}$. The partial chain $1, 2, 3, 5, 8, 9$ cannot lead to a chain of length 7 for 39 since $a_5 = 9 < 10 = b_5$.

At each stage in an addition chain's development, the possible choices for the next element a_{i+1} after a_i are added as a new stack segment. This includes all a_{i+1} such that $a_{i+1} = a_j + a_k$ for $0 \le k \le j \le i$ and $a_i < a_{i+1} \le n$. In the partial chain $1, 2, 3, 5, 8, 13$ for 39 the possible choices for the next element in the chain are $14, 15, 16, 18, 21$, and 26. The bound associated with the sixth step in the chain is 20. Since $18 < 20$, it can be discarded, as can $14, 15$, and 16. The numbers 21 and 26 are added to the stack.

The bounds $\{\lceil n/2^{lb-i} \rceil\}$ can be found by dividing n by 2 and each successive result by 2. At each stage the result is rounded up to the nearest integer if necessary. This follows from the fact that $\lceil \lceil n/2^{lb-i} \rceil / 2 \rceil = \lceil n/2^{lb-i+1} \rceil$. If $a_i \ge b_i = \lceil n/2^{lb-i} \rceil$, br$[a_i]$ is retained and all possibilities for a_{i+1} such that $a_i < a_{i+1} \le n$ and $a_{i+1} \ge b_{i+1} = \lceil n/2^{lb-(i+1)} \rceil$ are stored for future consideration.

6. Class 1 bounds refined. If n is not a power of 2, the bounds $\{\lceil n/2^{lb-i} \rceil\}$ can be improved by the following considerations. For an odd integer n, the last step

in an addition chain for n is $n = a_r = a_{r-1} + a_k$ for some $k < r-1$. The last step must be a star step in a minimal chain since otherwise, a_{r-1} could be eliminated, resulting in a shorter chain. Also, $k < r-1$, or else n would be even. Thus, if $r = lb$, then $n \le a_{lb-1} + a_{lb-2}$, which implies that $n \le 3a_{lb-2}$.

It follows that a_{lb-2} cannot lead to a chain of lb steps for n unless $a_{lb-2} \ge \lceil n/3 \rceil$. a_{lb-2} will not be greater than or equal to $\lceil n/3 \rceil$ unless $a_{lb-3} \ge \lceil n/(3 \cdot 2) \rceil$, $a_{lb-4} \ge \lceil n/(3 \cdot 2^2) \rceil$, etc. Thus, br$[a_i]$ is retained only when $a_i \ge \lceil n/(3 \cdot 2^{lb-(i+2)} \rceil$ for $i = 0, \ldots, lb-2$, and $a_{lb-1} \ge \lceil n/2 \rceil$. The bounds are

$$\text{(B)} \qquad b_i = \begin{cases} \lceil n/(3 \cdot 2^{lb-(i+2)}) \rceil & 0 \le i \le lb-2, \\ \lceil n/2^{lb-i} \rceil & lb-1 \le i \le lb. \end{cases}$$

For $n = 39$, the bounds are $\{1, 1, 2, 4, 7, 13, 20, 39\}$ which is a significant improvement over bounding sequence (A).

Regardless of whether n is even or odd, it can be expressed uniquely as $n = 2^t m$, where m is odd and $t \ge 0$. Bounding sequence (B) generalizes to

$$\text{(C)} \qquad b_i = \begin{cases} \lceil n/(3 \cdot 2^{lb-(i+2)} \rceil & 0 \le i \le lb-t-2, \\ \lceil n/2^{lb-i} \rceil & lb-t-1 \le i \le lb. \end{cases}$$

This leads to the following theorem.

THEOREM 2.[1] *Let $\{b_i\}$ denote bounding sequence* (C) *for a positive integer n. If for some step i in an addition chain for n, $a_i < b_i$, then the partial chain $a_0, a_1, \ldots, a_i$ cannot lead to a chain of length lb for n, and br$[a_i]$ can be pruned from the search tree.*

Proof. Suppose $n = 2^t m$ where m is odd. Bounding sequence (C) can be split into two parts.

Region 1. $n, n/2, n/2^2, \ldots, n/2^{t-1}, m = n/2^t, \lceil m/2 \rceil$.

These are the bounds in reverse order corresponding to steps i such that $lb-t-1 \le i \le lb$.

Region 2. $\lceil m/3 \rceil, \lceil m/(3 \cdot 2) \rceil, \ldots, \lceil m/(3 \cdot 2^{lb-t-(i+2)}) \rceil = \lceil n/(3 \cdot 2^{lb-(i+2)}) \rceil, \ldots$.

These are the bounds in reverse order corresponding to steps i such that $0 \le i \le lb-t-2$.

In Region 1, $b_i = \lceil n/2^{lb-i} \rceil$, while in Region 2, $b_i = \lceil n/(3 \cdot 2^{lb-(i+2)}) \rceil$. This latter bound is determined by noting that $b_{lb-t-2} = \lceil m/3 \rceil, b_{lb-t-3} = \lceil m/(3 \cdot 2) \rceil$, $\ldots, b_{lb-t-j} = \lceil m/(3 \cdot 2^{j-2}) \rceil = \lceil n/(3 \cdot 2^{t+j-2}) \rceil, \ldots$. If $i = lb-t-j$, then $t+j-2 = lb-(i+2)$.

For Region 1, if $a_i < \lceil n/2^{lb-i} \rceil$, then br$[a_i]$ can be pruned from the search tree by the same reasoning used in Theorem 1.

For Region 2, suppose $a_i < \lceil m/(3 \cdot 2^{lb-t-(i+2)}) \rceil = \lceil n/(3 \cdot 2^{lb-(i+2)}) \rceil$. Then $a_i < n/(3 \cdot 2^{lb-i-2})$. Assume n is not a power of two since, if it is a power of two, Region 2 does not apply. Then every step in a chain for n cannot be a doubling. Suppose step s is a nondoubling in which case $a_s \le a_{s-1} + a_{s-2}$. If $s > i+1$, then

$$\begin{aligned} a_s &\le a_{s-1} + a_{s-2} \le 2^{(s-1)-i} a_i + 2^{(s-2)-i} a_i \\ &= 2^{(s-2)-i}(2a_i + a_i) = 2^{(s-2)-i}(3a_i). \end{aligned}$$

[1] Note that in this theorem and following theorems, a_i will often be replaced by $2a_{i-1}$ in a set of inequalities since $a_i \le 2a_{i-1}$. More generally, for $j < i, a_i$ will be replaced by $2^{i-j}a_j$ since $a_i \le 2^{i-j}a_j$. Additional inequalities such as $4a_i + a_{i-1} \le 3(a_i + a_{i-1})$ also follow from the fact that $a_i \le 2a_{i-1}$. These and other similar inequalities will be used frequently.

It follows that $n = a_{lb} \le 2^{lb-s}a_s \le 2^{lb-s}2^{(s-2)-i}(3a_i) < (2^{lb-2-i})(3)(n/(3\cdot 2^{lb-i-2}) = n$. Of course $n < n$ is a contradiction. Thus, there can be no nondoubling step in the chain after step $i+1$. If every step in the chain after step $i+1$ is a doubling, then $n = a_{lb} = 2^{lb-(i+1)}a_{i+1}$. This means that 2^{lb-i-1} divides n. Since $i \le lb - t - 2$, it follows that $lb - i - 1 \ge t + 1$. This is a contradiction since t is the highest power of 2 dividing n.

Thus, for any a_i in Region 2 which is less than the corresponding bound, br$[a_i]$ can be pruned from the search tree. □

7. Pruning bounds (class 2). More can be done by way of pruning the search tree by considering the sum $a_i + a_{i-1}$. It is often the case that if this sum is less than b_{i+1} of bounding sequence (C), then br$[a_i]$ can be pruned from the search tree. Bounding sequences for $a_i + a_{i-1}$ will be called class 2 bounds. (Note: The same bounding sequence, when used for a_i, is called a class 1 bounding sequence and, when used for $a_i + a_{i-1}$, is called a class 2 bounding sequence.) It is a somewhat curious fact that if n is a multiple of 5, then bounding sequence (C) must be replaced by bounding sequence (A) when used as a class 2 bound. First, the case for odd n will be considered. Bounding sequence (B) is used when n is odd, and n is not a multiple of 5. It is convenient to split it into three regions.

Region 1. $n, \lceil n/2 \rceil$.

Region 2. $\lceil n/3 \rceil$.

Region 3. $\lceil n/(3\cdot 2) \rceil, \ldots, \lceil n/(3\cdot 2^{lb-(i+2)}) \rceil, \ldots$.

In Region 1, b_{i+1} is either n or $\lceil n/2 \rceil$; that is, $i = lb - 1$ or $i = lb - 2$.

$\boldsymbol{i = lb-1}$. In this case, $a_i + a_{i-1} < b_{i+1}$ implies that $a_{lb-1} + a_{lb-2} < b_{lb} = a_{lb} = n$. Thus, n must be $a_{lb-1} + a_{lb-1} = 2a_{lb-1}$. This contradicts the fact that n is odd.

$\boldsymbol{i = lb - 2}$. If $a_{lb-2} + a_{lb-3} < b_{lb-1} = \lceil n/2 \rceil$, then $a_{lb-2} + a_{lb-3} < n/2$. Since n is odd, $n \le a_{lb-1} + a_{lb-2} \le 2a_{lb-2} + 2a_{lb-3} < 2(n/2) = n$ which is a contradiction.

For Region 2, $i = lb - 3$. If $a_{lb-3} + a_{lb-4} < b_{lb-2} = \lceil n/3 \rceil$, then $a_{lb-3} + a_{lb-4} < n/3$. We assume that $a_i \ge b_i$ for $i = 1, \ldots, n$, since it has been shown previously (Theorem 2) that if $a_i < b_i$ for any i, then a chain of lb steps for n is not possible. This means that $a_{lb-2} \ge n/3$. Since $a_{lb-3} + a_{lb-4} < n/3$, it follows that $a_{lb-2} = 2a_{lb-3}$. Step $lb - 1$ must be a star step since step $lb - 2$ is a doubling. The possibilities for a_{lb-1} need to be considered. These can be drawn from the set $\{a_{lb-2} + a_j, j \le lb - 2\}$. In what follows, $n \le a_{lb-1} + a_{lb-2}$ since n is odd, and $a_{lb-2} = 2a_{lb-3}$.

(i) $a_{lb-1} = a_{lb-2} + a_{lb-4}$:

$$n = a_{lb} \le a_{lb-1} + a_{lb-2} = 4a_{lb-3} + a_{lb-4} \le 3(a_{lb-3} + a_{lb-4}) < 3(n/3) = n.$$

Clearly, no possibilities $a_{lb-2} + a_j, j < lb - 4$ need be considered.

(ii) $a_{lb-1} = a_{lb-2} + a_{lb-3}$:

Since n is odd and step lb is a star step, the possibilities for step lb can be drawn from the set $\{a_{lb-1} + a_j, j \le lb - 2\}$.

(iia) If $n = a_{lb-1} + a_{lb-2}$, then $n = 5a_{lb-3}$. Thus, 5 divides n. If $n = 95 = 1011111_2$, then $\lambda(95) = 6$, and $\nu(95) = 6$. $lb = \lambda(95) + \lceil \log_2 \nu(95) \rceil = 9$. The bounding sequence $\{b_i\}$ is $\{1, 1, 1, 2, 4, 8, 16, 32, 48, 95\}$. An addition chain for 95 is $1, 2, 3, 5, 8, 11, 19, 38, 76, 95$. Note that $a_{lb-3} + a_{lb-4} = 19 + 11 = 30 < 32 = b_{lb-2}$. Thus, when n is divisible by 5, there often exist minimal addition chains for n when $a_{lb-3} + a_{lb-4} < b_{lb-2}$.

(iib) If $n = a_{lb-1} + a_{lb-3}$, this leads to the contradiction that $n < n$ by similar reasoning as that used in (i). Again no possibilities $n = a_{lb-1} + a_j, j < lb - 3$ need be considered.

(iii) $a_{lb-1} = a_{lb-2} + a_{lb-2}$:
If $n = a_{lb-1} + a_{lb-2}$, then $n = 3(2a_{lb-3})$ is even, and if $n = a_{lb-1} + a_j, j < lb-3$, then $n < n$ as before. Since lb must be a star step, the only possibility is $n = a_{lb-1} + a_{lb-3}$ in which case $n = 5a_{lb-3}$. n is divisible by 5 and, as in case (iia), minimal chains for n often exist when $a_{lb-3} + a_{lb-4} < b_{lb-2}$.

For Region 3, $i \le lb-4$. If $a_i + a_{i-1} < b_{i+1} = \lceil n/(2^{lb-i-3} \cdot 3) \rceil$, then $a_i + a_{i-1} < n/(2^{lb-i-3} \cdot 3)$. It follows that

$$\begin{aligned} a_{lb-3} + a_{lb-4} &< 2^{(lb-3)-i} a_i + 2^{(lb-4)-(i-1)} a_{i-1} \\ &= 2^{(lb-3)-i}(a_i + a_{i-1}) < n/3. \end{aligned}$$

It follows from the same arguments as used for Region 2 that $\mathrm{br}[a_i]$ will be pruned from the search tree unless n is a multiple of 5.

These considerations establish the following theorem.

THEOREM 3. *Let n be odd, n not a multiple of 5, and let $\{b_i\}$ be bounding sequence* (B). *If $a_i + a_{i-1} < b_{i+1}$ for some i, then the partial chain $1 = a_0, a_1, \ldots, a_i$ cannot lead to a minimal addition chain for n.*

Comment. From Theorem 1, it is known that $a_i \ge b_i$ if $1 = a_0, a_1, \ldots, a_i$ is to lead to a minimal chain for n. The following example shows that additional branches can be pruned from the search tree when $a_i + a_{i-1} < b_{i+1}$.

For $n = 39, \{b_i\} = \{1, 1, 2, 4, 7, 13, 20, 39\}$. A partial chain $1, 2, 3, 5, 7$ satisfies the requirement that $a_i \ge b_i$. However, $a_4 + a_3 = 7 + 5 < 13 = b_5$. Thus, $\mathrm{br}[7] = \mathrm{br}[a_4]$ can be pruned from the search tree.

A generalization. Theorem 3 generalizes to the case where $n = 2^t m$ and m is odd. In this case, bounding sequence (C) is used, and it is convenient to split it into the following regions.

Region 1. $n, n/2, n/2^2, \ldots, n/2^{t-1}, m = n/2^t, \lceil m/2 \rceil$.

Region 2. $\lceil m/3 \rceil$.

Region 3. $\lceil m/(3 \cdot 2) \rceil, \ldots, \lceil m/(3 \cdot 2^{lb-t-(i+2)} \rceil = \lceil n/(3 \cdot 2^{lb-(i+2)}) \rceil, \ldots$.

THEOREM 4. *Suppose n is not a multiple of 5 and $\{b_i\}$ is bounding sequence* (C). *If $a_i + a_{i-1} < b_{i+1}$ for some i and $n \ne 2^{lb-i} a_i$ for i in Region 1, then the partial chain $1 = a_0, a_1, \ldots, a_i$ cannot lead to a minimal addition chain for n.*

Note. The condition that $n \ne 2^{lb-i} a_i$ for i in Region 1 is necessary as can be seen from the following example. $1, 2, 3, 5, 7, 14, 21, 42, 84, 168, 336$ is a minimal chain for $n = 336$. Bounding sequence (C) for 336 is $1, 1, 2, 4, 7, 11, 21, 42, 84, 168, 336$. Even though $a_6 + a_5 = 21 + 14 < 42 = b_7, \mathrm{br}[a_6]$ cannot be pruned from the search tree.

The proof of Theorem 4, while tedious, is analogous to the proof of Theorem 3, with $lb - t - i$ replacing $lb - i, i \ge 0$. Also, some step $s \ge lb - t$ will need to be a nondoubling or else 2^{t+1} will divide n, which contradicts the fact that 2^t is the highest power of 2 dividing n. The element a_s in Theorem 4 plays a role similar to that of n in Theorem 3 since they are both formed by nondoublings.

Proof.

Region 1. $lb - t - 2 \le i \le lb - 1$. Note that if $i \ge lb - t - 2$, then $i + 1 \ge lb - t - 1$, and b_{i+1} will be in Region 1. If $a_i + a_{i-1} < b_{i+1} = \lceil n/2^{lb-(i+1)} \rceil$, then suppose there exists a step $s > i$ that is not a doubling. Then

$$a_s \le a_{s-1} + a_{s-2} \le 2^{(s-1)-i} a_i + 2^{(s-2)-(i-1)} a_{i-1} = 2^{(s-1)-i}(a_i + a_{i-1}).$$

It follows that

$$n = a_{lb} \le 2^{lb-s} a_s \le 2^{lb-s} 2^{(s-1)-i}(a_i + a_{i-1}) < 2^{(lb-1)-i}(n/2^{(lb-1)-i}) = n.$$

Thus, every step after step i must be a doubling. This means that $n = 2^{lb-i}a_i$ if the partial chain $1 = a_0, a_1, \ldots, a_i$ can be extended to a chain of length lb for n.

Region 2. $i = lb - t - 3$ for $b_{i+1} = \lceil m/3 \rceil$.

If $a_{lb-t-3} + a_{lb-t-4} < \lceil m/3 \rceil = \lceil n/(2^t \cdot 3) \rceil$, then $a_{lb-t-3} + a_{lb-t-4} < n/(2^t 3)$. Since it is assumed that $a_i \geq b_i$ for $i = 1, \ldots, n$, then $a_{lb-t-2} \geq b_{lb-t-2} = \lceil n/(2^t \cdot 3) \rceil \geq n/(2^t 3)$, and it follows that $a_{lb-t-2} = 2a_{lb-t-3}$. Since step $lb-t-2$ is a doubling, step $lb-t-1$ must be a star step; that is, $a_{lb-t-1} = a_{lb-t-2} + a_k$ for some $k \leq lb-t-2$. It will be shown that $k = lb-t-2$ or $k = lb-t-3$. Suppose that $k \leq lb-t-4$ and that there is a nondoubling after step $lb-t-1$. Let s be the first such step. Then $a_{lb-t-1} \leq a_{lb-t-2} + a_{lb-t-4}$ and by using inequalities as noted in footnote 1 of Theorem 2, it follows that

$$\begin{aligned} a_s &\leq a_{s-1} + a_{s-2} \leq 2^{(s-1)-(lb-t-1)}a_{lb-t-1} + 2^{(s-2)-(lb-t-2)}a_{lb-t-2} \\ &= 2^{s-lb+t}(a_{lb-t-1} + a_{lb-t-2}) \leq 2^{s-lb+t}(2a_{lb-t-2} + a_{lb-t-4}) \\ &< 2^{s-lb+t}3(n/(2^t 3)) = 2^{s-lb}n. \end{aligned}$$

Thus, $n = a_{lb} \leq 2^{lb-s}a_s < 2^{lb-s}2^{s-lb}n = n$. This means that all steps after step $lb-t-1$ are doublings which, as noted, is not possible. Thus, either $a_{lb-t-1} = a_{lb-t-2} + a_{lb-t-3}$ or $a_{lb-t-1} = a_{lb-t-2} + a_{lb-t-2}$.

So far, if $a_{lb-t-3} + a_{lb-t-4} < n/(2^t 3)$, then it has been established that:

(1) $a_{lb-t-2} = 2a_{lb-t-3}$; and

(2) $a_{lb-t-1} = a_{lb-t-2} + a_{lb-t-3}$ or $a_{lb-t-1} = a_{lb-t-2} + a_{lb-t-2}$.

Suppose $\boldsymbol{a_{lb-t-1} = a_{lb-t-2} + a_{lb-t-3}}$. Then $a_{lb-t-1} = 3a_{lb-t-3}$. There must be a step $s \geq lb-t$ that is a nondoubling or as shown before 2^{t+1} divides n. Let s be the first nondoubling after step $lb-t-1$. The possibilities for step s that have a chance can be drawn from the set $\{a_{s-1} + a_{s-2}, a_{s-1} + a_{s-3}, 2a_{s-2}\}$.

(i) If $a_s = a_{s-1} + a_{s-3}$, then $a_s \leq 2^{(s-1)-(lb-t-1)}a_{lb-t-1} + 2^{(s-3)-(lb-t-3)}a_{lb-t-3}$. It follows by inequalities, as noted in footnote 1 that

$$\begin{aligned} a_s &\leq 2^{s-lb+t}(a_{lb-t-1} + a_{lb-t-3}) = 2^{s-lb+t}(4a_{lb-t-3}) \\ &< 2^{s-lb+t}3(a_{lb-t-3} + a_{lb-t-4}) < 2^{s-lb+t}3(n/(2^t 3)) = 2^{s-lb}n. \end{aligned}$$

Thus, $n = a_{lb} \leq 2^{lb-s}a_s < 2^{lb-s}2^{s-lb}n = n$.

(ii) If $a_s = a_{s-1} - a_{s-2}$, then if $\boldsymbol{s = lb - t}$, it follows that $a_{lb-t} = a_{lb-t-1} + a_{lb-t-2}$. Since $a_{lb-t-1} = a_{lb-t-2} + a_{lb-t-3}$ and $a_{lb-t-2} = 2a_{lb-t-3}$, this means that $a_{lb-t} = 5a_{lb-t-3}$. If there is a step $h > lb-t$ which is a nondoubling, then by reasoning similar to that discussed before, it can be shown that $n < n$. This means that all the steps after step $lb-t$ must be doublings. Thus

$$n = a_{lb} = 2^{lb-(lb-t)}a_{lb-t} = 2^t(5a_{lb-t-3}).$$

This contradicts the fact that n is not a multiple of 5.

Now, suppose $\boldsymbol{s > lb - t}$. This means that $a_{lb-t} = 2a_{lb-t-1}$, since step s is the first nondoubling after step $lb-t-1$. The reasoning used before concerning nondoubling steps can be combined with the fact that $a_{lb-t-3} + a_{lb-t-4} < n/(2^t 3)$ to show that $a_s < 2^{(s-lb+t)}3(n/(2^t 3)) = 2^{(s-lb)}n$, and $n = a_{lb} \leq 2^{lb-s}a_s < n$.

(iii) If $a_s = 2a_{s-2}$, then $a_s \leq 2(2^{(s-2)-(lb-t-2)}a_{lb-t-2}) = 2^{s-lb+t+1}a_{lb-t-2} = 2^{s-lb+t}(4a_{lb-t-3})$. As in case (i), this results in $n < n$.

Suppose $\boldsymbol{a_{lb-t-1} = 2a_{lb-t-2}}$. Since $a_{lb-t-2} = 2a_{lb-t-3}$, it follows that $a_{lb-t-1} = 4a_{lb-t-3}$. There must be a first step $s \geq lb-t$ that is a nondoubling or else 2^{t+1}

divides n as before. The possibilities for step s are $\{a_{s-1}+a_{s-2}, a_{s-1}+a_{s-3}, a_{s-1}+a_{s-4}, 2a_{s-2}\}$. It will become evident that these are the only cases that need to be considered.

(i) $a_s = a_{s-1}+a_{s-4} \le 2^{(s-1)-(lb-t-1)}a_{lb-t-1} + 2^{(s-4)-(lb-t-4)}a_{lb-t-4}$
$= 2^{s-lb+t}(4a_{lb-t-3}+a_{lb-t-4}) \le 2^{s-lb+t}3(a_{lb-t-3}+a_{lb-t-4})$
$< 2^{s-lb+t}3(n/(2^t3))$.

Thus, $a_s < 2^{s-lb}n$, and $n = a_{lb} \le 2^{lb-s}a_s < 2^{lb-s}2^{s-lb}n = n$.

(ii) $a_s = 2a_{s-2}$ leads to $n < n$ as in (iii) of the case where $a_{lb-t-1} = a_{lb-t-2} + a_{lb-t-3}$.

(iii) $a_s = a_{s-1}+a_{s-2}$. This is the most difficult case, since this is the largest possible value of a_s for the nondoubling step which must occur somewhere between steps $lb-t$ and lb. In what follows, $a_{lb-t-1} = 2a_{lb-t-2}$ and $a_{lb-t-2} = 2a_{lb-t-3}$. Suppose $\boldsymbol{s = lb-t}$. Then $a_{lb-t} = a_{lb-t-1}+a_{lb-t-2} = 6a_{lb-t-3}$. If there are no more nondoublings in the chain after step $lb-t$, then $n = a_{lb} = 2^{lb-(lb-t)}a_{lb-t} = 2^t6a_{lb-t-3}$ which means 2^{t+1} divides n. Thus, there must be a first nondoubling step $h > s$. The possibilities for a_h are drawn from the set $\{a_{h-1}+a_{h-2}, a_{h-1}+a_{h-3}, 2a_{h-2}\}$.

(iiia) $a_h = a_{h-1}+a_{h-3} \le 2^{(h-1)-(lb-t)}a_{lb-t} + 2^{(h-3)-(lb-t-2)}a_{lb-t-2}$
$= 2^{h-1-lb+t}6a_{lb-t-3} + 2^{h-1-lb+t}2a_{lb-t-3} = 2^{h-lb+t}4a_{lb-t-3}$.

From this it follows, as in prior cases, that $n < n$.

(iiib) $a_h = 2a_{h-2} \le 2(2^{(h-2)-(lb-t-1)}a_{lb-t-1} = 2^{h-lb+t}4a_{lb-t-3}$ which leads to $n < n$.

(iiic) $a_h = a_{h-1}+a_{h-2}$:

$$a_h = a_{h-1}+a_{h-2} \le 2^{(h-1)-(lb-t)}a_{lb-t} + 2^{(h-2)-(lb-t-1)}a_{lb-t-1}$$
$$= 2^{h-1-lb+t}6a_{lb-t-3} + 2^{h-1-lb+t}4a_{lb-t-3} = 2^{h-lb+t}5a_{lb-t-3}.$$

The 5 in the bound is not sufficient to conclude that $n < n$. If there is another step $l > h$ which is a nondoubling, then, as before, it can be shown that $a_l \le 2^{l-lb+t}4a_{lb-t-3}$ from which it follows that $n < n$. It follows that $n = a_{lb} = 2^{lb-h}a_h = 2^{lb-h}(a_{h-1}+a_{h-2})$. If $h > lb-t+1$, then

$$n = 2^{lb-h}(a_{h-1}+a_{h-2}) \le 2^{lb-h}(2^{(h-1)-(lb-t)}a_{lb-t} + 2^{(h-2)-(lb-t)}a_{lb-t})$$
$$= 2^{lb-h}(2^{h-1-lb+t}6a_{lb-t-3} + 2^{h-2-lb+t}6a_{lb-t-3})$$
$$= 2^{t-1}9a_{lb-t-3} \le 2^{t-1}6(a_{lb-t-3}+a_{lb-t-4}) < 2^t3(n/(2^t3)) = n.$$

If $h = lb-t+1$, then $n = 2^{lb-(lb-t+1)}(a_{lb-t}+a_{lb-t-1}) = 2^{t-1}10a_{lb-t-3}$.

From this it follows that 5 divides n.

Now we continue with the case $a_{lb-t-1} = 2a_{lb-t-2}$, where step s is the first nondoubling after step $lb-t-1$ and $a_s = a_{s-1}+a_{s-2}$. Suppose $\boldsymbol{s > lb-t}$; then step $s-1$ is a doubling. If there are no more nondoublings after step s, then

$$n = 2^{lb-s}a_s = 2^{lb-2}(a_{s-1}+a_{s-2}) = s^{lb-s}(3a_{s-2})$$
$$= 2^{lb-s}3(2^{(s-2)-(lb-t-1)}a_{lb-t-1}) = 2^{t-1}3a_{lb-t-1} = 2^{t-1}3(4a_{lb-t-3}).$$

This implies that 2^{t+1} divides n, which is a contradiction. As in the previous case, let step h be the first nondoubling step after step s. The only possibility for a_h that does not lead to $n < n$ is $a_h = a_{h-1}+a_{h-2}$. As in case (iiic), it can be shown that there can be no more nondoublings after step h or else $n < n$. Thus, $n = 2^{lb-h}a_h = 2^{lb-h}(a_{h-1}+a_{h-2})$. The cases $h = s+1$ and $h > s+1$ are handled separately.

Since $s > lb - t$, step $lb - t$ is a doubling, and $a_{lb-t} = 8a_{lb-t-3}$.

If $h = s + 1$, it can be shown by reasoning similar to that used before that $n = 2^t 5 a_{lb-t-3}$, which means that 5 divides n while if $h > s + 1$; then

$$\begin{aligned} n &= 2^{lb-h}a_h = 2^{lb-h}(a_{h-1} + a_{h-2}) = 2^{lb-h}(2^{h-1-s}a_s + 2^{h-2-s}a_s) \\ &= 2^{lb-h}2^{h-2-s}(3a_s) = 2^{lb-2-s}3(a_{s-1} + a_{s-2}) \\ &= 2^{lb-2-s}3(2^{(s-1)-(lb-t)}a_{lb-t} + 2^{(s-2)-(lb-t-1)}a_{lb-t-1}) \\ &= 3(2^{t-3}12a_{lb-t-3}) \le 3(2^{t-3}8(a_{lb-t-3} + a_{lb-t-4})) < 2^t 3(n/(2^t 3)) = n. \end{aligned}$$

(iv) $a_s = a_{s-1} + a_{s-3}$. As before, it can be shown that if there is a step $h > s$ that is a nondoubling, then $n < n$. Thus, $n = 2^{lb-s}a_s = 2^{lb-s}(a_{s-1} + a_{s-3})$. The cases $s = lb - t$, $s = lb - t + 1$, $s = lb - t + 2$, and $s \ge lb - t + 3$ must be considered. In all cases it can be shown that 5 divides n. It is important to note that s is the first step after $lb - t - 1$ that is a nondoubling. For instance, if $s = lb - t + 2$, then

$$\begin{aligned} n &= 2^{lb-s}(a_{s-1} + a_{s-3}) = 2^{lb-(lb-t+2)}(a_{lb-t+1} + a_{lb-t-1}) \\ &= 2^{t-2}(16a_{lb-t-3} + 4a_{lb-t-3}) = 2^t(5a_{lb-t-3}). \end{aligned}$$

Region 3. $i < lb - t - 3$.

For this region, the bound $a_i + a_{i-1} < b_{i+1}$ translates into $a_i + a_{i-1} < \lceil m/(3 \cdot 2^{lb-t-(i+3)} \rceil = \lceil n/(3 \cdot 2^{lb-(i+3)}) \rceil$ which implies $a_i + a_{i-1} < n/(2^{lb-i-3}3)$. It follows that

$$\begin{aligned} a_{lb-t-3} + a_{lb-t-4} &< 2^{(lb-t-3)-i}a_i + 2^{(lb-t-4)-(i-1)}a_{i-1} \\ &= 2^{(lb-t-3)-i}(a_i + a_{i-1}) < 2^{(lb-t-3)-i}n/(2^{lb-i-3}3) = n/(2^t 3). \end{aligned}$$

It follows from the same arguments as used for Region 2 that br$[a_i]$ will be pruned from the search tree unless n is a multiple of 5. □

It has been shown that if $n = 2^t m$, m odd, $\{b_i\}$ is bounding sequence (C), and $a_i < b_i$ for some i, then br$[a_i]$ can be pruned from the search tree and, furthermore, if n is not a multiple of 5, and $a_i + a_{i-1} < b_{i+1}$, then br$[a_i]$ can be pruned from the search tree provided that $n \ne 2^{lb-i}a_i$ for some step i in Region 1. If n is a multiple of 5 and if $\{b_i\}$ is bounding sequence (A), then if $a_i + a_{i-1} < b_{i+1} = n/2^{lb-(i+1)}$ for some i, br$[a_i]$ can be pruned from the search tree if there exists a step $s > i$ that is not a doubling since $a_s \le a_{s-1} + a_{s-2} \le 2^{(s-1)-i}a_i + 2^{(s-2)-(i-1)}a_{i-1} = 2^{(s-1)-i}(a_i + a_{i-1})$ from which it follows that $n < n$. Thus, when n is a multiple of 5, Theorem 4 holds for n provided that bounding sequence (C) is replaced by bounding sequence (A).

8. Pruning bounds applied. When the sequence $\{b_i\}$ is used as a class 1 bounding sequence, it will be called a *vertical* bounding sequence, and when it is used as a class 2 bounding sequence, it will be called a *slant* bounding sequence. If n is not a multiple of 5, then bounding sequence (C) can be used for both the vertical and slant bounds. The algorithm for generating addition chains checks to see: (1) if $a_i \ge b_i$; if this condition is satisfied, it next checks: (2) to see that if $n \ne 2^{lb-i}a_i$ for i in Region 1; then is $a_i + a_{i-1} \ge b_{i+1}$. If these conditions are met, then the partial chain $1 = a_0, a_1, \ldots, a_i$ is not rejected and candidates for a_{i+1} are considered. If (1) fails, then br$[a_i]$ is pruned from the search tree, and if (1) passes but (2) fails, br$[a_i]$ is pruned from the search tree.

If n is a multiple of 5, the same strategy is employed; however, bounding sequence (C) is used for the vertical bounds while bounding sequence (A) is used for the slant bounds.

TABLE 1

Value of n						
Test	23	127	191	568	607	1,015[2]
no bds	293	1,890,353 time: 20.9	24,383,588 time: 327.4			
V (A)	193	330,791 time: 1.6	2,805,121 time: 15.1	1,632,557 time: 8.57	63,548,562 time: 372.4	5,901,139 time: 32.0
V (C)	153	208,318 time: 1.0	1,789,703 time: 9.8	1,587,969 time: 8.18	31,649,325 time: 192.1	2,146,835 time: 12.1
V/S (A)	155	241,396 time: 1.0	2,025,178 time: 9.6	964,319 time: 4.39	40,847,534 time: 203.3	3,218,491 time: 14.6
V/S (C)	137	174,788 time: 0.82	1,482,328 time: 7.5	954,573 time: 4.28	24,823,053 time: 139.8	1,900,148 time: 9.9

The effect of these pruning bounds can be measured in terms of time required to search for chains as well as in terms of the number of "pops" off the stack. In the latter case, what is being popped off the stack for a given step i is the next candidate for a_i. The pruning bounds eliminate certain branches of the search tree, and the number of "pops" will diminish as the pruning bounds are put into effect. Several cases are considered for given values of n. The first case (no bds) has no pruning bounds. Case V (A) implements vertical pruning bounds using bounding sequence (A) while case V (C) implements vertical bounds using bounding sequence (C), etc. Case V/S (A) shows the effect of using bounding sequence (A) for both vertical and slant bounds. Case V/S (C) uses bounding sequence (C) for both vertical and slant bounds and when $n \equiv 0 \pmod 5$ replaces bounding sequence (C) with bounding sequence (A) for the slant bounds. Table 1 shows results for $n = 23$, 127, 191, 568, 607, and 1,015. These are interesting numbers from the standpoint of not giving trivial values yet not requiring so much time as to be unreasonable. The times, of course, are relative to the computer being used and are significant primarily in how they relate to each other as opposed to their actual values. It should be noted that when n is odd, bounding sequence (C) reduces to bounding sequence (B).

Table 1 records the number of "pops" that occur in an algorithm that finds all addition chains of minimal length for the given values of n. Also recorded is the time in seconds taken by each search. The times for $n = 23$ are too small to be significant.

A good check that the bounding sequences do not cut too much from the search tree is the observation that NMC(n) remains the same in all cases. As can be seen, there is significant improvement in going from V(A) to V/S(C).

9. Pruning bounds (class 1 further refined); the $2^i + 1$ phenomenon. For n odd, bounding sequence (B) can be improved as a vertical (class 1) bounding sequence depending on the multiplicative nature of n. If n is not a multiple of an integer of the form $2^i + 1$, then the following sequence can be used for the vertical bounding sequence.

$$\text{(D)} \qquad b_i = \begin{cases} \lceil n/(2^{lb-i-1}+1) \rceil & 0 \le i \le lb-1, \\ n & i = lb. \end{cases}$$

Suppose $a_i < \lceil n/(2^{lb-i-1}+1) \rceil$. Then $a_i < n/(2^{lb-i-1}+1)$. The cases $i = lb-1$ and $i = lb-2$ have been established previously. Let $i = lb-k$ for some k such that

[2]For $n = 1{,}015$, in the V/S (C) case bounding sequence (A) is used for the slant bounds since n is divisible by 5.

$3 \le k \le lb$.

$\boldsymbol{k = 3}$. $a_{lb-3} < n/5$. Since n is odd, step lb is not doubling. Yet it must be a star step. If $n = a_{lb} = a_{lb-1} + a_{lb-3}$, then $n \le 5a_{lb-3} < n$. If $n = a_{lb} = a_{lb-1} + a_{lb-2}$, then if $a_{lb-1} = 2a_{lb-2}$, it follows that 3 divides n, and if $a_{lb-1} = a_{lb-2} + a_{lb-3}$, then $n = a_{lb} = 2a_{lb-2} + a_{lb-3} \le 5a_{lb-3} < n$.

$\boldsymbol{k \ge 4}$. $a_{lb-k} < n/(2^{k-1}+1)$. If $n = a_{lb} = \boldsymbol{a_{lb-1} + a_{lb-k}}$, then

$$n = a_{lb} = a_{lb-1} + a_{lb-k} \le 2^{(lb-1)-(lb-k)} a_{lb-k} + a_{lb-k} = (2^{k-1}+1)a_{lb-k} < n.$$

Suppose $n = a_{lb} = \boldsymbol{a_{lb-1} + a_{lb-j}}$ for some $j, 2 \le j < k$. If $a_{lb-1} = 2^{j-1}a_{lb-j}$, then $2^{j-1}+1$ divides n. If $a_{lb-1} \ne 2^{j-1}a_{lb-j}$, then let s be the first integer greater than or equal to 1 such that step $lb - s$ is a nondoubling. Then $1 \le s < j$ and $n = a_{lb} = a_{lb-1} + a_{lb-j} = 2^{s-1}a_{lb-s} + a_{lb-j}$. This implies that

$$\begin{aligned} n &= 2^{s-1}a_{lb-s} + a_{lb-j} \le 2^{s-1}(a_{lb-s-1} + a_{lb-s-2}) + 2^{lb-j-(lb-k)}a_{lb-k} \\ &\le 2^{s-1}(2^{k-s-1}a_{lb-k} + 2^{k-s-2}a_{lb-k}) + 2^{k-j}a_{lb-k} \\ &= (2^{k-2} + 2^{k-3} + 2^{k-j})a_{lb-k}. \qquad \text{(Note: } s+2 \le k\text{)}. \end{aligned}$$

If $j \ge 3$, then $2^{k-2} + 2^{k-3} + 2^{k-j} < 2^{k-1} + 1$, and $n < n$. The only possibility is $j = 2$.

Consider $n = a_{lb} = \boldsymbol{a_{lb-1} + a_{lb-2}}$. Since $1 \le s < 2$, it follows that $s = 1$, and $a_{lb-1} \le a_{lb-2} + a_{lb-3}$. There are two cases for step $lb - 1$.

(i) $a_{lb-1} = a_{lb-2} + a_{lb-h}$. (Note: $h < k$. If $h = k$, then $n < n$.)

If $h = 2$, then 3 divides n. Thus, $a_{lb} = a_{lb-1} + a_{lb-2} = 2a_{lb-2} + a_{lb-h}$, where $h \ge 3$. Suppose there is a nondoubling between step $lb - h$ and step $lb - 2$. Suppose $lb - p$ is the first nondoubling going back in the chain from step $lb - 2$. Then $2 \le p < h$.

$$\begin{aligned} a_{lb} &= a_{lb-1} + a_{lb-2} = 2a_{lb-2} + a_{lb-h} = 2(2^{p-2}a_{lb-p}) + a_{lb-h} \\ &\le 2(2^{p-2}(a_{lb-p-1} + a_{lb-p-2})) + a_{lb-h}. \\ &\le 2^{p-1}(2^{k-p-1}a_{lb-k} + 2^{k-p-2}a_{lb-k}) + 2^{k-h}a_{lb-k} \\ &= (2^{k-2}a_{lb-k} + 2^{k-3}a_{lb-k}) + 2^{k-h}a_{lb-k} \\ &= (2^{k-2} + 2^{k-3} + 2^{k-h})a_{lb-k} < n \quad \text{since } h \ge 3. \end{aligned}$$

Thus, $a_{lb-2} = 2^{h-2}a_{lb-h}$ and $a_{lb} = a_{lb-1} + a_{lb-2} = 2a_{lb-2} + a_{lb-h} = 2(2^{h-2}a_{lb-h}) + a_{lb-h} = (2^{h-1}+1)a_{lb-h}$. This implies that $2^{h-1}+1$ divides n.

(ii) $a_{lb-1} = 2a_{lb-3}$. $n = a_{lb} = a_{lb-1} + a_{lb-2} = a_{lb-2} + 2a_{lb-3}$.

Then $n = a_{lb-2} + 2a_{lb-3} \le 2^{k-2}a_{lb-k} + 2(2^{k-3}a_{lb-k}) = 2^{k-1}a_{lb-k} < 2^{k-1}(n/(2^{k-1} + 1)) < n$.

This establishes the result that if n is not divisible by an integer of the form $2^i + 1, i \ge 0$, then the sequence (D) is a valid class 1 bounding sequence when generating addition chains for an integer n.

THEOREM 5. *If $2^j+1, j \ge 0$ does not divide n and $\{b_i\}$ denotes bounding sequence* (D), *then if $a_i < b_i$ for some i, $\mathrm{br}[a_i]$ can be pruned from the search tree.*

It appears that this result can generalized for $n = 2^t m$ (m odd) using the following bounding sequence.

$$\text{(E)} \qquad b_i = \begin{cases} \lceil n/2^t(2^{lb-t-(i+1)} + 1) \rceil & 0 \le i \le lb - t - 2, \\ \lceil n/2^{lb-i} \rceil & lb - t - 1 \le i \le lb. \end{cases}$$

Theorem 5 then would be restated substituting $j > 0$ for $j \ge 0$ and bounding sequence (E) for bounding sequence (D).

TABLE 2

Test	Value of n					
	23	127	191	568	607	1015[3]
no bds	293	1,890,353 time: 20.9	24,383,588 time: 327.4			
V (A)	193	330,791 time: 1.6	2,805,121 time: 15.1	1,632,557 time: 8.57	63,548,562 time: 372.4	5,901,139 time: 32.0
V (C)	153	208,318 time: 1.0	1,789,703 time: 9.8	1,587,696 time: 8.18	31,649,325 time: 192.1	2,146,835 time: 12.1
V (D)	153	186,506 time: 0.9	1,566,167 time: 8.95		25,381,280 time: 164.4	
VBEST	94	170,294 time: 0.9	1,440,524 time: 8.84	1,245,559 time: 8.29	23,316,309 time: 162.1	1,216,206 time: 8.24
V/S (A)	155	241,396 time: 1.0	2,025,178 time: 9.6	964,319 time: 4.39	40,847,534 time: 203.3	3,218,491 time: 14.6
V/S (C)	137	174,788 time: 0.82	1,482,328 time: 7.5	954,573 time: 4.28	24,823,053 time: 139.8	1,900,148 time: 9.9
V(D)/S (C)	137	166,120 time: 0.82	1,384,000 time: 7.36		22,486,525 time: 134.4	
VBEST/S (C)	92	150,522 time: 0.82	1,262,492 time: 7.09	792,751 time: 4.28	20,485,705 time: 130.4	1,087,144 time: 6.9

The numbers $23, 127, 191$, and 607 in Table 1 satisfy the hypothesis of Theorem 5. Table 2 shows results of using bounding sequence (D) for the vertical bounding sequence. Certain rows of the table use what is called VBEST for the vertical bounding sequence and bounding sequence (C) for slant bounds. VBEST is a bounding sequence that is obtained à posteriori by running the program and generating all minimal addition chains for n. For each i, b_i is the minimal a_i found among all the minimal addition chains for n. It is the best possible vertical bounding sequence, since lowering any b_i will cut out some of the minimal chains for n. In practice it is not a good bounding sequence since it cannot be determined until all the minimal addition chains have been found, but it is a good measure by which to judge the other vertical bounding sequences. As previously noted, bounding sequence (C) reduces to bounding sequence (B) when n is odd.

The V(D)/S (C) bounding sequence compares favorably with the VBEST/S (C) bounding sequence, and in all cases there is significant improvement over using just V(A) which is the obvious bounding sequence to try first.

While vertical bounding sequence (D) leads to a significant improvement in pruning the search tree, this sequence can only be used when n is not a multiple of an integer of the form $2^i + 1$. In other words, if n is a multiple of an integer of the form 2^i+1, then less aggressive pruning bounds must be used. More branches on the search tree must be explored. The search for minimal addition chains for such numbers is, in a sense, less efficient. Primes, on the other hand, (unless they are Fermat primes) can use bounding sequence (D) which results in a more efficient search.

A summary of the bounding sequences is included in Table 3.

10. Some test results. Algorithms for generating minimal length addition chains are significant from the standpoint of what their implementation in a computer program and the subsequent generation of data reveal about the addition chain

[3]For $n = 1015$, in the V/S (C) and VBEST/S(C) cases bounding sequence (A) is used for the slant bounds since n is divisible by 5. Also, sequence (D) cannot be used for vertical bounds since n is a multiple of 5.

TABLE 3

Bounding sequence	Sequence	Comments
A	$b_i = \lceil n/2^{lb-i} \rceil, i = 0, \ldots, lb$	vertical/slant bounds
B	$b_i = \begin{cases} \lceil n/(3 \cdot 2^{lb-(i+2)}) \rceil, & 0 \le i \le lb-2 \\ \lceil n/2^{lb-i} \rceil, & lb-1 \le i \le lb \end{cases}$	n odd, vertical bounds, slant bounds for $n \neq 5k$
C	$b_i = \begin{cases} \lceil n/(3 \cdot 2^{lb-(i+2)}) \rceil, & 0 \le i \le lb-t-2 \\ \lceil n/2^{lb-i} \rceil, & lb-t-1 \le i \le lb \end{cases}$	$n = 2^t m$, m odd, vertical bounds, slant bounds if $n \neq 5k$ and $n \neq 2^{lb-i} a_i$ for $i \ge lb-t-2$
D	$b_i = \begin{cases} \lceil n/(2^{lb-i-1}+1) \rceil, & 0 \le i \le lb-1 \\ n, & i = lb \end{cases}$	n odd, vertical bounds $n \neq (2^j+1)k$, $j \ge 0$
E	$b_i = \begin{cases} \lceil n/2^t(2^{lb-t-(i+1)}+1) \rceil, & 0 \le i \le lb-t-2 \\ \lceil n/2^{lb-i} \rceil, & lb-t-1 \le i \le lb \end{cases}$	$n = 2^t m$, m odd, vertical bounds $n \neq (2^j+1)k$, $j > 0$

problem. Knuth [12] found that $l(191) = l(382)$. He found other integers as well for which $l(2n) = l(n)$. This was a somewhat surprising discovery. Utz [20] had asked if $l(n) < l(2n)$ for all $n > 0$, and it had supposedly been proved earlier by Jonquières [11] that $l(2n) = l(n) + 1$. Subsequently, it has been proved [17, 18] that there are infinite classes of integers for which $l(2n) = l(n)$. If $h(x)$ denotes the number of integers $n \le x$ for which $l(2n) = l(n)$, then computer generated data suggest the possibility that $h(x)$ is bounded below by some constant times x.

While Hansen [10] proved that there exist integers that do not admit star chains among their minimal addition chains, Knuth's computer calculations revealed $n =$ 12,509 as the first number that does not admit a star chain among its minimal chains. It is very likely that $\{2^m(81) + 17, m \ge 8\}$ is an infinite class of such integers.

The addition chain problem is exponential in nature [8]. Improving pruning algorithms enables one to generate the next level of data which can reveal properties that had not previously been observed. For example, $n = 49{,}593$ and $n = 49{,}594$ are adjacent integers, neither of which has a star chain among its minimal chains. $n = 13{,}818$ and $n = 27{,}578$ are even numbers for which $l(2n) = l(n)$. If $n = 2k$, this is an example of an integer for which $l(4k) = l(2k)$. Are there integers for which $l(8k) = l(4k)$? The pairs $n = 22{,}453$, $n = 22{,}455$ and $n = 25{,}019$, $n = 25{,}021$ have the property that $l(2n) = l(n)$. Are there consecutive integers with this property? $n = 29{,}479$ is an integer, all of whose minimal chains start with $1, 2, 3$ which verifies a conjecture of Chin and Tsai [5]. Such integers appear very rarely. The first integer with over 1,000,000 minimal addition chains is 15,126. In fact, NMC(15, 126) $= 1{,}047{,}580$.

Knuth found $c(r)$, the first integer for which $l(n) = r$, for $1 \le r \le 18$. The algorithms discussed in the paper were instrumental in determining that $c(19) =$ $18{,}287, c(20) = 34{,}303, c(21) = 65{,}131$. Flammenkamp and Bleichenbacher have extended this sequence to $c(22) = 110{,}591$, $c(23) = 196{,}591$, $c(24) = 357{,}887$, $c(25) =$ $685,951$, $c(26) = 1,176,431$, and $c(27) = 2,211,837$. See [12] and the online version of [15]. $n = 65{,}131$ is the first integer that requires six small steps in a minimal addition chain. The increase in the required number of small steps in an addition chain greatly increases the computing time. Table 4 shows how the computing time increases with the number of small steps while $\lambda(n)$ is held constant. It also shows how the computing time increases with $\lambda(n)$ while holding the number of small steps constant.

TABLE 4

n	binary n	$\lambda(n)$	number of small steps in chain	pops time
10,241	10100000000001	13	2	4,300 0.00 secs.
10,247	10100000000111	13	3	1,212,209 7.14 secs.
10,271	10100000011111	13	4	220,689,095 1441.00 secs.
20,487	101000000000111	14	3	1,682,449 10.33 secs.
20,494	101000000001110	14	3	2,886,936 15.60 secs.
40,967	1010000000000111	15	3	2,279,240 14.50 secs.
40,988	1010000000011100	15	3	5,286,601 28.23 secs.
81,927	10100000000000111	16	3	3,024,682 19.99 secs.
81,976	10100000000111000	16	3	8,593,425 46.20 secs.

Holding $\lambda(n)$ constant for odd n, while increasing the number of small steps, greatly increases the computing time. As the number of ones in the binary representation of n increases, the number of small steps increases in minimal addition chains for n. In fact $N(n)$, the number of small steps, appears to be bounded below by $\lceil \log_2 \nu(n) \rceil$. As can be seen, computation time is influenced much more strongly by the size of $\nu(n)$ than by the size of n. The minimal addition chains for $n = 1{,}048{,}577$ can be found much more quickly than the minimal addition chains for $n = 191$.

Increasing $\lambda(n)$, while holding the number of small steps constant, increases the computation time relatively slowly. Numbers of the form $n = 2^t m$ for m odd and $t \geq 1$ have longer computation times than odd integers with the same values of $\lambda(n)$ and $N(n)$. As t increases with $\lambda(n)$ and $N(n)$ held constant, the computing time increases. An examination of pruning bounds (C) shows that the bounds are less severe for integers with a large value of t.

The pruning bounds operate the best on odd integers not divisible by 5 and, in particular, on odd integers not divisible by integers of the form $2^i + 1, i \geq 1$.

Data generated by this algorithm have suggested theorems concerning NMC(n), some of which have been established in [19]. These mathematical proofs indirectly confirm the validity of the programs that generated the data that suggested the theorems.

11. Conclusion. The pruning bounds that have been developed significantly improve the efficiency of the search for minimal length addition chains for an integer. It is of interest to note that, for multiples of $2^i + 1$, the pruning bounds cannot be as tight. As can be seen, the best general vertical bounds compare favorably with the VBEST vertical bounds for the examples cited in Table 2.

Introducing slant bounds (bounds for $a_i + a_{i-1}$) increases the efficiency of the search. This class of bounds can be extended to bounds for $a_i + a_{i-2}$ and $a_{i-1} + a_{i-1}$ and perhaps other sums. It appears likely, however, that improvements in efficiency are slight and are negated by the time needed to run all the checks on the additional bounds.

The most famous unsolved problem in addition chains is the Scholz–Brauer con-

jecture which states that $l(2^n-1) \leq n+l(n)-1$. Brauer [4] proved that the conjecture is true when n includes a star chain among its minimal chains. Hansen [10] developed the concept of l^0-chains of which star chains are a proper subset. He proved that the Scholz–Brauer conjecture is true if n includes an l^0-chain among its minimal chains. It remains an open question as to whether every integer includes an l^0-chain among its minimal chains. Computer searches may prove useful in helping to settle this question.

Acknowledgments. This paper has benefited from fruitful discussions with Khalaf Haddad, a former student, and Dan Eilers, president of Irvine Compiler Corporation.

REFERENCES

[1] F. Bergeron, J. Berstel, S. Brlek, and C. Duboc, *Addition chains using continued fractions*, J. Algorithms, 10 (1989), pp. 403–412.
[2] F. Bergeron, J. Berstel, and S. Brlek, *Efficient computation of addition chains*, J. de Théor. Nombres Bordeaux, 6 (1994), pp. 21–38.
[3] J. Bos and M. Coster, *Addition chain heuristics*, in Proceedings CRYPTO '89, 1990, pp. 400–407.
[4] A. T. Brauer, *On addition chains*, Bull. Amer Math. Soc., 45 (1939), pp. 736–739.
[5] Y. H. Chin and Y.-H. Tsai, *Algorithms for finding the shortest addition chain*, in Proceedings National Computer Symposium, Kaoshiung, Taiwan, Dec. 1985, pp. 1398–1414.
[6] H. Dellac, *Question* 49, l'Interm. des Math., 1 (1894), p. 20.
[7] D. Dobkin and R. J. Lipton, *Addition chain methods for the evaluation of specific polynomials*, SIAM J. Comput., 9 (1980), pp. 121–125.
[8] P. Downey, B. Leong, and R. Sethi, *Computing sequences with addition chains*, SIAM J. Comput., 10 (1981), pp. 638–646.
[9] P. Erdős, *Remarks on number theory III on addition chains*, Acta Arith., 6 (1960), pp. 77–81.
[10] W. Hansen, *Zum Scholz-Brauerchen Problem*, J. Reine Angew. Math., 202 (1959), pp. 129–136 (in German).
[11] E. de Jonquières, *Question* 393, (A. Goulard), l'Interm. des Math., 2 (1985), pp. 125–126.
[12] D. E. Knuth, *The Art of Computer Programming*, vol. 2, 3rd ed., Addison–Wesley, Reading, MA, 1997, pp. 461–485.
[13] A. Scholz, *Aufgabe* 253, Jahresber. Deutsch. Math.-Verein., 47 (1937), pp. 41–42.
[14] A. Schönhage, *A lower bound for the length of addition chains*, Theoret. Comput. Sci., 1 (1975), pp. 1–12.
[15] N. J. A. Sloane and S. Plouffe, *The Encyclopedia of Integer Sequences*, Academic Press, San Diego, 1995.
[16] M. V. Subbarao, *Addition chains—some results and problems*, in Number Theory and Applications, R. A. Mollin, ed., Kluwer Academic Publishers, Dordrecht, 1989, pp. 555–574.
[17] E. G. Thurber, *The Scholz–Brauer problem on addition chains*, Pacific J. Math., 49 (1973), pp. 229–242.
[18] E. G. Thurber, *Addition chains and solutions of* $l(2n) = l(n)$ *and* $l(2^n - 1) = n + l(n) - 1$, Discrete Math., 16 (1976), pp. 279–289.
[19] E. G. Thurber, *Addition chains—an erratic sequence*, Discrete Math., 122 (1993), pp. 287–305.
[20] W. R. Utz, *A note on the Scholz–Brauer problem on addition chains*, Proc. Amer. Math. Soc., 4 (1953), pp. 462–463.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1264–1283

DISTRIBUTIONAL WORD PROBLEM FOR GROUPS*

JIE WANG†

Abstract. This paper studies the word problem for finitely presented groups under the restriction that words can only be rewritten for a bounded number of times. We obtain a similar result to the Novikov–Boone theorem in the setting of average-case NP-completeness. The word problem we consider here is to decide, when given a finite presentation of a group G, words x, y, z, and an integer k, whether $(x^{-1}yx)z$ can be derived from $z(x^{-1}yx)$ in the presentation of G in k steps. We show that when each component of the instance is chosen uniformly at random, the problem cannot be solved fast on average unless every NP problem under every reasonable distribution on instances can be solved fast on average.

Key words. average-case NP-completeness, finitely presented groups, word problem

AMS subject classifications. 68Q15, 20F05, 20F10, 03D40

PII. S0097539797332263

1. Introduction.

The notion of NP-completeness, introduced by Cook [6] and independently by Levin [19], captures the intrinsic complexity of certain computational problems that are difficult to solve but whose solutions are easy to verify. Karp [17] introduced the methodology of many–one reductions and demonstrated the rich variety of NP-complete problems. By 1979, several hundred additional NP-complete problems were discovered [8]. Since then, many more problems from almost all areas involving computing have been identified as NP-complete.

Despite many years of intensive effort, no efficient algorithms have been found that can solve any NP-complete problem, and so NP-complete problems are generally thought of as being computationally intractable. However, NP-completeness is a worst-case concept. Being NP-complete does not provide information about how difficult a problem might be on the average case. Indeed, several NP-complete problems have been shown to be tractable "on average." For example, although Hamiltonian Path is NP-complete, Gurevich and Shelah [13] showed that if a graph is chosen under a commonly used distribution on random graphs, then the Hamiltonian Path problem can be solved by a deterministic algorithm in expected linear time. Thus, the average-case complexity of a problem is, in many respects, a more significant measure than its worst-case complexity.

Given a decision problem and a probability distribution on instances—such a pair is called a *distributional problem*—it is an important issue to either find an expected polynomial-time algorithm to solve the problem or prove that such an algorithm does not exist. Levin [20] provided two central notions for studying this issue. One is analogous to the class P and provides an *easiness* notion; the other is analogous to the class of NP-complete sets and provides a *hardness* notion. For the first, Levin defined a robust notion of what it means for the running time of an algorithm to be polynomial on average. For the second, Levin defined deterministic many–one reductions that preserve average-polynomial-time solvability. With this machinery in place,

*Received by the editors December 21, 1997; accepted for publication (in revised form) March 24, 1998; published electronically March 22, 1999.
http://www.siam.org/journals/sicomp/28-4/33226.html
†Department of Mathematical Sciences, University of North Carolina, Greensboro, NC 27402 (wang@uncg.edu). This work was supported in part by NSF grant CCR-9424164 and UNC Greensboro in the form of research leave.

a distributional problem is *average-case NP-complete* if the decision problem component belongs to NP and every distributional problem consisting of an NP problem and a reasonable distribution[1] is reducible to it. Levin [20] showed that distributional *tiling* with a simple distribution is average-case NP-complete, and so it cannot be solved fast on average unless every NP problem under every reasonable distribution can be solved fast on average.

It is considerably more difficult to obtain average-case completeness results than to obtain worst-case completeness results. Reductions that work for the worse case often do not work for the average case. For the reduction to work for the average-case completeness, the probability distribution of the target instance obtained from the reduction cannot be too small compared with that of the source instance. So far only a handful of distributional problems has been shown to be average-case NP-complete. These problems are the distributional tiling [20], halting [12], Post correspondence [9, 12], graph edge coloring [27], matrix transformation [4, 11], matrix representability [28], and string rewritings for semigroups [32]. To advance the theory of average-case NP-completeness, it is essential to identify a greater number of average-case NP-complete problems from various areas, which will also help develop the methodology of reductions to provide tools for proving average-case completeness results. This task is challenging and is a major concern in the research of average-case NP-completeness.

We contribute to the variety of the list of average-case NP-complete problems by showing that the distributional word problem for finitely presented groups is average-case NP-complete. This result may be interesting in its own right in group theory; for we show that, for all practical purposes, some elementary questions about finitely presented groups cannot be answered in average polynomial time, although they can be answered in deterministic exponential time. This paper is the full version of the extended abstract presented in [29].

The word problem for groups was first considered by Dehn [7] and Thue [25], which is to decide, when given a group G and words x, y, whether x is equivalent to y in G. The solution was given by Novikov [22] and, independently, by Boone (1954–1957), who showed that there exists a finitely presented group with an unsolvable word problem. In 1959, Boone [5] exhibited a much simpler group than any of those previously given, and he proved that it has an unsolvable word problem. In 1963, Britton proved Britton's lemma in the course of simplifying and shortening Boone's proof. Further improvements were made afterward by Boone, Collins, and Miller (see [24] for these citations). Groups that have finite presentations have nice combinatorics properties (see, for example, [21]). Moreover, a finite presentation of a group can be read in as an input to an ordinary computer. A possible application of the word problem in public-key cryptography was discussed in [33].

Let $X = Y$ be a relation given in a finite presentation of a group. Then $X^{-1}Y$, XY^{-1}, $Y^{-1}X$, and YX^{-1} are called relators. One can obtain an equivalent word by inserting or deleting a relator at any point on a given word. We assume that rewriting one symbol in a word takes one step. The word problem we consider here is to decide, when given a finite presentation of G, words $x,\ y,\ z$, and an integer k, whether $(x^{-1}yx)z$ can be derived from $z(x^{-1}yx)$ in the presentation of G in k steps. We show that when each component of the instance is chosen uniformly at random, the distributional word problem is average-case NP-complete. We prove it using a deterministic many–one reduction based on the Boone lemma.

[1] By reasonable distribution we mean a distribution that is polynomial-time computable or is dominated by a distribution that is polynomial-time computable. See section 2 for the definition.

This paper is structured as follows. Basic definitions and results of average-case NP-completeness are given in section 2. We define a distributional word problem for finitely presented groups in section 3. In section 4, we briefly outline the Boone–Collins–Miller proof of the Novikov–Boone theorem, which provides building blocks of our completeness proof. The completeness proof is then given in section 5. Some related results are given in section 6.

2. Preliminaries. We provide, in this section, basic definitions and results of average-case complexity theory that we will use in this paper. Motivations regarding the definitions will also be discussed. For a recent survey on average-case complexity theory, the reader is referred to Wang's article [30].

We use the binary alphabet $\Sigma = \{0, 1\}$ for encoding strings. Denote by $|x|$ the length of a binary string x. A *probability distribution* μ is a real-valued function from Σ^* to [0,1] such that $\sum_x \mu(x) = 1$. We assume that μ on the empty string is always 0. We may also use *distribution*, *probability*, *weight*, or *density* to denote probability distribution. The probability distributions we consider are on instances of computational problems. If a binary string does not encode an instance of the underlying problem, then that string has zero probability. The *distribution function* of μ, denoted by μ^*, is defined by $\mu^*(x) = \sum_{y \le x} \mu(y)$, where $\le$ is the standard lexicographical order on Σ^*. For a set $A = \{x : \phi(x)\}$, we use $\mu[\phi(x)]$ or $\mu[A]$ to denote $\sum_{x \in A} \mu(x)$, and we use $|A|$ to denote its cardinal. The conditional distribution of μ on a set A is equal to $\mu(x)/\mu[A]$ if $x \in A$ and $\mu[A] > 0$, and 0 otherwise. For a function f, we use $f^{\varepsilon}(x)$ to denote $(f(x))^{\varepsilon}$ for $\varepsilon > 0$ and we use f^{-1} to denote the inverse of f. We use $\mathbb{R}^+$ to denote the positive reals and $\mathbb{N}$ the natural numbers.

2.1. Average polynomial time. One would naturally intend to measure average-case efficiency of an algorithm using expected polynomial time. An algorithm runs in *expected polynomial time* over a probability distribution μ if there exist $k \ge 0$ such that for all n, $\sum_{x,|x|=n} t(x)\mu_n(x) \le O(n^k)$, where $t(x)$ is the running time of the algorithm on x, and $\mu_n(x)$ is the conditional distribution of μ on strings of length n. However, this definition is machine dependent and so cannot be used to build a robust theory. To define a robust measure of average time, we need to take care of the following subtle and important issues. These issues were either mentioned explicitly or hinted at by Levin [20], and various aspects of the issues have been elaborated on, for example, by Johnson [16], Gurevich [10, 12], Venkatesan [26], and Impagliazzo [14]. From this, Levin's notion of average polynomial time (given here as Definition 2.1) can be derived naturally and can be well justified.

Model independence. Let $\Sigma^n = \{x : |x| = n\}$. Let A be a subset of Σ^n and $|A|$ be proportional to $2^n(1 - 2^{-0.1n})$. Suppose an algorithm runs in polynomial time on every $x \in A$, and runs in $2^{0.09n}$ time on every $x \in \Sigma^n - A$. Then it is easy to see that the expected running time on strings of length n is bounded above by a polynomial in n. However, the expected running time will be exponential in a machine model that is quadratically slower. If a problem is easy on average in one model of computation, then it should be easy on average in all polynomially equivalent models. Even within the same model, if t is polynomial on average, then for any $k > 0$, t^k should also be polynomial on average. Moreover, if a problem is

polynomial on average under one encoding method, then it should be polynomial on average under all polynomially equivalent encodings.

Balancing. Let A be a subset of Σ^n and $|A|$ be proportional to $2^n(1-n^{-2})$. Suppose an algorithm solves a problem in polynomial time on every $x \in A$. Then we still have no guarantee of an algorithm that is fast on average, since the algorithm may take exponential time to solve the instances in $\Sigma^n - A$. A balance is therefore required between the portion of hard instances and the hardness of these instances. The portion of the hard instances should be polynomially related to the time needed to solve them. Clearly, it is at least necessary that only a subpolynomial portion of instances should require superpolynomial time, though this will not in general be sufficient.

Under the average-case measurement, an efficient algorithm with input distribution μ is allowed to run a longer time on instances with smaller weights. One may measure the "rareness" of instances by a real-valued function $r\colon \Sigma^+ \to \mathbb{R}^+$, defined in such a way that if the weight of an instance x is smaller, then the value of its rareness $r(x)$ is larger. As discussed above regarding the balancing issue, the portion of inputs x with high values of rareness should be small. Probably the most general way to satisfy this requirement is to have, for some positive constants k and c and for all $l \in \mathbb{R}^+$, $\mu[r(x) > l^k] < c/l$ or, equivalently, $\sum_x r^\varepsilon(x)\mu(x) < \infty$ for some $\varepsilon > 0$.[2] One may then measure the average-case running time of an algorithm on input x as a function of $|x|r(x)$ rather than $|x|$. This captures and formalizes the idea that a longer time is allowed on inputs with smaller weights, and it also guarantees model independence. The running time t of an algorithm is said to be *polynomial on average* if there exists a $k > 0$ such that, for all x, $t(x) \le (|x|r(x))^k$. Hence, $t^{1/k}(x)|x|^{-1} \le r(x)$. Let $\delta = \min\{\varepsilon, 1\}$. Then $\sum_x t^{\delta/k}(x)|x|^{-1}\mu(x) \le \sum_x (t^{1/k}|x|^{-1})^\delta \mu(x) < \sum r^\delta(x)\mu(x) < \infty$. This gives rise naturally to the following definition given by Levin [20].

DEFINITION 2.1. *A function $f\colon \Sigma^+ \to \mathbb{N}$ is* polynomial on μ-average *if there exists an $\varepsilon > 0$ such that $\sum_x f^\varepsilon(x)|x|^{-1}\mu(x) < \infty$.*

Definition 2.1 is model independent and encoding independent. It satisfies the balancing requirement as shown in footnote 2, based on that the following lemma is straightforward.

LEMMA 2.2. *A function f is polynomial on μ-average iff there are constants $k > 0$ and c such that, for all $l \in \mathbb{R}^+$, $\mu[f(x) > (l|x|)^k] < c/l$.*

Clearly, every polynomial is polynomial on average with respect to any distribution μ. If a function f is an expected polynomial over a distribution μ, then f is polynomial on μ-average. If f and g are polynomial on μ-average, then so are $\max(f, g)$, $f + g$, $f \cdot g$, and f^l for any positive real number l [12].

Let A be a problem and μ be a distribution on instances of A. Then (A, μ) is called a *distributional problem*. A distributional problem (A, μ) is solvable in average polynomial time if A can be solved by a *deterministic* algorithm whose running time is polynomial on μ-average. (A, μ) is called a *distributional decision problem* if A is a decision problem. We will focus on distributional decision problems in this paper; for simplicity, we will omit the word "decision" hereafter.

Let (D, μ) denote a distributional problem, where D is the set of all positive instances x with $\mu(x) > 0$. The problem is to decide, for a given instance x

[2] To see the equivalence, assume that $\mu[r(x) > l^k] < c/l$. Let $\varepsilon = \frac{1}{2k}$. Then, for all $l \in \mathbb{R}^+$, $\mu[r^\varepsilon(x) > l] < c/l^2$, and so $\sum_x r^\varepsilon \mu(x) \le \sum_{n=1}^\infty n\mu[n-1 < r^\varepsilon(x) \le n] = \sum_{n=0}^\infty \mu[r^\varepsilon(x) > n] < \infty$. The other direction is straightforward by a simple application of Markov's inequality.

with $\mu(x) > 0$, whether $x \in D$. If $D \in$ NP, then (D, μ) is called a distributional NP problem.

Denote by AP the class of all distributional problems that are solvable in average polynomial time.

REMARK 2.1. One may impose certain restrictions on the expressions that define average computation time. One such restriction [23] requires that the expressions that define average computation time be converged not just on one distribution but on all distributions of the same rank. Belanger and Wang [1] showed that the class of distributional NP problems defined with respect to the ranking of distributions does not provide harder problems than standard average-case NP-complete problems.

2.2. Polynomial-time reductions. Given two distributional problems, we wish to know which one is computationally more difficult. A standard technique for such comparisons is to find a reduction from one problem to another.

Let f be a function with input distribution μ. It is reasonable to define the output distribution on y, denoted by $f(\mu)(y)$, to carry all the weights of those instances x with $f(x) = y$. Namely, for all $y \in \text{range}(f)$, $f(\mu)(y) = \sum_{f(x)=y} \mu(x)$. Denote by $f^*(\mu)$ the distribution function of $f(\mu)$, so $f^*(\mu)(y) = \sum_{f(x)\leq y} \mu(x)$. Reductions f from (A, μ) to (B, ν) should be closed for AP, meaning that if $(B, \nu) \in$ AP then so is (A, μ). One way to guarantee this is to require that f efficiently reduces A to B as in the worst case and that it should not reduce an instance of A that has a large probability to an instance of B that has a very small probability. This means that the induced weight on the output $y = f(x)$ should be bounded above (within a polynomial factor) by the weight on y according to the distribution of B. Namely, $f(\mu)(y) \leq |y|^{O(1)}\nu(y)$.[3] If $|y| \leq |x|^{O(1)}$, it follows that there exists a distribution μ_1 such that, for all x, it holds that $\mu(x) \leq |x|^{O(1)}\mu_1(x)$ and $\nu(f(x)) = f(\mu_1)(f(x))$,[4] which turns out to be sufficient for defining reductions. The following definitions are due to Levin [20], and we state them following Gurevich [12].

DEFINITION 2.3. *Let μ and ν be distributions. Then μ is* dominated by ν*, written as $\mu \preceq \nu$, if there exists a polynomial p such that, for all x, $\mu(x) \leq p(|x|)\nu(x)$.*

DEFINITION 2.4. *Let μ and ν be distributions on instances of the decision problems A and B, respectively, and let f be a reduction from A to B. Then μ is* dominated by ν with respect to f*, written as $\mu \preceq_f \nu$, if there exists a distribution μ_1 on A such that $\mu \preceq \mu_1$ and $\nu(y) = f(\mu_1)(y)$ for all $y \in \text{range}(f)$, i.e., $\nu(y) = \sum_{f(x)=y} \mu_1(x)$.*

DEFINITION 2.5. *(A, μ) is* polynomial-time reducible *to (B, ν) if there is a polynomial-time computable reduction f such that for all x, $x \in A$ iff $f(x) \in B$, and $\mu \preceq_f \nu$.*

It is straightforward to see that, if a reduction f is one to one, then $\mu \preceq_f \nu$ iff $\mu \preceq \nu \circ f$ [12]. This observation is useful in proving completeness results. In the sequel, we will often use "p-time" to denote "polynomial-time." The following lemma is straightforward and a proof can be found, for example, in [12, 30].

LEMMA 2.6. (1) *If (A, μ) is p-time reducible to (B, ν) and (B, ν) is in* AP*, then so is (A, μ).* (2) *The p-time reductions are transitive.*

[3] Actually, we are dealing with the conditional probability distribution $\nu[y|Y]$ with $Y = \text{range}(f)$. Since $\nu[y|Y] = \nu(y)/\nu[Y]$ and $\nu[Y]$ is a positive constant, this is equivalent to using $\nu(y)$ in the inequality.

[4] The converse is also true if $|x|$ is bounded by a polynomial in $|f(x)|$ [12].

2.3. Polynomial-time computable distributions. Let (D, μ) be a distributional NP problem. If every other distributional NP problem is p-time reducible to it, then (D, μ) is average-case complete. In order for such a complete problem to exist, a certain condition on allowable distributions is needed.[5] Levin [20] suggested that it is reasonable to require distributions to be p-time computable. A real-valued function $f : \Sigma^* \to [0, 1]$ is p-time computable [18] if there is a deterministic algorithm which, on every string x and every positive integer k, outputs a finite binary fraction y in time bounded by a polynomial in $|x|$ and k and such that $|f(x) - y| \leq 2^{-k}$. Clearly, if μ^* is p-time computable then so is μ. Blass showed that the converse is not true unless P = NP (see [12]). With this fact in mind, we assume throughout this paper that when we say that a distribution μ is p-time computable we mean that both μ and μ^* are p-time computable.

Levin (see [16]) hypothesized that any natural distribution μ is either p-time computable or is dominated by a p-time computable distribution. Strong evidence that supports this hypothesis is the fact that all the commonly used discrete distributions do satisfy this hypothesis.

Denote by DistNP the class of all distributional NP problems (D, μ) such that $\mu \preceq \nu$ for some p-time computable distribution ν. By Levin's hypothesis, DistNP includes all natural distributional NP problems. A distributional problem (A, μ) is called *average-case* NP-*complete* (or *complete for* DistNP) if $(A, \mu) \in$ DistNP and every other distributional problem in DistNP is p-time reducible to (A, μ).

REMARK 2.2. Distributions μ that are p-time computable are *p-sampleable* [10, 2] meaning that there is a probabilistic algorithm that, without reading any input, outputs x with probability $\mu(x)$ in time polynomial in $|x|$. Although there are p-sampleable distributions that are not p-time computable under the assumption that one-way functions exist [2], Impagliazzo and Levin [15] showed that for NP search problems, p-sampleable distributions do not generate harder instances than simply picking instances uniformly at random. In particular, they showed that for any distributional NP search problem (A, μ), where μ is p-sampleable, there is a distributional NP search problem (B, ν), where ν is a p-time computable, uniform distribution, such that (A, μ) is reducible to (B, ν) under a randomized reduction.[6] For decision problems, the same result can be obtained using a randomized truth-table reduction [2] (see also [30]). Hence, as far as average-case NP-completeness is concerned, it is sufficient to focus on distributions that are p-time computable.

2.4. Uniform distributions. Distributions may be defined on all binary strings by first selecting lengths and then selecting strings of that length. Although it is mathematically impossible to select strings with equal chance from an infinite sample space, strings of the same length can be selected with equal likelihood. It is also impossible to select integers from $\mathbb{N}$ with the same probability, but one can select an integer with a probability close to being "uniform." A p-time computable distribution μ on Σ^* is called *uniform* if for all x, $\mu(x) = \pi(|x|)2^{-|x|}$, where $\sum_n \pi(n) = 1$ and there is a polynomial p such that for all but finitely many n, $\pi(n) \geq 1/p(n)$.

It is important to note that for the purpose of proving completeness results, $\pi(n) \geq 1/p(n)$ is the only requirement needed since domination allows a polyno-

[5] If arbitrary distributions are allowed, Wang and Belanger [31] showed that no complete distributional problems can exist with respect to one-to-one p-time reductions.

[6] We will only use deterministic reductions in this paper. Randomized reductions are weaker reductions that can be used to establish completeness results when deterministic reductions are not applicable [9, 12, 27, 32].

mial factor, and so some longer strings can certainly be given more weights than shorter ones. Levin [20] used n^{-2} for $\pi(n)$ for notational convenience (normalized by dividing by $\sum_n n^{-2} = \pi^2/6$), and $|x|^{-2}2^{-|x|}$ is often referred to as the default uniform distribution.

2.5. Distribution controlling lemma. Let μ be a p-time computable distribution. Then μ is dominated (with respect to p-time computable functions) by uniform distributions, which is an important property for proving completeness results. Levin [20] first proved this property using a "perfect rounding" technique. Gurevich [12] provided a different and easier proof, based on that Wang and Belanger [32] further showed that if $\mu(x) \geq 2^{-p(|x|)}$ for some fixed polynomial p, then μ will also dominate the same uniform distribution within a constant factor.

LEMMA 2.7 (distribution controlling lemma). *Let μ be a p-time computable distribution.*

1. *There exists a total, one to one, p-time computable, and p-time invertible function $\alpha\colon \Sigma^* \to \Sigma^*$ such that for all x, $\mu(x) < 4 \cdot 2^{-|\alpha(x)|}$.*
2. *If there exists a polynomial p such that for all x, $\mu(x) > 2^{-p(|x|)}$, then there is a total, one-to-one, p-time computable, and p-time invertible function $\beta\colon \Sigma^* \to \Sigma^*$ such that for all x, $4 \cdot 2^{-|\beta(x)|} \leq \mu(x) < 20 \cdot 2^{-|\beta(x)|}$.*

3. Finitely presented groups and word problems. A finite presentation of a group consists of a set of generators (abstract symbols) and a set of relations that relate the freely generated words. To be precise, let A be a finite set. The free group $[A]$ is a group including all elements that can be uniquely written as a reduced *word* in the form $a_1^{\pm}a_2^{\pm}\cdots a_n^{\pm}$, where $a_i \in A$, $a_i^{\pm}$ represents a_i or a_i^{-1}, and no a_i appears adjacent to a_i^{-1}. When $n = 0$, we get the empty word e, which is the identity of the group. A word is positive if it does not contain negative exponents. The empty word e is regarded as a positive word. Positive words are also called strings. Words are multiplied by juxtaposing with all expressions $a_i a_i^{-1}$ and $a_i^{-1}a_i$ canceled out until a reduced word is obtained. For each word w, the inverse word w^{-1} consists of all the symbols of w written in reverse order, where each a_i is replaced by a_i^{-1} and each a_i^{-1} is replaced by a_i.

A relation on A is an expression $X = Y$, where X and Y are words on A. Let R be a finite set of relations on A. Let $\mathcal{K}_R$ denote the normal subgroup of $[A]$ generated by the XY^{-1} for $X = Y \in R$. A group G has a set of generators A and a set of relations R on A if G is isomorphic to the quotient group $[A]/\mathcal{K}_R$. A and R form a finite presentation of G, denoted by $[A; R]$. We extend the notation $[A; R]$ to allow several generators or sets of generators before the semicolon and several relations or sets of relations after the semicolon.

A quantifier may also be used to describe a bunch of similar relations in a compressed form. For instance, $\forall x \in A : x^3 = x^2$ represents relations $a_i^3 = a_i^2$ for $i = 1, 2, \ldots, n$. In this case, we say that $a_i^3 = a_i^2$ for a given i is a relation specified by the quantified statement $\forall x \in A : x^3 = x^2$. For simplicity, we use the term "quantified relation" for such a quantified statement.

Recall that if $X = Y$ is a relation, then XY^{-1}, $X^{-1}Y$, YX^{-1}, and $Y^{-1}X$ are called relators. Obviously, if a is a generator, then aa^{-1} and $a^{-1}a$ are relators. Let u and v be (not necessarily reduced) words. Then $u \equiv v$ means that u and v have exactly the same spelling. An equivalent word can be obtained by eliminating or introducing relators at any point on the original word. We define the following elementary transformations (rewriting rules) for a finite presentation $[A; R]$ of a group, denoted by $\leftrightarrow_R$, where $\leftrightarrow_R$ is a symmetric operation.

DEFINITION 3.1. *Let y be a relator.*

1. *If x is an empty word, then $x \leftrightarrow_R y$.*
2. *If x is not empty and $X \equiv x_1x_2$ (x_1 or x_2 could be possibly null), then $x \leftrightarrow_R x_1yx_2$.*
3. *If $x \equiv x_1Xx_2$, $x' \equiv x_1X^{-1}x_2$ (x_1 and x_2 could be possibly null), and $X = Y$ is a relation, then $X \leftrightarrow_R x_1Yx_2$ and $x' \leftrightarrow_R x_1Y^{-1}x_2$.*

We assume that rewriting one symbol in a word takes one step. So if $u \leftrightarrow_R v$, then it will take at most $\max\{|u|, |v|\}+O(1)$ steps to derive v from u, or derive u from v. Also, if $x \equiv a_1^{\pm} \cdots a_m^{\pm}$, where each a_i is a symbol, then obtaining $x^{-1} = a_m^{\mp} \cdots a_1^{\mp}$ takes $O(\Sigma_{i=1}^m |a_i|)$ steps. Operations like this that do not involve relations are referred to as group operations.

DEFINITION 3.2. *Let $[A; R]$ be a finite presentation of a group. Let u and v be two words on A. Then u can be obtained from v in n elementary transformations in $[A; R]$, written as $u \overset{n}{\longleftrightarrow}_R v$, if there are words $u_1, \ldots, u_{n-1}$ on A such that $u \leftrightarrow_R u_1 \leftrightarrow_R \cdots \leftrightarrow_R u_{n-1} \leftrightarrow_R v$. Write $\overset{*}{\longleftrightarrow}_R$ to denote $\overset{k}{\longleftrightarrow}_R$ for some k. Write $u =_k v$ (in $[A; R]$) if v can be derived from u in k steps using relations in R and group operations.*

The subscript R is often omitted from $\leftrightarrow_R$ when there is no confusion. Let G be a finitely presented group. Then G has many different finite presentations. The number of steps used in derivations depends on the presentation. Let $[A; R]$ and $[A'; R']$ be two different presentations of G. If $u =_k v$ in $[A; R]$ for some k, then $u =_{k'} v$ in $[A'; R']$ for some k'. When we are not concerned with the number of steps used in derivations, we simply write $u = v$ (in G) to denote $u =_k v$ in some presentation of G.

Let A be a finite set of binary strings $\{a_1, \ldots, a_m\}$. We use $\| A \|_0$ to denote $\sum_{i=1}^m |a_i|$ and $\| A \|_1$ to denote $\prod_{i=1}^m |a_i|$. For a presentation $[A; R]$, we assume that words over A and relations (with or without quantifiers) in R are properly coded as binary strings.

We consider the following distributional word problem for finitely presented groups, which is a slight modification of the original word problem of Dehn and Thue.

DEFINITION 3.3. *The* distributional word problem for groups *is the following decision problem.*

Instance: *A finite presentation $[A; R]$ of a group, binary strings u, v, w, and a unary notation 1^n representing positive integer n, where $A = \{a_1, \ldots, a_l\}$ of generators and $R = \{r_1, \ldots, r_m\}$ of relations (with or without quantifiers), all coded in binary form.*

Question: *Is $(u^{-1}vu)w =_k w(u^{-1}vu)$ in $[A; R]$ for $k \leq n$? ($u^{-1}vu$ is called a conjugation of u on v, denoted by $u \diamond v$.)*

Distribution: *Randomly and independently select positive integers l, m, n, and binary strings u, v, and w. Then randomly and independently choose binary strings $a_1, \ldots, a_l$ and $r_1, \ldots, r_m$. The random choices are made with respect to the default uniform distributions on positive integers and binary strings. Hence, the probability distribution is proportional to*

$$\frac{2^{-(\|A \cup R\|_0 + |u| + |v| + |w|)}}{(lmn|u||v||w| \ \| A \cup R \|_1)^2}.$$

The distributional word problem for groups is in DistNP, for we can first guess a bounded sequence of relations and the locations where elementary transformations will

be applied and then verify whether we have a correct guess. We will show in section 5 that the distributional word problem for groups is average-case NP-complete.

We can similarly define a word problem for semigroups, where we start with generators and relations (also called string-rewriting rules) as before but without the inverses. In defining equivalent strings we can only replace any occurrence of X by Y and vice versa, where $X = Y$ is a string-rewriting rule. We can similarly define elementary string-rewriting operation $\leftrightarrow$ and operation $\overset{k}{\longleftrightarrow}$, which are symmetric operations. A string-rewriting system (i.e., the set of all string-rewriting rules) serves as a bridge in our proof to link a Turing machine computation to a finite presentation of a group.

We want to know what kind of relations in a finitely presented group would make the bounded word problem difficult (or easy) to solve in average polynomial time. Clearly, for free groups, the word problem can be solved in linear time. We also note that for finitely generated groups with relations $ab = ba$ for all symbols a and b, the word problem can be solved in polynomial time. To identify relations that will make the word problem difficult to solve, we will first investigate, in the next section, undecidable word problems for finitely presented groups.

4. Undecidable word problem for groups. For the reader's convenience, we briefly outline the Boone–Collins–Miller proof of the Novikov–Boone theorem in exactly the same form presented by Rotman [24]. We first describe the Boone lemma.

Let Γ be a semigroup with the following finite presentation:

$$\Gamma = [q, q_1, \dots, q_N, s_1, \dots, s_M; F_i q_{i_1} G_i = H_i q_{i_2} K_i, i \in I],$$

where F_i, G_i, H_i, K_i are (possibly empty) positive words on $\{s_1, \dots, s_M\}$ and q_{i_1}, $q_{i_2} \in \{q, q_1, \dots, q_N\}$. Here $|I|$ is the number of relations in the presentation. (The q and s symbols will be used to represent states and nonstate symbols of some fixed Turing machine that accepts the halting problem.)

The following notation is convenient. Let $X \equiv a_1^{\epsilon_1} \cdots a_m^{\epsilon_m}$ be a (not necessarily positive) word; then $\overline{X} \equiv a_1^{-\epsilon_1} \cdots a_m^{-\epsilon_m}$. If X and Y are words on $\{s_1, \dots, s_M\}$, define

$$(X q_j Y)^* \equiv \overline{X} q_j Y,$$

where $q_j \in \{q, q_1, \dots, q_N\}$.

A word w is *special* if $w \equiv \overline{X} q_j Y$, where X and Y are positive words on $\{s_1, \dots, s_M\}$, and $q_j \in \{q, q_1, \dots, q_N\}$. We now present a finitely presented group G that has an undecidable word problem. G has generators

$$q, q_1, \dots, q_N, s_1, \dots, s_M, r_i, i \in I, \chi, \tau, \kappa,$$

and relations, for all $i \in I$ and $b = 1, \dots, M$,

$$\begin{aligned}
\chi s_b &= s_b \chi^2, \\
r_i s_b &= s_b \chi r_i \chi, \\
r_i^{-1} \overline{F}_i q_{i_1} G_i r_i &= \overline{H}_i q_{i_2} K_i, \\
\tau r_i &= r_i \tau, \\
\tau \chi &= \chi \tau, \\
\kappa r_i &= r_i \kappa, \\
\kappa \chi &= \chi \kappa, \\
\kappa(q^{-1} \tau q) &= (q^{-1} \tau q) \kappa.
\end{aligned}$$

LEMMA 4.1. (see [5]). *If w is a special word, then $k(w^{-1}tw) = (w^{-1}tw)k$ in G iff $w^* = q$ in Γ.*

The proof of the Boone lemma is based on HNN (Higman–Neumann–Neumann) extensions on free products of finitely presented groups. For a detailed and comprehensive proof of Boone's lemma, the reader is referred to Rotman [24, pp. 363–371].

To obtain the Novikov–Boone theorem, we first state a special form of the Markov–Post theorem given as Lemma 4.2 below. Its proof can be found from [24, pp. 360–361].

LEMMA 4.2. *There is a finitely presented semigroup*

$$\Gamma = [q, q_1, \ldots, q_N, s_1, \ldots, s_M; F_i q_{i_1} G_i = H_i q_{i_2} K_i, i \in I]$$

with unsolvable word problem, where F_i, G_i, H_i, K_i are (possibly empty) positive words on $\{s_1, \ldots, s_M\}$ and q_{i_1}, $q_{i_2} \in \{q, q_1, \ldots, q_N\}$. There is no algorithm to determine, for arbitrary positive words X and Y on $\{s_1, \ldots, s_M\}$ and for q_j, $1 \le j \le N$, whether $Xq_jY = q$ in Γ.

The symbols $q, q_1, \ldots, q_N$ in Lemma 4.2 are sometimes referred to as states, for they correspond to states of some fixed Turing machine that accepts the halting problem.

From Boone's lemma, we know that if there were an algorithm to determine whether two words are equivalent in G, then there would be an algorithm to determine, for an arbitrary special word w, whether $w^* = q$ in Γ, which is impossible by Lemma 4.2. This gives the following Novikov–Boone theorem.

THEOREM 4.3 (see [22, 5]). *There exists a finitely presented group having an undecidable word problem.*

5. Main theorem.

THEOREM 5.1. *The distributional word problem for groups is average-case* NP*-complete.*

We will devote this entire section to proving this theorem. The proof is as follows. Given a distributional NP problem $(D, \mu) \in$ DistNP, we construct a reduction such that for any instance x of D, the reduction transforms x into a finitely presented group $G = [A; R]$, strings u, v, w, and a unary notation $1^{\eta(|x|)}$ with the following properties, where η is a polynomial:

1. $x \in D$ iff $(u^{-1}vu)w = w(u^{-1}vu)$ in G iff $(u^{-1}vu)w =_k w(u^{-1}vu)$ in $[A; R]$ for $k \le \eta(|x|)$.
2. μ is dominated by the distribution of the distributional word problem with respect to the reduction.

The reduction construction is based on the Boone lemma. There are some added difficulties, however. First, we need to show a similar result as Lemma 4.2 in the setting of average-case NP-completeness. Wang and Belanger [32] have shown that the distributional word problem for semigroups is average-case NP-complete. But the relations constructed in the proof there do not have the forms we desire. We would like all relations to be in the form of $EqF = HpK$, where p and q are states, E, F, H, and K are positive words on nonstate symbols. This can be achieved by carefully reengineering the proof given in [32].

Second, we note that it is often difficult to satisfy the domination property for distributions required in the reductions. In general, if the size of the output of a reduction is a linear growth on the size of its input, then the domination property could be damaged. To make this point clearer, let us consider the distributional halting problem.

Let $M_1, M_2, \ldots$ be a fixed enumeration of nondeterministic Turing machines in

which the index i is an integer in binary form that codes the symbols, states, and transition table of the ith Turing machine M_i. The distributional halting problem consists of binary strings i, x, and 1^n as instances, where i and n are integers. The question is to decide whether M_i accepts x within n steps. Denote by K the set of all positive instances. The probability distribution $\mu_K(i, x, 1^n)$ of instance $(i, x, 1^n)$ (positive or negative) is proportional to $2^{-(l+m)}l^{-2}m^{-2}n^{-2}$, where $l = |i|$ and $m = |x|$. The distributional halting problem (K, μ_K) is average-case NP-complete as mentioned in section 1. If we try to reduce (K, μ_K) to a distributional problem (D, μ) in an effort to show that (D, μ) is complete, we will need to find a reduction f such that $\mu_K \preceq_f \mu$. If the reduction f transforms $(i, x, 1^n)$ to z such that z has a parameter y with $|y| \geq c|x|$ for some $c > 1$, and $2^{-|y|}$ is an irreducible factor of μ, then μ cannot dominate μ_K with respect to f. The only possibility left in this connection is for y to satisfy $|y| = |x| + O(\log |x|)$. This motivated Gurevich [12] to invent a dynamic binary coding scheme to meet this requirement. Using such a coding scheme, Gurevich [12] showed that the distributional Post correspondence is average-case NP-complete. We will use this coding scheme to construct our reduction as well.

5.1. Dynamic coding scheme. Let A be a finite alphabet with $|A| > 2$. Given a binary string x (we also assume that x starts with 1), Gurevich [12] designed a dynamic binary coding scheme to encode symbols in A such that the following statements hold.

1. All coded symbols have the same length l, and $l = 2\log |x| + O(1)$.
2. String x (not coded) is distinguishable from each coded symbol. In other words, no coded symbol can be a substring of x.
3. If a nonempty suffix z of a coded symbol u is a prefix of a coded symbol v, then $z = u = v$.
4. x can be written as a unique concatenation of the following fixed binary strings 1, 10, 000, 100, which are not prefixes of any coded symbol.

The four strings 1, 10, 000, 100 are called *base strings*. Gurevich [12] originally used strings 1, 10, 00, and 000 as base strings, which, in general, does not form a unique concatenation for strings beginning with 1.

The construction of such a coding system can be done as follows. Let R be the regular set $0100(00 + 11)^*11$. Let l be the least even integer such that $2^{(l-6)/2} \geq |x| + |A|$. Hence, $l = 2\log |x| + O(1)$. The string x has at most $|x|$ substrings of length l. So we can select a set S of R-strings of length l such that $|S| = |A|$, no string in S is a substring of x, and every string in S starts with 01. Moreover, from R it is straightforward to show that the third condition also holds [12]. To see that the fourth condition is satisfied, let y be a string that does not start with 01 and is different from 0; then it is straightforward to show by induction that y forms a unique concatenation of the base strings.

So we can use S to code A by assigning one element in S to represent one symbol in A. This dynamic coding scheme satisfies all of the above four conditions.

5.2. Semigroup construction. For simplicity, we use the standard, nondeterministic, one-tape, single-headed Turing machines as our computation model. The tape is bounded on the left and unbounded to the right. At the initial state, the input is left justified on the tape and the head is positioned at the leftmost symbol of the input.

Let (D, μ) be an arbitrary distributional problem in DistNP. From distribution controlling lemma (1), there is a total, one-to-one, p-time computable, and p-time invertible function α such that $\mu(x) \preceq 2^{-|\alpha(x)|}$. Let M be a (nondeterministic) Turing

machine that accepts D in polynomial time. Without loss of generality we assume that all the computation paths of M are bounded by a polynomial in the length of inputs.

We construct a Turing machine M' with only one halting state. M' takes input $1w$ and determines whether $w = \alpha(x)$ for some x. If $\alpha^{-1}(w)$ is not defined, then M' goes into an infinite loop; otherwise, M' simulates M on x. If M on x reaches an accepting state, then M' erases all the tape symbols, moves the head to the left, and halts. If M on x reaches a rejecting state, then M' simply goes into an infinite loop. We assume that when M' enters an infinite loop, it does not change anything on the tape. Then for all x, M' on input $1\alpha(x)$ has a halting computation if and only if M accepts x. Because of the time bound on M, it is easy to see that if M' on input $1\alpha(x)$ halts, then the number of steps it takes is bounded by a polynomial in $|x|$. This proves the following lemma.

LEMMA 5.2. *$x \in D$ iff M' halts on $1\alpha(x)$ in time polynomial in $|x|$ iff M' halts on $1\alpha(x)$.*

We construct a set $\Pi = \Pi(M')$ of production rules to describe instantaneous descriptions of M'. Let h be the halting state, s the initial state, B the blank symbol, Q the set of states, and δ the transition function, all for M'. We use a symbol \$ as an end marker to handle the blank tape detail. Π consists of the following ordered pairs (productions).

1. For any $q \in Q - \{h\}$, $p \in Q$, $a, b, c \in \Sigma \cup \{B\}$, if $\delta(q, a) = (p, b, R)$, then Π contains $\langle qac, bpc\rangle$; Π also contains $\langle qa\$, bpB\$\rangle$.
2. For any $q \in Q - \{h\}$, $p \in Q$, $a, b, d, \in \Sigma \cup \{B\}$, and $c \in \Sigma \cup \{\$\}$, if $\delta(q, a) = (p, b, L)$, then Π contains $\langle dqB\$, pd\$\rangle$; Π also contains $\langle dqac, pdbc\rangle$ for $a \neq B$, $c \neq \$$, or $b \neq B$.

We now construct a finite string-rewriting system $\Gamma = \Gamma(M')$ based on Π. We first construct a new set of symbols S_1 and a disjoint new set of states Q_1. As we discussed earlier, we require that Γ contains rewriting rules only in the form of $EqF = HpK$, where p, q are states in Q_1, and E, F, H, K are strings on S_1.

Since Π is finite, we can fix an order $ord : \Pi \to \{1, 2, \ldots, |\Pi|\}$ for the ordered pairs in Π. We use new symbols d_i, $i = 1, \ldots, |\Pi|$ to handle nondeterminism in case the inverse of a production rule is applied. We use two extra states s_1 and s_2 to transform input $1\alpha(x)$ to its coded form before the actual simulation of M' on $1\alpha(x)$ begins. We use another two extra states s_3, s_4 and a set of new symbols $Q' = \{q' : q \in Q\}$ as markers to move d_i around when needed. Let

$$S_1 = \{0, 1, B, \$\} \cup \{d_i : i = 1, \ldots, |\Pi|\} \cup Q',$$
$$Q_1 = Q \cup \{s_1, s_2, s_3, s_4\}.$$

Γ will contain $\gamma = 24 + |Q| + 7|\Pi| + 3|\Pi||Q| + |\Pi|^2$ rules, which will be described later.

Let S_2 be a new set defined below:

$$S_2 = S_1 \cup Q_1 \cup \{\chi, \kappa, \tau, \xi\}.$$

Some extra symbols are needed for writing quantified relations. Let S_3 be a finite alphabet distinct from S_2 that is sufficient to describe quantified relations (given in the construction of G later). So S_3 contains quantifier symbols, relational operation symbols, and variable symbols, etc. We also assume that S_3 contains $\{!, \uparrow, (,)\}$. Symbols χ, κ, τ, ξ will be used to construct relations in the construction of finite presentations for groups, and symbols in S_3 will be used to describe relations.

Let $z = 1\alpha(x)$. Fix a dynamic binary coding scheme for z and the finite set

$$S_4 = S_2 \cup S_3.$$

It follows that all coded symbols have the same length $l = 2\log|\alpha(x)| + O(1) = O(\log|x|)$. For any positive word w on S_4, we use $\underline{w}$ to denote the coded word of w.

Let $\mathcal{B} = \{1, 10, 100, 000\}$. The string-rewriting system Γ on $S_1 \cup Q_1$ consists of the following rewriting rules:

$\Gamma 1$: $\forall u \in \mathcal{B}$:

$$\begin{aligned}
\underline{s}u &= \underline{\$us_1},\\
\underline{s_1}u &= \underline{us_1},\\
\underline{us_1\$} &= \underline{s_2u\$},\\
\underline{us_2} &= \underline{s_2u},\\
\underline{\$s_2} &= \underline{\$s}.
\end{aligned}$$

$\Gamma 2$: $\forall\langle\alpha,\beta\rangle \in \Pi$: $\underline{\alpha} = \underline{d_i\beta}$, where $i = ord(\langle\alpha,\beta\rangle)$.

$\Gamma 3$: $\forall a \in \{0,1\}, \forall q \in Q, \forall d_i, d_j$:

$$\begin{aligned}
\underline{d_iq} &= \underline{d_is_3q'},\\
\underline{d_iaq} &= \underline{d_ias_3q'},\\
\underline{d_ias_3} &= \underline{d_is_3a},\\
\underline{ad_is_3} &= \underline{d_is_3a},\\
\underline{d_jd_is_3} &= \underline{d_jd_is_4},\\
\underline{\$d_is_3} &= \underline{\$d_is_4},\\
\underline{s_4a} &= \underline{as_4},\\
\underline{s_4q'} &= \underline{q}.
\end{aligned}$$

$\Gamma 4$:

$$\begin{aligned}
\forall d_i : \underline{d_ih} &= \underline{h},\\
\underline{\$hB\$} &= \underline{h}.
\end{aligned}$$

Recall that a string-rewriting system defines a presentation of a semigroup.

LEMMA 5.3. *There is a polynomial ζ such that M' halts on input z iff $\underline{s}z\underline{\$} = \underline{h}$ in Γ iff $\underline{s}z\underline{\$} \overset{k}{\longleftrightarrow} \underline{h}$ in Γ for $k \le \zeta(|x|)$. Moreover, the length of each word ever obtained in the transformation process is bounded by $\zeta(|x|)$.*

Proof. Starting from string $\underline{s}z\underline{\$}$, the only rewriting rules that can be applied are the ones in $\Gamma 1$, starting with the first one, until $\underline{s}z\underline{\$}$ is transformed into $\underline{\$sz\$}$. In other words, the rules in $\Gamma 2$–$\Gamma 4$ can be applied only after $\underline{\$sz\$}$ is obtained. Since z can be uniquely written as $u_1 \cdots u_m$, where $u_i \in \mathcal{B}$, this transformation can be carried out in $2m+1$ steps: $\underline{s}z\underline{\$} \leftrightarrow \underline{\$u_1s_1}u_2 \cdots u_m\underline{\$} \overset{m-1}{\longleftrightarrow} \underline{\$u_1 \cdots u_ms_1\$} \leftrightarrow \underline{\$u_1 \cdots s_2u_m\$} \overset{m-1}{\longleftrightarrow} \underline{\$s_2u_1 \cdots u_m\$} \leftrightarrow \underline{\$sz\$}$.

Assume that M' halts on z in polynomial time $\theta(|x|)$. Using the rules in $\Gamma 2$, which duplicate the steps of M', and using the rules in $\Gamma 3$ as necessary, $\underline{\$sz\$}$ can be transformed into $\underline{\$d_{i_1}d_{i_2}\ldots d_{i_{g(z)}}hB\$}$. Note that after each application of a $\Gamma 2$ rule, it will be necessary to apply at most $g(z) \le 2\theta(|x|) + 3$ $\Gamma 3$ rules to move the symbol $\underline{d_i}$ as far left as possible. This can be seen as follows. When a $\Gamma 2$ rule is applied,

we obtain the following string: $\underline{\$ud_iaqv\$}$ or $\underline{\$ud_iqv\$}$, where u is a string with some d's (possibly null) as a prefix. We consider the second case (the first case is similar). That is, $\underline{\$ud_iqv\$} \leftrightarrow \underline{\$ud_is_3q'v\$}$. We assume that u contains some d's as prefixes (the case that u contains no d's is similar). Let $u = u'd_ju''$, where u' is a string of (possibly null) d's. Let $\ell = |u''|$. It follows that $\underline{\$ud_is_3q'v\$} \overset{\ell}{\longleftrightarrow} \underline{u'd_jd_is_3u''q'v\$} \leftrightarrow \underline{u'd_jd_is_4u''q'v\$} \overset{\ell}{\longleftrightarrow} \$u'd_jd_iu''s_4q'v\$ \leftrightarrow \$u'd_jd_iu''qv\$$.

So transforming $\underline{\$sz\$}$ into $\underline{\$d_{i_1}d_{i_2}\dots d_{i_{g(z)}}hB\$}$ takes at most $\theta(|x|)(2\theta(|x|)+3)$ steps. Finally, $\underline{\$d_{i_1}d_{i_2}\dots d_{i_{g(z)}}hB\$}$ can be transformed into $\underline{h}$ in $\theta(|x|)+1$ steps by $\Gamma 4$ rules.

Conversely, if $\underline{s}z\underline{\$}$ can be transformed into $\underline{h}$, then M' will halt on x. It is important to note that the inverse of a production in Π can undo a simulated step of M', but since each simulated step is kept track of through the d_i's using the $\Gamma 2$ rules, it can only bring the string back to a previous simulated ID of M'.

By Lemma 5.2, M' halts on z iff M' halts on z in time $\theta(|x|)$. So letting $\zeta(n) = (2\theta(n)(\theta(n)+2)+2m+1)l$, where l is the length of a coded symbol, completes the proof. □

Notice that $|\underline{s}z\underline{\$}| = |x| + O(\log|x|)$, which is what we desire. The following corollary is a direct consequence of Lemmas 5.2 and 5.3.

COROLLARY 5.4. *$x \in D$ iff $\underline{s}z\underline{\$} = \underline{h}$ in Γ.*

5.3. Group construction and the reduction. We now construct a finitely presented group G as in Boone's lemma based on the semigroup Γ constructed in section 5.2. However, there is an obstacle in the construction in the polynomial setting. By Boone's lemma, we know that $\underline{s}z\underline{\$} = \underline{h}$ in Γ iff $(\underline{s}z\underline{\$} \diamond \underline{\tau})\underline{\kappa} = \underline{\kappa}(\underline{s}z\underline{\$} \diamond \underline{\tau})$ in G. But there is no guarantee that $\underline{s}z\underline{\$} \overset{k}{\longleftrightarrow} \underline{h}$ in Γ with $k \le \zeta(|x|)$ would imply that $(\underline{s}z\underline{\$} \diamond \underline{\tau})\underline{\kappa} \overset{m}{\longleftrightarrow} \underline{\kappa}(\underline{s}z\underline{\$} \diamond \underline{\tau})$ in G with $m \le \eta(|x|)$ for some polynomial η. In fact, m will be in the exponential in $|x|$. We observe that such an exponential blow up comes from a very special type of strings. By carefully introducing some new generators and relations to serve as "cancellation" rules, we can eliminate the exponential blow up.

The set of generators of G is

$$A = \mathcal{B} \cup \{\underline{a} : a \in S_2\}.$$

The symbols $\underline{!}$ and $\underline{\uparrow}$ are used to represent negative and power words. The symbols in S_3 are used to represent quantified relations.

We will use words of the form $\underline{\xi}^{\pm n}$ in constructing group relations, where $\underline{\xi}^n \equiv \underline{\xi\xi\cdots\xi}$ for n ξ's, and $\underline{\xi}^{-n} \equiv \underline{\xi!\xi!\cdots\xi!}$ for n $\xi!$'s.

For any word w on S_4, we use $\underline{w}$ to denote the coded word of w.

Let x and y be variables; we write $\underline{x^y}$ to denote the concatenation of $\underline{x}$ for y times. Symbolically, $\underline{x^y}$ is written as $\underline{x \uparrow y}$ and $\underline{x^{-y}}$ is written as $\underline{x! \uparrow y}$.

We assume that all integers are written in the original binary form (i.e., not coded in the dynamic coding scheme). Recall that each coded symbol is a string of the form $0100(00+11)^*11$. We code a positive binary integer by replacing 1 with 10, and 0 with 01. A binary number so coded is easily distinguished from any coded symbol in S_4. Let $\mathrm{bin}(n)$ denote such a coded binary number for positive integer n. So $|\mathrm{bin}(n)| = 2|n|$.

We now define a finite set of relations $R = R(M')$ to construct a finite presentation of group

$$G = [A; R].$$

Fix an order for the rewriting rules in Γ. R consists of the following relations, where relations $R1$ to $R5$ are from the proof of the undecidable word problem given in section 4 and relations $R6$ to $R7$ are used to control the length of derivations.

$R1$: $\forall a \in S_1$: $\underline{\chi a} = \underline{a\chi^2}$.

$R2$: $\forall a \in S_1$ and $\forall r_i$: $\underline{r_i a} = \underline{a\chi r_i \chi}$.

$R3$: $\underline{r_i^{-1}\overline{E}qFr_i} = \underline{\overline{H}pK}$, if $\underline{EqF} = \underline{HpK}$ is the ith relation in Γ in the fixed order. Here p and q are states in Q_1.

$R4$:

$$\begin{aligned} \underline{\tau\chi} &= \underline{\chi\tau}, \\ \forall r_i: \ \underline{\tau r_i} &= \underline{r_i\tau}. \end{aligned}$$

$R5$:

$$\begin{aligned} \underline{\kappa\chi} &= \underline{\chi\kappa}, \\ \forall r_i: \ \underline{\kappa r_i} &= \underline{r_i\kappa}, \\ \underline{\kappa}(\underline{h^{-1}\tau h}) &= (\underline{h^{-1}\tau h})\underline{\kappa}. \end{aligned}$$

$R6$: Let W be a variable representing positive words on S_1; let t be a variable representing $|W|$. For all W with length $\leq \zeta(|x|)$ and for all r_i,

1.

$$\begin{aligned} \underline{r_i W} &= \underline{W\xi^t\chi^{-1}r_i\xi^t\chi^{-1}}, \\ \underline{\overline{W}r_i^{-1}} &= \underline{\xi^{-t}\chi r_i^{-1}\xi^{-t}\chi\overline{W}}, \\ \underline{r_i^{-1}W} &= \underline{W\xi^{-t}\chi r_i^{-1}\xi^{-t}\chi}, \\ \underline{\overline{W}r_i} &= \underline{\xi^t\chi^{-1}r_i\xi^t\chi^{-1}\overline{W}}; \end{aligned}$$

2.

$$\begin{aligned} \underline{\tau\xi^t\chi^{-1}} &= \underline{\xi^t\chi^{-1}\tau}, \\ \underline{\tau^{-1}\xi^{-t}\chi} &= \underline{\xi^{-t}\chi\tau^{-1}}, \\ \underline{\kappa\xi^t\chi^{-1}} &= \underline{\xi^t\chi^{-1}\kappa}, \\ \underline{\kappa^{-1}\xi^{-t}\chi} &= \underline{\xi^{-t}\chi\kappa^{-1}}. \end{aligned}$$

$R7$: Let u be a variable representing a number between 1 and $\zeta(|x|)$. Let v be a variable representing 2^u. For all u,

$$\underline{\xi^u} = \underline{\chi^v}.$$

It is easy to see that R contains finitely many relations. The number of relations and the number of quantified relations are independent of x. The length of each relation except for $R6$ and $R7$ is independent of x. For the quantified relations in $R6$ and $R7$, the only thing that depends on x is the value of $\zeta(|x|)$. Anything else can be symbolically written down as the way it is in our coding system (i.e., without further evaluations). For example, 2^u is symbolically written as $\mathrm{bin}(2) \uparrow u$, whose length is $4 + 2l$. The length of each relation is therefore $O(\log |x|)$.

Next, we show that M' halts on input z iff $(\underline{sz\$} \diamond \underline{\tau})\underline{\kappa} =_m \underline{\kappa}(\underline{sz\$} \diamond \underline{\tau})$ in $[A; R]$, where $m \leq$ a fixed polynomial.

LEMMA 5.5. *There is a polynomial η such that if $\underline{s}z\underline{\$} \overset{k}{\longleftrightarrow} \underline{h}$ in Γ with $k \le \zeta(|x|)$, then $(\underline{s}z\underline{\$} \diamond \underline{\tau})\underline{\kappa} =_m \underline{\kappa}(\underline{s}z\underline{\$} \diamond \underline{\tau})$ in $[A;R]$ with $m \le \eta(|x|)$.*

Proof. Assume that $\underline{s}z\underline{\$} \overset{k}{\longleftrightarrow} \underline{h}$ in Γ with $k \le \zeta(|x|)$. Then

$$\underline{s}z\underline{\$} \equiv W_1 \leftrightarrow W_2 \leftrightarrow \cdots \leftrightarrow W_k \equiv \underline{h},$$

and each $|W_i|$ is bounded by $c\zeta(|x|)\log|x|$ for a fixed constant c. Suppose $W_i \leftrightarrow W_{i+1}$ by applying a rewriting rule $\underline{EqF} = \underline{HpK}$; then either $W_i \equiv \underline{UEqFV}$ and $W_{i+1} \equiv \underline{UHpKV}$, or $W_{i+1} \equiv \underline{UEqFV}$ and $W_i \equiv \underline{UHpKV}$, where U and V are positive words on S_1.

Let X be of the form $\underline{UqV}$, where U and V are words on S_1 and $q \in Q_1$. Recall that X^* denotes $\underline{\overline{U}qV}$. We have

$$\underline{\overline{U}(\overline{E}qF)\underline{V}} \overset{2}{\longleftrightarrow} \underline{\overline{U}r_i(r_i^{-1}\overline{E}qFr_i)\underline{r_i^{-1}V}}.$$

Using one relation in $R3$ and two relations specified in $R6(1)$, we have

$$\begin{aligned}\underline{\overline{U}r_i(r_i^{-1}\overline{E}qFr_i)\underline{r_i^{-1}V}} &\leftrightarrow \underline{\overline{U}(r_i\overline{H}pKr_i^{-1})\underline{V}} \\ &\overset{2}{\longleftrightarrow} \mathbf{U}_i(\underline{\overline{UH}pKV})\mathbf{V}_i,\end{aligned}$$

where $\mathbf{U}_i = \underline{\xi^{|U|}\chi^{-1}r_i\xi^{|U|}\chi^{-1}}$ and $\mathbf{V}_i = \underline{\xi^{-|V|}\chi r_i^{-1}\xi^{-|V|}\chi}$. Since $|W_j|$'s are bounded by $O(\zeta(|x|)\log|x|)$, it is straightforward to see that $|\mathbf{U}_i| = 2l|U|+5l \le (2c\zeta(|x|)\log|x|+5)l$ and $|\mathbf{V}_i| = 4l|V| + 4l \le 4l(c\zeta(|x|)\log|x|+1)$. So we have

$$W_i^* \overset{5}{\longleftrightarrow} \mathbf{U}_i W_{i+1}^* \mathbf{V}_i.$$

Write $\mathbf{U}_i^{-1} \equiv \underline{\chi\xi^{-|U|}r_i^{-1}\chi\xi^{-|U|}}$, and $\mathbf{V}_i^{-1} \equiv \underline{\chi^{-1}\xi^{|V|}r_i\chi^{-1}\xi^{|V|}}$.

We can similarly obtain the same result if $W_i \equiv \underline{UHpKV}$ and $W_{i+1} \equiv \underline{UEqFV}$. In this case, $\mathbf{U}_i = \underline{\xi^{-|U|}\chi r_i^{-1}\xi^{-|U|}\chi}$ and $\mathbf{V}_i = \underline{\xi^{|V|}\chi^{-1}r_i\xi^{|V|}\chi^{-1}}$.

So we have

$$\underline{s}z\underline{\$} \equiv W_1 \equiv W_1^* \overset{O(k)}{\longleftrightarrow} \mathbf{U}W_k^*\mathbf{V} \equiv \mathbf{U}\underline{h}\mathbf{V} \text{ in } G, \tag{1}$$

where $\mathbf{U} = \mathbf{U}_1\mathbf{U}_2\cdots\mathbf{U}_{k-1}$, $\mathbf{V} = \mathbf{V}_{k-1}\cdots\mathbf{V}_2\mathbf{V}_1$, $|\mathbf{U}| \le k(2c\zeta(|x|)\log|x|+5)l \le O(\zeta^2(|x|)\log^2|x|)$, and, similarly, $|\mathbf{V}| \le O(\zeta^2(|x|)\log^2|x|)$. This implies that there is a polynomial ϑ such that $\underline{s}z\underline{\$} =_{k'} \mathbf{U}\underline{h}\mathbf{V}$ in $[A;R]$ and $k' \le \vartheta(|x|)$.

From (1), we have

$$(\underline{s}z\underline{\$})^{-1} \equiv \underline{(}\underline{s}z\underline{\$}\underline{)!} \overset{O(k)}{\longleftrightarrow} \underline{(}\mathbf{U}\underline{h}\mathbf{V}\underline{)!} \text{ in } G. \tag{2}$$

Thus,

$$(\underline{s}z\underline{\$})^{-1} =_{O(k')} \mathbf{V}^{-1}\underline{h^{-1}}\mathbf{U}^{-1} \text{ in } G, \tag{3}$$

where $\mathbf{U}^{-1} \equiv \mathbf{U}_{k-1}^{-1}\cdots\mathbf{U}_1^{-1}$ and $\mathbf{V}^{-1} \equiv \mathbf{V}_{k-1}^{-1}\cdots\mathbf{V}_1^{-1}$.

From (1) and (3), and using a polynomial number of relations in $R4$ and $R6(2)$ of R, we have

$$\begin{aligned}(\underline{s}z\underline{\$} \diamond \underline{\tau})\underline{\kappa} &=_{O(k')} \mathbf{V}^{-1}\underline{h^{-1}}(\mathbf{U}^{-1}\underline{\tau}\mathbf{U})\underline{h}(\mathbf{V}\underline{\kappa}) \\ &\overset{O(k)}{\longleftrightarrow} \mathbf{V}^{-1}[(\underline{h^{-1}\tau h})\underline{\kappa}]\mathbf{V} \\ &\leftrightarrow \mathbf{V}^{-1}[\underline{\kappa}(\underline{h^{-1}\tau h})]\mathbf{V} \\ &\overset{O(k)}{\longleftrightarrow} \underline{\kappa}\mathbf{V}^{-1}\underline{h^{-1}}\mathbf{U}^{-1}\underline{\tau}\mathbf{U}\underline{h}\mathbf{V} \\ &\overset{O(k)}{\longleftrightarrow} \underline{\kappa}(\underline{s}z\underline{\$} \diamond \underline{\tau}).\end{aligned}$$

Note that each elementary transformation above takes at most $O(\zeta(|x|)\log|x|)$ steps to rewrite words. This implies that $(\underline{s}z\underline{\$}\diamond\underline{\tau})\underline{\kappa} =_m \underline{\kappa}(\underline{s}z\underline{\$}\diamond\underline{\tau})$ in G for $m \le \eta(|x|)$, where η is some fixed polynomial. This completes the proof. □

From Lemmas 5.2, 5.3, and 5.5, we have Lemma 5.6.

LEMMA 5.6. *If $x \in D$, then $(\underline{s}z\underline{\$}\diamond\underline{\tau})\underline{\kappa} =_m \underline{\kappa}(\underline{s}z\underline{\$}\diamond\underline{\tau})$ in $[A;R]$ for $m \le \eta(|x|)$.*

We now consider the reverse direction of Lemma 5.6. We will first use Tietze transformations to show that G also has presentation $[A-\{\xi\}; R1, R2, R3, R4, R5]$. For the reader's convenience, we state the Tietze theorem as Lemma 5.7 below. The reader is referred to [21, pp. 48–51] for more details.

LEMMA 5.7. *Given a presentation*

$$[b_1, b_2, , \ldots; P_1, P2, \ldots] \tag{4}$$

for a group H, then any other presentation for H can be obtained by a repeated application of the following Tietze transformations to presentation (4)*:*

- **T1** *If the words $T_1, T_2 \ldots$ are derivable from $P_1, P_2, \ldots$, then add $T_1, T_2 \ldots$ to the defining relators in presentation* (4).
- **T2** *If some of the relators, say, $T_1, T_2, \ldots$, listed among the defining relators $P_1, P_2 \ldots$ are derivable from the others, delete $T_1, T_2, \ldots$ from the defining relators in presentation* (4).
- **T3** *If $K_1, K_2, \ldots$ are any words in $b_1, b_2, \ldots$, then adjoin the symbols $c_1, c_2, \ldots$ to the generating symbols in* (4) *and adjoin the relations $c_1 = K_1, c_2 = K_2, \ldots$ to the defining relators in presentation* (4).
- **T4** *If some of the defining relators in* (4) *take the form $a_1 = V_1, a_2 = V_2, \ldots$, where $a_1, a_2, \ldots$ are generators in presentation* (4) *and $V_1, V_2, \ldots$ are words in the generators other than $a_1, a_2, \ldots$, then delete $a_1, a_2, \ldots$ from the generators, delete $a_1 = V_1, a_2 = V_2, \ldots$ from the defining relations, and replace $a_1, a_2, \ldots$ by $V_1, V_2, \ldots$, respectively, in the remaining defining relators in presentation* (4).

So by Tietze transformation T4, we can obtain another presentation for G by removing ξ from the relator set A, removing relations $R7$, and replacing $\xi^{\pm t}$ in relations $R6$ with $\chi^{\pm 2^t}$. Next, we show that the relations specified in $R6$ (after replacing $\xi^{\pm t}$ with $\chi^{\pm 2^t}$) can be derived from relations in $R1$–$R5$.

LEMMA 5.8. *Relations specified in $R6$ (after replacing $\xi^{\pm t}$ with $\chi^{\pm 2^t}$) can be derived from relations in $R1$–$R5$.*

Proof. It suffices to demonstrate how to obtain $\underline{r_i V} \overset{*}{\longleftrightarrow} \underline{V\chi^{2^{|V|}}\chi^{-1}r_i\chi^{2^{|V|}}\chi^{-1}}$ using relations not in $R6$, where V is a positive word. The other ones can be similarly obtained. We prove it by induction on $|V|$. The case of $|V| = 1$ is obvious. For the general case, let $V = V'a$, where $a \in S_1$; then $\underline{r_i V} \overset{*}{\longleftrightarrow} \underline{V'\chi^{k'}\chi^{-1}r_i\chi^{k'}\chi^{-1}a}$, where $k' = 2^{|V'|}$. Keep using relations in $R1$ and $R2$; then we have $\underline{V'\chi^{k'}\chi^{-1}r_i\chi^{k'}\chi^{-1}a} = \underline{V'\chi^{k'-1}r_i\chi^{k'-2}\chi a} \leftrightarrow \underline{V'\chi^{k'}r_i\chi^{k'-1}a\chi^2} \overset{*}{\longleftrightarrow} \underline{V\chi^{2k'-1}r_i\chi^{2k'-1}} = \underline{V\chi^{2k'}\chi^{-1}r_i\chi^{2k'}\chi^{-1}}$. This completes the proof. □

REMARK 5.1. We note that exponentially many elementary transformations are required to obtain relations specified in $R6$ from relations in $R1$–$R5$. This is why there is exponential blow up without using the generator ξ and the relations specified in $R6$.

Hence, by Tietze transformation T2, we know that G also has a finite presentation $[A-\{\xi\}; R1, R2, R3, R4, R5]$, which is exactly the same construction as in the Boone lemma. So by Boone's lemma, $(\underline{s}z\underline{\$}\diamond\underline{\tau})\underline{\kappa} = \underline{\kappa}(\underline{s}z\underline{\$}\diamond\underline{\tau})$ in G iff $\underline{s}z\underline{\$} = \underline{h}$ in Γ. The following lemma is therefore straightforward.

LEMMA 5.9. *If* $(\underline{s}z\underline{\$} \diamond \underline{\tau})\underline{\kappa} =_m \underline{\kappa}(\underline{s}z\underline{\$} \diamond \underline{\tau})$ *in* $[A; R]$ *with* $m \leq \eta(|x|)$, *then* $\underline{s}z\underline{\$} = \underline{h}$ *in* Γ, *which implies that* $x \in D$ *by Corollary* 5.4.

From Lemmas 5.6 and 5.9, we get Corollary 5.10.

COROLLARY 5.10. $x \in D$ *iff* $(\underline{s}z\underline{\$} \diamond \underline{\tau})\underline{\kappa} =_m \underline{\kappa}(\underline{s}z\underline{\$} \diamond \underline{\tau})$ *in* $[A; R]$ *with* $m \leq \eta(|x|)$.

We are now ready to finish the proof of Theorem 5.1. By Corollary 5.10, we define a reduction f as follows:

$$f(x) = ([A; R], \underline{s}z\underline{\$}, \underline{\tau}, \underline{\kappa}, 1^{\eta(|x|)}).$$

Clearly, f is one to one. As shown earlier, A contains a constant number of symbols depending on M, and R contains a constant number of relations (some with quantifiers). The length of each coded symbol and the length of each of these relations are $O(\log |x|)$. So $\| A \cup R \|_0 = O(\log |x|)$, and $\| A \cup R \|_1 = O(\log^{O(1)} |x|)$. Note that $|\underline{s}z\underline{\$}| = |x| + O(\log |x|)$. Recall that $z = 1\alpha(x)$. Hence, the probability distribution of $f(x)$, which is proportional to

$$\frac{2^{-(\| A \cup R \|_0 + |\underline{s}z\underline{\$}| + |\underline{\kappa}| + |\underline{\tau}|)}}{(O(1)|\underline{s}z\underline{\$}||\underline{\kappa}||\underline{\tau}|\ \| A \cup R \|_1)^2} \geq \frac{1}{P(|x|)} 2^{-|\alpha(x)|}$$

for a polynomial P, dominates $\mu(x)$. This completes the proof of Theorem 5.1.

6. Some related results.

6.1. Worst-case NP-complete word problem. In the setting of worst-case complexity, we need not worry about preserving distributions for instances between the source problem and the target problem, and so we do not need to use dynamic coding schemes. Also, we can consider a simpler version of the bounded word problem for groups; namely, instead of taking x, y, z as inputs and asking whether $(x^{-1}yx)z$ can be derived from $z(x^{-1}yx)$, we will simply take x and y as inputs and ask whether x can be derived from y.

DEFINITION 6.1. *The* bounded word problem for groups *is the following decision problem.*

Instance: *A finite presentation* $[A; R]$ *of a group, strings* x, y, *and a unary notation* 1^n.

Question: *Is* $x =_k y$ *in* $[A; R]$ *for* $k \leq n$?

Similar to the proof of Theorem 5.1, we can obtain a much simpler proof to the following theorem.

THEOREM 6.2. *The bounded word problem for groups is (worst-case)* NP*-complete.*

6.2. Isomorphisms of average-case NP-complete problems. The issue of isomorphisms of complete sets is an interesting topic. Berman and Hartmanis [3] were the first to study isomorphisms of NP-complete sets. Two sets A and B are said to be p-isomorphic if there is a one to one, onto, p-time computable, and p-time invertible reduction f such that A is reducible to B via f, and B is reducible to A via f^{-1}. Berman and Hartmanis showed that all the known (worst-case) NP-complete sets are p-isomorphic. Wang and Belanger [32] generalized Berman and Hartmanis's notion of isomorphisms and defined a notion of isomorphism for distributional problems. Two distributional problems (A, μ_A) and (B, μ_B) are *p-isomorphic* if there exists a p-time computable and p-time invertible bijection ϕ such that (A, μ_A) is p-time reducible to (B, μ_B) via ϕ, and (B, μ_B) is p-time reducible to (A, μ_A) via ϕ^{-1}. They then showed the following version of the Cantor–Bernstein–Myhill theorem for distributional problems.

LEMMA 6.3 (see [32]). *Let f and g be one-to-one, p-time computable and p-time invertible reductions of (A, μ_A) to (B, μ_B) and (B, μ_B) to (A, μ_A), respectively, such that $f \circ g$ and $g \circ f$ are length increasing.*[7] *Assume also that $\mu_A \approx \mu_B \circ f$ and $\mu_B \approx \mu_A \circ g$, where $\mu \approx \nu$ means $\mu \preceq \nu$ and $\nu \preceq \mu$. Then (A, μ_A) and (B, μ_B) are p-isomorphic.*

Based on Lemma 6.3 and distribution controlling lemma (2), Wang and Belanger [32] showed that all the known average-case NP-complete problems under p-time many–one reductions are indeed p-isomorphic. Based on the proof of Theorem 5.1, we can show that the distributional word problem for groups is p-ismorphic to the distributional halting problem.

THEOREM 6.4. *The distributional word problem for groups is p-ismorphic to the distributional halting problem.*

Proof. We provide a sketch of the proof here. It was shown in [32] that if $A \in$ NP and ν satisfies the hypothesis of distribution controlling lemma (2), then there is a reduction g from (A, ν) to the distributional halting problem (K, μ_K) such that g is one to one, length-increasing, p-time computable, and p-time invertible. Moreover, $\mu \approx \mu_K \circ g$. Let (A, ν) denote the distributional word problem for groups. Then, clearly, ν satisfies the hypothesis of distribution controlling lemma (2).

In the proof of Theorem 5.1, let (D, μ) be (K, μ_K). Replace α by β from distribution controlling lemma (2) for μ_K, and use the same reduction f. Then f is one to one, length increasing, p-time computable, and p-time invertible. Moreover, $\mu_K \approx \nu \circ f$. This completes the proof by Lemma 6.3. □

Acknowledgments. I am grateful to Paul Duvall and Theresa Vaughan for helping me with some of my algebra questions. I thank Jay Belanger and R. Venkatesan for asking some interesting questions regarding an early version of this paper.

REFERENCES

[1] J. BELANGER AND J. WANG, *No NP problems over ranking of distributions are harder*, Theoret. Comput. Sci., 181 (1997), pp. 229–245.
[2] S. BEN-DAVID, B. CHOR, O. GOLDREICH, AND M. LUBY, *On the theory of average case complexity*, J. Comput. System Sci., 44 (1992), pp. 193–219.
[3] L. BERMAN AND J. HARTMANIS, *On isomorphisms and density of NP and other complete sets*, SIAM J. Comput., 6 (1977), pp. 305–321.
[4] A. BLASS AND Y. GUREVICH, *Matrix transformation is complete for the average case*, SIAM J. Comput., 24 (1995) pp. 3–29.
[5] W. BOONE, *The word problem*, Ann. Math., 70 (1959), pp. 207–265.
[6] S. COOK, *The complexity of theorem-proving procedures*, in Proc. of the 3rd Annual Symposium on Theory of Computing, ACM Press, New York, 1971, pp. 151–158.
[7] M. DEHN, *Über unendliche diskontinuíerliche gruppen*, Math. Ann., 71 (1911), pp. 73–77.
[8] M. GAREY AND D. JOHNSON, *Computers and Intractability, A Guide to the Theory of* NP-*Completeness*, W. H. Freeman, San Francisco, CA, 1979.
[9] Y. GUREVICH, *Complete and incomplete randomized NP problems*, in Proc. of the 28th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1987, pp. 111–117.
[10] Y. GUREVICH, *The challenger-solver game: Variations on the theme of P =? NP*, Bull. European Assoc. Theoret. Comput. Sci., 1989, pp. 112–121. (Reprinted in *Current Trends in Theoretical Computer Science*, G. Rozenberg and A. Salomaa, eds., World Scientific, River Edge, NJ, 1993, pp. 245–253.)
[11] Y. GUREVICH, *Matrix decomposition problem is complete for the average case*, in Proc. of the 31st Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 802–811.

[7] A function f is length increasing if $|f(x)| > |x|$.

[12] Y. Gurevich, *Average case completeness*, J. Comput. System Sci., 42 (1991), pp. 346–398.
[13] Y. Gurevich and S. Shelah, *Expected computation time for Hamiltonian path problem*, SIAM J. Comput., 16 (1987), pp. 486-502.
[14] R. Impagliazzo, *A personal view of average-case complexity*, in Proc. of the 10th Conference on Structure in Complexity Theory, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 134–147.
[15] R. Impagliazzo and L. Levin, *No better ways to generate hard NP instances than picking uniformly at random*, in Proc. of the 31st Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 812–821.
[16] D. Johnson, *The* NP*-completeness column: An ongoing guide*, J. Algorithms, 5 (1984), pp. 284–299.
[17] R. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computation, R. Miller and J. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.
[18] K. Ko, *On the definition of some complexity classes of real numbers*, Math. Systems Theory, 16 (1993), pp. 95–109.
[19] L. Levin, *Universal sorting problems*, Problemy Peredachi Informatsii, 9 (1973), pp. 115–116 (in Russian); Problems Inform. Transmission, 9 (1973), pp. 265–266 (in English).
[20] L. Levin, *Average case complete problems*, SIAM J. Comput., 15 (1986), pp. 285–286.
[21] W. Magnus, A. Karrass, and D. Solitar, *Combinatorial Group Theory, Presentation of Groups in Terms of Generators and Relations*, John Wiley & Sons, New York, 1966.
[22] P. Novikov, *On the algorithmic unsolvability of the word problem in group theory*, Trudy Mat. Inst. Steklov., 44 (1955), p. 143.
[23] R. Reischuk and C. Schindelhauer, *Precise average case complexity*, in Proc. of the 10th Annual Symposium on Theoretical Aspects of Computer, Lecture Notes in Comput. Sci. 665, Springer-Verlag, New York, 1993, pp. 650–661.
[24] J. Rotman, *The Theory of Groups*, 3rd ed., Wm. C. Brown Publishers, Dubuque, IA, 1988.
[25] A. Thue, *Problems über Veränderungen von Zeihenreihen nach gegebenen Regeln*, Skr. utzit av Vid Kristiania, I. Mat.-Naturv. Klasse, 10 (1914).
[26] R. Venkatesan, *Average-Case Intractability*, Ph.D. thesis, Boston University, Boston, MA, 1991.
[27] R. Venkatesan and L. Levin, *Random instances of a graph coloring problem are hard*, in Proc. of the 20th Annual Symposium on Theory of Computing, ACM Press, New York, 1988, pp. 217–222.
[28] R. Venkatesan and S. Rajagopalan, *Average case intractability of diophantine and matrix problems*, in Proc. of the 24th Annual Symposium on Theory of Computing, ACM Press, New York, 1992, pp. 632–642.
[29] J. Wang, *Average-case completeness of a word problem for groups*, in Proc. of the 27th Annual Symposium on Theory of Computing, ACM Press, New York, 1995, pp. 325–334.
[30] J. Wang, *Average-case computational complexity theory*, in Complexity Theory Retrospective II, L. Hemaspaandra and A. Selman, Eds., Springer-Verlag, New York, 1997, pp. 295–328.
[31] J. Wang and J. Belanger, *On average-*P *vs. average-*NP, in Complexity Theory: Current Research, K. Ambos-Spies, S. Homer, and U. Schönings, eds., Cambridge University Press, Cambridge, UK, 1993, pp. 47–67.
[32] J. Wang and J. Belanger, *On the NP-isomorphism problem with respect to random instances*, J. Comput. System Sci., 50 (1995), pp. 151–164.
[33] N. Wagner and M. Magyarik, *A public key cryptosystem based on the word problem*, in Advances in Cryptography—Proceedings of CRYPTO'84, Lecture Notes in Comput. Sci. 196, Springer-Verlag, New York, 1985, pp. 19–37.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1284–1297

LINEAR TIME ALGORITHMS FOR DOMINATING PAIRS IN ASTEROIDAL TRIPLE-FREE GRAPHS*

DEREK G. CORNEIL†, STEPHAN OLARIU‡, AND LORNA STEWART§

Abstract. An independent set of three vertices is called an *asteroidal triple* if between each pair in the triple there exists a path that avoids the neighborhood of the third. A graph is asteroidal triple-free (AT-free) if it contains no asteroidal triple. The motivation for this investigation is provided, in part, by the fact that AT-free graphs offer a common generalization of interval, permutation, trapezoid, and cocomparability graphs.

Previously, the authors have given an existential proof of the fact that every connected AT-free graph contains a dominating pair, that is, a pair of vertices such that every path joining them is a dominating set in the graph. The main contribution of this paper is a constructive proof of the existence of dominating pairs in connected AT-free graphs. The resulting simple algorithm, based on the well-known lexicographic breadth-first search, can be implemented to run in time linear in the size of the input, whereas the best algorithm previously known for this problem has complexity $O(|V|^3)$ for input graph $G = (V, E)$. In addition, we indicate how our algorithm can be extended to find, in time linear in the size of the input, all dominating pairs in a connected AT-free graph with diameter greater than 3. A remarkable feature of the extended algorithm is that, even though there may be $O(|V|^2)$ dominating pairs, the algorithm can compute and represent them in linear time.

Key words. algorithms, dominating pairs, asteroidal triple-free graphs, lexicographic breadth-first search

AMS subject classifications. 05C85, 68R10

PII. S0097539795282377

1. Introduction.

Considerable attention has been paid to exploiting algorithmically different aspects of the linear structure exhibited by various families of graphs. Examples of such families include interval graphs [15], permutation graphs [11], trapezoid graphs [6, 10], and cocomparability graphs [13].

The linearity of these four classes is usually described in terms of ad hoc properties of each of these classes of graphs. For example, in the case of interval graphs, the linearity property is traditionally expressed in terms of a linear order on the set of maximal cliques [4, 5]. For permutation graphs the linear behavior is explained in terms of the underlying partial order of dimension 2 [1]; for cocomparability graphs the linear behavior is expressed in terms of topological orderings of transitive orientations of comparability graphs [14], and so on.

As it turns out, the classes mentioned above are all subfamilies of a class of graphs called the asteroidal triple-free graphs (AT-free graphs). An independent set of three vertices is called an *asteroidal triple* if between any pair in the triple there exists a path that avoids the neighborhood of the third. AT-free graphs were introduced over

*Received by the editors March 3, 1995; accepted for publication (in revised form) June 20, 1997; published electronically March 22, 1999.

http://www.siam.org/journals/sicomp/28-4/28237.html

†Department of Computer Science, University of Toronto, Toronto, ON M5S 1A7, Canada (dgc@cs.utoronto.ca). The research of this author was supported by financial assistance from the Natural Sciences and Engineering Council of Canada.

‡Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162 (olariu@cs.odu.edu). The research of this author was supported in part by National Science Foundation grants CCR-9407180 and CCR-9522093 and by ONR grant N00014-97-1-0526.

§Department of Computing Science, University of Alberta, Edmonton, AB T6G 2H1, Canada (stewart@cs.ualberta.ca). The research of this author was supported by financial assistance from the Natural Sciences and Engineering Council of Canada.

three decades ago by Lekkerkerker and Boland [15], who showed that a graph is an interval graph if and only if it is chordal and AT-free. Thus, Lekkerkerker and Boland's result may be viewed as showing that the absence of asteroidal triples imposes the linear structure on chordal graphs that results in interval graphs. Recently, we have studied AT-free graphs with the stated goal of identifying the "agent" responsible for the linear behavior observed in the four subfamilies. Specifically, in [9] we presented evidence that the property of being AT-free is what is enforcing the linear behavior of these classes.

One strong "certificate" of linearity is the existence of a *dominating pair* of vertices, that is, a pair of vertices with the property that every path connecting them is a dominating set. In [9], we gave an existential proof of the fact that every connected AT-free graph contains a dominating pair.

The main contribution of this paper is a constructive proof of the existence of dominating pairs in connected AT-free graphs. A remarkable feature of our approach is that the resulting simple algorithm, based on the well-known lexicographic breadth-first search of [16], can easily be implemented to run in time $O(|V| + |E|)$, where the input is a connected AT-free graph $G = (V, E)$. In addition, our algorithm can be extended to find, in time linear in the size of the input, all dominating pairs in a connected AT-free graph with diameter greater than 3.

It should be noted that the fastest algorithm known to us, which recognizes whether or not a graph $G = (V, E)$ is AT-free, runs in time $O(|V|^3)$.

To put our result in perspective, we observe that previously, the most efficient algorithm for finding a dominating pair in a graph $G = (V, E)$ was the straightforward $O(|V|^3)$ algorithm described in [2].

For each of the four families mentioned above, vertices that occupy the extreme positions in the corresponding intersection model [12] constitute a dominating pair. It is interesting to note, however, that a linear time algorithm for finding a dominating pair was not previously known, even for cocomparability graphs, a strict subclass of AT-free graphs.

The remainder of this paper is organized as follows. Section 2 contains some relevant terminology and background. Section 3 is a description of the lexicographic breadth-first search algorithm of [16] along with some properties of that algorithm. In section 4 we present an algorithm which finds a dominating pair in a connected AT-free graph. In sections 5 and 6, we show how to extend the dominating pair algorithm to find all dominating pairs in a connected AT-free graph with sufficiently large diameter. Section 7 contains our conclusions.

2. Background. All the graphs in this work are finite with no loops or multiple edges. In addition to standard graph theoretic terminology compatible with [3], we shall define some new terms. We let $d(v)$ denote the *degree* of vertex v; $d(u, v)$ denotes the *distance* between vertices u and v in a graph, that is, the number of edges on a shortest path joining u and v. In addition, we let $\mathrm{diam}(G)$ denote the *diameter* of the graph G, that is, $\max_{u,v \in G} d(u, v)$. Two vertices u and v with $d(u, v) = \mathrm{diam}(G)$ are said to *achieve the diameter*. Given a graph $G = (V, E)$ and a vertex x, we let $N(x)$ denote the set of neighbors of x; $N'(x)$ denotes the set of neighbors of x in the complement $\overline{G}$ of G.

Let $\pi = v_1, v_2, \ldots, v_k$ be a path of graph G. If the subgraph of G induced by $\{v_1, v_2, \ldots, v_k\}$ has exactly $k - 1$ edges, i.e., none other than the edges of the path, then π is said to be an *induced*, or *chordless*, path. All the paths in this work are assumed to be induced unless stated otherwise. We refer to a path joining vertices x

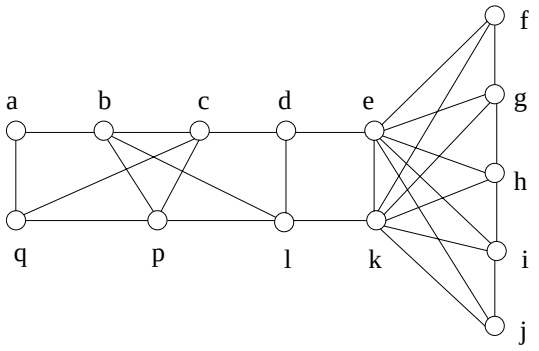


FIG. 1. *A connected AT-free graph G.*

and y as an x,y-path. We say that a vertex u *intercepts a path* π if u is adjacent to at least one vertex on π; otherwise, u is said to *miss* π. Let $G = (V, E)$ be a graph, π a path in G, x a vertex of G, and X a subset of V. Let $V(\pi)$ be the vertices of G that are on the path π. We shall use the following notation: $\pi - x$ refers to the subgraph of G induced by the vertices $V(\pi) - \{x\}$, $\pi + x$ refers to the subgraph of G induced by the vertices $V(\pi) \cup \{x\}$, and $\pi \cup X$ refers to the subgraph of G induced by the vertices $V(\pi) \cup X$.

For a connected AT-free graph with a pair of vertices x, y we let $D(x, y)$ denote the set of vertices that intercept all x,y-paths. Note that (x, y) is a dominating pair if and only if $D(x, y) = V$. We say that vertices u and v are *unrelated with respect to* x if $u \notin D(v, x)$ and $v \notin D(u, x)$. A vertex x of an AT-free graph G is called *pokable* if the graph obtained from G by adding a pendant vertex adjacent to x is AT-free. It is not hard to see that if an AT-free graph G contains no unrelated vertices with respect to x, then x is pokable. A *pokable dominating pair* is a dominating pair such that both vertices are pokable. A vertex x is a *pokable dominating pair vertex* if x is pokable and there exists y such that (x, y) is a dominating pair. To illustrate these definitions, consider the graph $G = (V, E)$ of Figure 1. In this graph, $D(c, l) = \{b, c, d, k, l, p, q\}$, $D(c, e) = V \backslash \{a\}$, and $D(a, e) = D(q, i) = V$. Any pair consisting of one vertex from $\{a, q\}$ and one vertex from $\{e, f, g, h, i, j, k\}$ is a dominating pair; a is pokable and h is not pokable (adding a pendant vertex h' adjacent to h would create the AT $\{f, j, h'\}$).

3. Lexicographic breadth-first search. Our dominating pair algorithm invokes Procedure LBFS (short for lexicographic breadth-first search), which, when given a connected graph G and a vertex x of G, returns a numbering of the vertices of G. We reproduce below the details of LBFS from [16].

PROCEDURE LBFS(G, x).
{Input: a connected graph $G = (V, E)$ and a distinguished vertex x of G;
Output: a numbering σ of the vertices of G}
begin
  label$(x) \leftarrow |V|$;
  for each vertex v in $V - \{x\}$ **do**
    label$(v) \leftarrow \Lambda$;
  for $i \leftarrow |V|$ **downto** 1 **do begin**
    pick an unnumbered vertex v with (lexicographically) the largest label;
    $\sigma(v) \leftarrow i$; {assign to v number i}
    for each unnumbered vertex u in $N(v)$ **do**

append i to label(u)
end
end; {LBFS}

Notice that the numbering returned by LBFS is not unique. One numbering that could result from LBFS(G, q), where G is the graph of Figure 1, is $\sigma(q) = 14$, $\sigma(p) = 13$, $\sigma(c) = 12$, $\sigma(a) = 11$, $\sigma(b) = 10$, $\sigma(l) = 9$, $\sigma(d) = 8$, $\sigma(k) = 7$, $\sigma(e) = 6$, $\sigma(i) = 5$, $\sigma(h) = 4$, $\sigma(j) = 3$, $\sigma(g) = 2$, $\sigma(f) = 1$.

A few definitions relating to LBFS are in order at this point. Let x be an arbitrary vertex of a connected graph G, and consider running LBFS(G, x). For vertices a, b of G we write $a \prec b$ whenever $\sigma(a) < \sigma(b)$, and we shall say that b is *larger* than a. To make the notation more manageable we shall sometimes write $v_1 \prec v_2 \prec \cdots \prec v_k$ as shorthand for $v_1 \prec v_2, v_2 \prec v_3, \ldots, v_{k-1} \prec v_k$. We shall denote by $\lhd$ the lexicographic total order of the set of LBFS labels. We let $\lambda(a, b)$ denote the label of a when b was about to be numbered. Given vertices a, b, c with $a \prec c$ and $b \prec c$, we shall say that a and b *are tied at* c if $\lambda(a, c) = \lambda(b, c)$. Given a vertex y, an a,b-path is said to be *y-majorizing* if all the vertices on the path are larger than y.

We assume that $G = (V, E)$ is an arbitrary connected graph and that LBFS(G, x) has been invoked, where x is an arbitrary vertex of G. The following fundamental properties of LBFS will be used later.

PROPOSITION 3.1. *Let a, b, and c be vertices of G satisfying $a \prec b$, $b \prec c$, $ac \in E$, and $bc \notin E$. Then there exists a vertex d in G adjacent to b but not to a and such that $c \prec d$.*

Proof. The existence of d follows immediately from the observation that when b was about to be processed by LBFS it could not have been tied with a. Since a inherited c's label, b must have inherited the label of a larger vertex nonadjacent to a; this is d. □

PROPOSITION 3.2 (monotonicity property). *Let a, b, c, and d be vertices of G such that $a \prec c$ or $a = c$, $b \prec c$ or $b = c$, and $c \prec d$. If $\lambda(a, d) \lhd \lambda(b, d)$, then $\lambda(a, c) \lhd \lambda(b, c)$.*

Proof. The proof follows directly from the lexicographic ordering of labels. □

LEMMA 3.3. *Let a, b, b', and c be vertices of G such that $a \prec b \prec c \prec b'$, $bb' \in E$, and $b'c \notin E$. Then a and c cannot be tied at b'.*

Proof. By Proposition 3.1 applied to vertices b, c, and b' we find a vertex c' adjacent to c but not to b and such that $b' \prec c'$. Write $C = \{t \mid tc \in E, tb \notin E, b' \prec t\}$. Clearly, $c' \in C$. In fact, we select c' to be the *largest* vertex in C.

If the statement is false, then a and c are tied at b'. Since $b' \prec c'$ and since $cc' \in E$, we must have $ac' \in E$. Now, Proposition 3.1 applied to vertices a, b, *and* c' yields a vertex b'' adjacent to b but not to a such that $c' \prec b''$.

Since $b' \prec b''$, the assumption that a and c are tied at b' guarantees that b'' is not adjacent to c. Therefore, Proposition 3.1 can be applied to vertices b, c, and b'', yielding a vertex c'' adjacent to c but not to b and such that $b'' \prec c''$. Since $b' \prec c' \prec b'' \prec c''$, it must be that $c'' \in C$, contradicting that c' is the largest vertex in C. □

LEMMA 3.4. *Let y, a, and b be pairwise nonadjacent vertices of G such that $y \prec a$ and $a \prec b$. If a and y are not tied at b, then y misses a y-majorizing a,b-path.*

Proof. Assume that a and y are not tied at b. We must exhibit a y-majorizing a,b-path missed by y.

Since $y \prec a$, $\lambda(y, a) \lhd \lambda(a, a)$ or $\lambda(y, a) = \lambda(a, a)$. Therefore, by the monotonicity property (Proposition 3.2), $\lambda(y, b) \lhd \lambda(a, b)$ or $\lambda(y, b) = \lambda(a, b)$. Now, since a and y

are not tied at b, $\lambda(y,b) \lhd \lambda(a,b)$. Consequently, we find a vertex a_1 adjacent to a but not to y and such that $b \prec a_1$. (Vertex a_1 is chosen to be the largest satisfying these conditions.) We may assume that a_1 is not adjacent to b, since otherwise the path a, a_1, b is the desired y-majorizing path.

Now, Lemma 3.3 guarantees that b and y cannot be tied at a_1. Thus, we find a vertex b_1 adjacent to b but not to y and such that $a_1 \prec b_1$. (As before, we select as b_1 the largest vertex with this property.) Trivially, we may assume that b_1 is adjacent to neither a nor a_1; else we have the desired y-majorizing path. Again, Lemma 3.3 tells us that a_1 and y cannot be tied at b_1 and so we find a vertex a_2 adjacent to a_1 but not to y and such that $b_1 \prec a_2$. (As before, we select as a_2 the largest vertex with this property.) It is easy to verify that a_2 is not adjacent to a (by the choice of a_1), b, or b_1.

Continuing as above, we obtain two chordless y-majorizing paths $a = a_0, a_1, a_2, \ldots$ and $b = b_0, b_1, b_2, \ldots$, both missed by y. If no vertex on the first path is adjacent to a vertex on the second one, then the paths are infinite, contradicting that G is finite. Therefore, such an adjacency must exist, yielding the desired a,b-path. ∎

4. The dominating pair algorithm. Our dominating pair algorithm takes as input a connected AT-free graph G and returns a pokable dominating pair of G. The algorithm provides a constructive proof of the existence of pokable dominating pairs in connected AT-free graphs. (An existential proof of this fact was given in [9].)

The four properties of LBFS specified in the preceding section hold for every connected graph G. The proof of correctness of the dominating pair algorithm relies on two additional properties of LBFS which hold when the input graph is a connected AT-free graph. We present these properties next.

THEOREM 4.1. *Let $G = (V, E)$ be a connected AT-free graph and let x and y be arbitrary vertices of G. Let $\prec$ be the vertex ordering corresponding to a numbering produced by LBFS(G, x). The subgraph of G induced by y and all vertices z with $y \prec z$ contains no unrelated vertices with respect to y.*

Proof. First, we give an overview of the proof. The existence of such unrelated vertices, u and v, and the fact that they are numbered before y in an LBFS from x, would imply that u and v are connected by a path through x. If y misses such a path, then $\{y, u, v\}$ is an AT.

In particular, if the statement is false, we find a vertex y and vertices u, v with $y \prec u \prec v$, such that u and v are unrelated with respect to y. This implies the existence of chordless paths $\pi(y,u) : y = u_1, u_2, \ldots, u_p = u$ missed by v, and $\pi(y,v) : y = v_1, v_2, \ldots, v_q = v$ missed by u, with the vertices on both paths, except for y, numbered by LBFS before y. We claim that

$$u \prec v_3. \tag{4.1}$$

If (4.1) is false, then $v_3 \prec u$ and $u \prec v$, and we must find a subscript i $(3 \leq i \leq q-1)$ such that $v_i \prec u$ and $u \prec v_{i+1}$. Now, Lemma 3.3 tells us that u and y cannot be tied at v_{i+1}. In turn, Lemma 3.4 guarantees the existence of a y-majorizing u,v_{i+1}-path missed by y. This path extends trivially to a y-majorizing u,v-path, implying that $\{y, u, v\}$ is an AT. Thus, (4.1) must hold.

Next, we claim that

$$u \text{ and } y \text{ are tied at } v_3. \tag{4.2}$$

The contrary would imply, by virtue of Lemma 3.4, the existence of a y-majorizing

u,v_3-path missed by y. This extends easily into a y-majorizing u,v-path missed by y, implying that $\{y, u, v\}$ is an AT. Thus, (4.2) must hold.

We note that $v_2 \prec v_3$; otherwise, since v_2 is adjacent to y and not to u, we would contradict (4.2). Further, we claim that

$$u \prec v_2. \tag{4.3}$$

Otherwise, by (4.1) we have $v_2 \prec u$ and $u \prec v_3$. Now, Lemma 3.3 specifies that u and y cannot be tied at v_3, contradicting (4.2). Thus, (4.3) must be true.

Proposition 3.1 applied to vertices $y \prec u$ and $u \prec v_2$ guarantees the existence of a vertex u' adjacent to u but not to y and such that $v_2 \prec u'$. Since u and y are tied at v_3, it must be the case that $y \prec u$, $u \prec v_2$, $v_2 \prec u'$, and $u' \prec v_3$. If u' is adjacent to v_3, then we have a u,v-path missed by y, contradicting that the graph is AT-free. Thus, u' is not adjacent to v_3. But now, Lemma 3.3 guarantees that y and u' cannot be tied at v_3. Further, Lemma 3.4 tells us that there must exist a y-majorizing u',v_3-path missed by y. This path extends in the obvious way to a y-majorizing u,v-path missed by y, contradicting that the graph is AT-free. This completes the proof of Theorem 4.1. □

We observe that, if G contains no unrelated vertices with respect to vertex v, then v is pokable. This observation and Theorem 4.1 combined imply that each vertex y of G is pokable in the subgraph of G induced by y and all vertices z with $y \prec z$. In particular, the last vertex numbered by LBFS(G, x) is pokable in G.

One additional theorem about LBFS, specialized to connected AT-free graphs, will lead to the dominating pair algorithm.

THEOREM 4.2. *Let $G = (V, E)$ be a connected AT-free graph and suppose that G contains no vertices unrelated with respect to vertex x of G. Let $\prec$ be a vertex ordering corresponding to a numbering produced by LBFS(G, x). Then, for all vertices u, v in V with $u \prec v$, $v \in D(u, x)$.*

Proof. The argument proceeds by noting that if $v \notin D(u, x)$, then there is a u,x-path missed by v and no v,x-path missed by u (since u and v cannot be unrelated with respect to x). However, an LBFS from x would number u before v, contradicting the conditions of the theorem.

Assume that the theorem is false and let v be the largest vertex in V for which there exists a vertex u with $u \prec v$ and $v \notin D(u, x)$. We now select a specific path π and a vertex u with $u \prec v$ such that π is a u,x-path missed by v. Let U be the set of all vertices u such that $u \prec v$ and $v \notin D(u, x)$ and let $\mathcal{P}$ be the set of all chordless u,x-paths in G that are missed by v. Among all minimum length paths in $\mathcal{P}$, we choose π to be the one that extends to the largest possible vertex at each step. Now u is the endpoint of π that is in the set U.

Formally, let $\mathcal{P}_{\mathcal{M}}$ be the subset of $\mathcal{P}$ consisting of all minimum length paths of $\mathcal{P}$. For paths $P = p_1, p_2, \ldots, p_k$ and $P' = p'_1, p'_2, \ldots, p'_k$ in $\mathcal{P}_{\mathcal{M}}$, we say that P' is greater than P if there exists a subscript i, $1 \le i \le k$, such that $\sigma(p'_j) = \sigma(p_j)$ for all $1 \le j < i$ and $\sigma(p'_i) > \sigma(p_i)$. Clearly, "greater than" is a total order on $\mathcal{P}_{\mathcal{M}}$. We choose $\pi : u = u_1, u_2, \ldots, u_k = x$ to be the unique greatest element of $\mathcal{P}_{\mathcal{M}}$.

Observe that $u_1 \prec v$ and $v \prec u_2$; otherwise we contradict the fact that π is in $\mathcal{P}_{\mathcal{M}}$. Now Proposition 3.1 guarantees the existence of a vertex v_2 adjacent to $v = v_1$ but not to u_1 and such that $u_2 \prec v_2$.

It is easily seen that v_2 is nonadjacent to u_i for all $i > 2$, since otherwise u and v are unrelated with respect to x. This immediately implies that $v_2 \prec u_3$ (otherwise we contradict the choice of v) and $u_2 v_2 \in E$ (otherwise we contradict the choice of both u and v).

Now apply Proposition 3.1 to vertices u_2, v_2, and u_3; we find a vertex v_3 adjacent to v_2 but not to u_2 and such that $u_3 \prec v_3$.

Since $u_2 \prec v_3$, v_3 cannot miss the path $u_2, u_3, \ldots, u_k = x$. Let t ($t \geq 3$) be the largest subscript for which $v_3u_t \in E$. Now v_3 must be adjacent to u_1; else u_1 and v_1 are unrelated with respect to x. (The v_1,x-path missed by u_1 is $v_1, v_2, v_3, u_t, \ldots, x$.) Note also that v_3 must be adjacent to v_1; otherwise we contradict the choice of π. (To see this, note that u_1, v_3 extends to a minimum length chordless u,x-path via u_t.) Now, $t = 3$; else the assignment $v \leftarrow u_2$ and $u \leftarrow v$ contradicts the initial choice of u and v.

Now assume that we have constructed a sequence $v = v_1, v_2, \ldots, v_i$ of vertices such that v_i ($i \geq 3$) satisfies the following conditions:

(a) $v_iv_{i-1}, v_iu_i \in E$;
(b) $u_i \prec v_i$ and $v_iu_{i-1} \notin E$;
(c) $v_iu_j \notin E$ for $j > i$;
(d) $v_iu_1, v_iv_1 \in E$.

We argue that there exists a vertex v_{i+1} satisfying conditions (a)–(d) with $i + 1$ in place of i. For this purpose, note that u_{i+1} exists since $u_i \prec v_i$ implies that $u_i \neq x$. Now, $v_i \prec u_{i+1}$ since otherwise the assignment $v \leftarrow v_i$ and $u \leftarrow u_{i+1}$ contradicts the initial choice of u and v.

Now, by (c), Proposition 3.1 applied to vertices u_i, v_i, and u_{i+1} guarantees the existence of a vertex v_{i+1} adjacent to v_i but not to u_i and such that $u_{i+1} \prec v_{i+1}$. Thus, (b) is verified. Let t be the largest subscript for which v_{i+1} is adjacent to u_t. (t exists and $t \geq i+1$, since otherwise the assignment $v \leftarrow v_{i+1}$ and $u \leftarrow u_i$ contradicts the initial choice of u and v.)

Note that u_{i-1} is adjacent to v_{i+1}, since if it is not, then u_{i-1} and v_1 are unrelated with respect to x. (By (b) and (d), the path contained in $v_1, v_i, v_{i+1}, u_t, u_{t+1}, \ldots, x$ is missed by u_{i-1}.) Also v_1 is adjacent to v_{i+1}, since otherwise we contradict the choice of π by going down π and picking the first edge u_jv_{i+1}, which we know exists (in particular, we know that $u_{i-1}v_{i+1} \in E$) and then going to u_t and on to x. Now u_1 is adjacent to v_{i+1}, since otherwise u_1 and v_1 are unrelated with respect to x (the path $v_1, v_{i+1}, u_t, \ldots, x$ would be missed by u_1). Thus, (d) holds.

Note that $t = i+1$; otherwise we should have picked u_i instead of v (for u_i misses the path u_1, v_{i+1}, u_t, etc.). Thus, both (a) and (c) hold.

But now we have reached a contradiction: v_k must exist and it must be that $x = u_k \prec v_k$, which is absurd. □

Theorem 4.2 implies that if G contains no vertices unrelated with respect to x, then (x, y) is a dominating pair in the subgraph of G induced by y and all vertices z with $y \prec z$. In particular, x and the last vertex numbered by LBFS(G, x) constitute a dominating pair of G.

We are now in a position to spell out the details of the dominating pair algorithm.

PROCEDURE DP(G).
{Input: a connected AT-free graph G;
Output: (y, z) a pokable dominating pair of G}
begin
  choose an arbitrary vertex x of G;
  if $N'(x) = \emptyset$ **then** return (x, x);
  LBFS(G, x);
  let y be the vertex numbered last by LBFS(G, x);
  LBFS(G, y);

let z be the vertex numbered last by LBFS(G, y);
return(y, z)
end; {DP}

As an example, we refer again to the graph G of Figure 1. We saw earlier that a possible numbering resulting from LBFS(G, q) corresponds to the ordering

$$f \prec g \prec j \prec h \prec i \prec e \prec k \prec d \prec l \prec b \prec a \prec c \prec p \prec q.$$

LBFS(G, f) may produce the ordering

$$a \prec q \prec b \prec p \prec c \prec l \prec d \prec j \prec i \prec h \prec g \prec k \prec e \prec f.$$

Thus, DP(G) may output the pokable dominating pair (a, f).

Finally, we state the following result.

THEOREM 4.3. *Procedure DP finds a pokable dominating pair in a connected AT-free graph, $G = (V, E)$, in $O(|V| + |E|)$ time.*

Proof. Clearly, (x, x) is a pokable dominating pair of G if $N'(x) = \emptyset$. Otherwise, by Theorem 4.1, G contains no unrelated vertices with respect to y and, hence, by Theorem 4.2, (y, z) is a dominating pair of G. In addition, Theorem 4.1 implies that both y and z are pokable in G. It is clear that a linear time implementation is possible (see [16] for details of a linear time implementation of LBFS). □

5. Computing dominated sets. Since dominating pairs play an important role in the study of AT-free graphs and, intuitively, correspond to the extreme endpoints of the linear structure of the graph, it is interesting to ask whether the above algorithm can be the basis of an efficient algorithm to find all of the dominating pairs in a connected AT-free graph. It turns out that we can indeed extend the algorithm to efficiently find all dominating pairs in a connected AT-free graph provided that the graph has diameter greater than 3. The diameter restriction is a consequence of Theorem 6.1, which states that the set of dominating pairs is precisely the Cartesian product of two subsets of vertices X and Y, provided the diameter of the graph is greater than 3. Thus, by computing X and Y, we have a linear-sized implicit representation of all dominating pairs. Such representations do not seem to hold for AT-free graphs with diameter less than 4.

Perhaps even more interesting in its own right, and a step in the direction of computing all dominating pairs, is a method that, given a connected AT-free graph G and a pokable dominating pair vertex x of G, computes the sets $D(v, x)$ for all vertices v of G. (Recall that $D(v, x)$ denotes the set of vertices that intercept all v,x-paths.) We describe this method first and, in the next section, we show how the information obtained can be used to compute all the dominating pairs in a connected AT-free graph with diameter greater than 3.

In order to understand our approach, which relies on a variant of LBFS, let us examine a few details of an efficient LBFS implementation. We use an adjacency list representation of a graph. Additionally, unnumbered vertices are stored in another data structure, specifically, a list of lists. At each stage of the algorithm, each list contains unnumbered vertices having the same label (i.e., vertices that are tied at the current stage), and lists are stored in decreasing lexicographic order of the corresponding labels. Thus, the largest label can be found in constant time. Let us examine the evolution of the list of lists during the execution of LBFS(G, x) where $G = (V, E)$. Initially, there are two lists: one contains the vertex x and corresponds to the label $|V|$, and the other contains all other vertices of G and corresponds to the label Λ.

Each time a new vertex u is numbered, it is removed from its list and its number is appended to the labels of its unnumbered neighbors. Each list that contains both an unnumbered neighbor of u and a vertex that is not adjacent to u, is split into two lists, one for the original label and one corresponding to the original label with $\sigma(u)$ appended. The first list follows the second in the ordered list of lists. It is important to note that, by the monotonicity property (Proposition 3.2), the relative order of the lists never changes. In order to access and move the neighbors of u in $O(d(u))$ time, an array of $|V|$ pointers indicates the location of each unnumbered vertex within the list of lists, and the lists are doubly linked.

We now return to the problem at hand, namely, given a connected AT-free graph $G = (V, E)$ and a pokable dominating pair vertex x of G, we wish to compute the sets $D(v, x)$ for all vertices v of G. We will modify LBFS to obtain a linear time algorithm for this problem. To begin, we observe that the sum of the cardinalities of the sets $D(v, x)$, for all $v \in V$, may be $O(|V|^2)$, and hence, a linear time algorithm must use an implicit representation of these sets. We handle this as follows: for each vertex v we compute a number, span(v), $1 \leq \text{span}(v) \leq \sigma(v)$, with the property that $D(v, x) = \{u | \sigma(u) \geq \text{span}(v)\} \cup N^-(v)$, where σ is a numbering resulting from LBFS(G, x) and $N^-(v) = N(v) \cap \{w | \sigma(w) < \sigma(v)\}$. Thus, span$(v)$ indicates an interval, with respect to σ, of vertices to be included in $D(v, x)$. When all vertices of a set $W \subseteq V$ have the same span value, we refer to that value as span(W).

The values of span(v) for all vertices v are computed incrementally. It is not necessary to update the values individually because all vertices on the same list will have the same span value. Thus, we store span values for each list, rather than for each vertex. The two initial lists have span values of $|V|$. Just before a vertex is numbered, the span value of its list is updated. When a list is split, the two new lists inherit the span value of the original list. A span value is assigned to an individual vertex when that vertex is finally numbered. As we maintain the lists, we store the size of each list. Thus, for the list W in each iteration, W, $|W|$, and span(W) can be accessed in constant time. Furthermore, over all iterations, all updates of span values can be accomplished in linear time. Thus, the overall complexity of the algorithm below is $O(|V| + |E|)$.

Procedure DSETS is a modified LBFS which computes implicit representations of $D(v, x)$ for all $v \in V$.

PROCEDURE DSETS(G, x).
{Input: a connected AT-free graph $G = (V, E)$ and a pokable dominating pair vertex x of G;
Output: a numbering σ of the vertices of G and, for each vertex v, span(v) such that $D(v, x) = \{u | \sigma(u) \geq \text{span}(v)\} \cup N^-(v)$ }
begin
label$(x) \leftarrow |V|$;
for each vertex v in $V - \{x\}$ **do**
label$(v) \leftarrow \Lambda$;
$W_1 \leftarrow \{x\}$; $W_2 \leftarrow V - \{x\}$; {Initialize two lists}
span$(W_1) \leftarrow$ span$(W_2) \leftarrow |V|$;
for $i \leftarrow |V|$ **downto** 1 **do begin** {main for loop}
pick an unnumbered vertex v with (lexicographically) the largest label;
let W be the list containing v and all vertices tied with v;
span$(W) \leftarrow$ min {span(W), $i + 1 - |W|$ };
remove v from W;

```
    σ(v) ← i; {assign to v number i}
    span(v) ← span(W);
    for each unnumbered vertex u in N(v) do
      append i to label(u);
    split lists as necessary so that there is a one-to-one correspondence between the
    resulting set of lists and the vertex labels
    end {main for loop}
  end; {DSETS}
```

Let us look again at the graph G of Figure 1. Suppose that the numbering returned by DSETS(G, q) corresponds to the ordering

$$f \prec g \prec j \prec h \prec i \prec e \prec k \prec d \prec l \prec b \prec a \prec c \prec p \prec q.$$

Now the span values computed by DSETS(G, q) will be

v	a	b	c	d	e	f	g	h	i	j	k	l	p	q
span(v)	11	10	11	8	6	1	1	1	1	1	7	9	11	14

It is easy to verify that the corresponding sets are correctly represented in this case. For example, $D(q, q) = \{a, c, p, q\}$, $D(l, q) = \{a, b, c, d, k, l, p, q\}$, and $D(e, q) = V$.

Before presenting the proof of correctness of Procedure DSETS, we examine the relationship between vertices x (such that there are no vertices unrelated with respect to x) and pokable dominating pair vertices. The following lemma acts as a bridge between the results of section 4 and the subsequent results of this section.

LEMMA 5.1. *Let G be a connected AT-free graph and let x be an arbitrary vertex of G. Then G contains no vertices unrelated with respect to x if and only if x is a pokable dominating pair vertex of G.*

Proof. The "only if" part follows from Theorem 4.2 and the fact that if G contains no vertices unrelated with respect to x, then x is pokable (since no AT can be created by adding a pendant vertex adjacent to x). To prove the "if" part, let y be a vertex of G such that (x, y) is a dominating pair of G, and consider unrelated vertices u and v with respect to x. Since (x, y) is a dominating pair, u and v intercept every path joining x and y. Let π be an x,y-path and let u' and v' be vertices on π adjacent to u and v, respectively. Trivially, both u' and v' are distinct from x. But now there exists a u,v-path in G that does not contain x (this path contains vertices u', v' and a subpath of π), implying that x is not pokable. □

The correctness of Procedure DSETS relies on the following theorem.

THEOREM 5.2. *Let x be a pokable dominating pair vertex of a connected AT-free graph $G = (V, E)$. For every vertex v of G, $D(v, x) = \{u | \sigma(u) \geq \text{span}(v)\} \cup N^-(v)$.*

Proof. Informally, notice that span(v) is the smallest numbered vertex that is tied with v at any point in the algorithm. Intuitively, all vertices in $\{u | \sigma(u) \geq \text{span}(v)\}$, as well as all neighbors of v, are in $D(v, x)$. The proof demonstrates that $D(v, x)$ is exactly equal to this set of vertices.

Formally, let $\sigma : V \mapsto \{1, 2, \ldots, n\}$ be a numbering returned by DSETS(G, x), let v be an arbitrary vertex of V, and let $D(v) = \{u | \sigma(u) \geq \text{span}(v)\} \cup N^-(v)$.

Our plan is to prove that $D(v) = D(v, x)$. To implement this plan we first prove that $D(v) \subseteq D(v, x)$. Suppose that $D(v) \not\subseteq D(v, x)$, and let u be a vertex in $D(v)$ but not in $D(v, x)$. Now $uv \notin E$ and, by Theorem 4.2 and Lemma 5.1, $\sigma(u) < \sigma(v)$. Thus, by the algorithm, $\text{span}(v) \leq \sigma(u) < \sigma(v)$. (To see this, notice that during the

iteration of the main for loop in which vertex v is numbered, the list W that contains v receives a span value less than or equal to $i+1-|W|$. Since W contains v, $|W| \geq 1$ and hence $\text{span}(W) \leq i$. Now, the desired inequality follows, since $\sigma(v)$ is assigned the value i and $\text{span}(v)$ is assigned the value $\text{span}(W)$.)

Let w be the vertex being processed when $\text{span}(v)$ was first set to its final value. (It could be that $w = v$.) Then u, v, and w were tied at w; that is, $N(u) \cap \{t|\sigma(t) > \sigma(w)\} = N(v) \cap \{t|\sigma(t) > \sigma(w)\} = N(w) \cap \{t|\sigma(t) > \sigma(w)\}$. Since $u \notin D(v,x)$ there exists a v,x-path $\pi : v = v_1, v_2, \ldots, v_k = x$ missed by u. By Theorem 4.2 and Lemma 5.1, all vertices of π are larger than u. Let i be the greatest index such that $\sigma(v_i) < \sigma(w)$. Clearly, $i < k$, since $\sigma(w) < \sigma(x)$. Now $\sigma(u) < \sigma(v_i) < \sigma(w)$ and thus, by the monotonicity property (Proposition 3.2), v_i was tied with u, v, and w at w. But v_i is adjacent to v_{i+1} with $\sigma(w) < \sigma(v_{i+1})$; thus, v_{i+1} belongs to $N(w) \cap \{t|\sigma(t) > \sigma(w)\}$ and is therefore adjacent to u, contradicting that π is missed by u. Thus, $D(v) \subseteq D(v,x)$.

We now prove that $D(v,x) \subseteq D(v)$, thereby completing the proof of the theorem. Suppose that $D(v,x) \not\subseteq D(v)$. Let u be a vertex in $D(v,x)$ but not in $D(v)$. Clearly, $uv \notin E$ and $\sigma(u) < \text{span}(v) \leq \sigma(v)$.

If at any stage u and v are tied with the vertex being processed, then $\text{span}(v)$ is set to a value less than or equal to $\sigma(u)$, and $\text{span}(v)$ is never increased. Thus, since $\text{span}(v) > \sigma(u)$, we know that u and v are never tied with the vertex being processed. Let z be the largest neighbor of v. Since u cannot be adjacent to any vertex greater than z (else $\sigma(u) < \sigma(v)$ is contradicted), and since there is a z,x-path consisting entirely of z and vertices larger than z (by the breadth-first nature of the search), u must be adjacent to z. (Otherwise we contradict the fact that $u \in D(v,x)$.) Now let Z be the vertices of $N(v) \cap \{t|\sigma(t) > \sigma(v)\}$ which have a neighbor greater than z, and let Z' be the remaining vertices of $N(v) \cap \{t|\sigma(t) > \sigma(v)\}$. Clearly, $z \in Z$. Note that u is adjacent to all vertices of Z; otherwise there is a v,x-path missed by u, which is a contradiction. We observe that for every $z \in Z$ and every $z' \in Z'$, it must be that $z' \prec z$. This follows from the monotonicity property (Proposition 3.2) and by the fact that, at z, all vertices of Z' have labels lexicographically less than all vertices of Z. Finally, all vertices of Z' are adjacent to all vertices of Z, since any vertex of Z' nonadjacent to a vertex of Z would have been processed after v. But when the first vertex of Z' is processed, all vertices of Z', u, and v are tied, contradicting our earlier statement that u and v are not tied at any stage. This completes the proof. □

Theorem 5.2, along with the discussion preceding Procedure DSETS, implies the following result.

THEOREM 5.3. *Let $G = (V,E)$ be a connected AT-free graph and let x be a pokable dominating pair vertex of G. Procedure DSETS computes implicit representations for the sets $D(v,x)$ for every vertex v of G in $O(|V| + |E|)$ time.*

6. Computing all dominating pairs. We now describe how to use the span values computed by Procedure DSETS to compute all dominating pairs in a connected AT-free graph with sufficiently large diameter. Our algorithm relies on the following result, the proof of which appears in [9].

THEOREM 6.1 (see [9]). *Let G be a connected AT-free graph with diam$(G) > 3$. There exist nonempty, disjoint sets X and Y of vertices of G such that (x,y) is a dominating pair if and only if $x \in X$ and $y \in Y$.*

We note that Theorem 6.1 is best possible in the sense that, for AT-free graphs of diameter less than 4, the sets X and Y are not guaranteed to exist. To wit, C_5 and the graph of Figure 2 provide counterexamples of diameters 2 and 3, respectively.

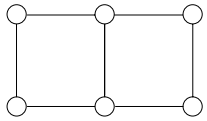


FIG. 2. *An AT-free graph of diameter* 3 *for which the sets* X *and* Y *do not exist.*

Procedure ALL-DPs takes as input a connected AT-free graph $G = (V, E)$ with diameter greater than 3, and returns X and Y, subsets of V such that (x, y) is a dominating pair if and only if $x \in X$ and $y \in Y$. Procedure DSETS is an integral part of Procedure ALL-DPs.

We begin with an informal description of Procedure ALL-DPs. The first step is to find a pokable dominating pair vertex, which is done by LBFS in linear time. Then, Procedure DSETS(G, x) computes span(v) for all vertices v in linear time. From the resulting span values, and by Theorem 6.1, it is easy to see how to proceed.

Now, Y is the set of all vertices y with $D(y, x) = V$ (whether or not $D(y, x) = V$ can be computed in $O(d(y))$ time by scanning the adjacency list of y and checking whether all vertices w with $\sigma(w) < \text{span}(y)$ are adjacent to v). Finally, we call DSETS(G, y), where y is the vertex that was numbered last by DSETS(G, x). The new set of span values can be used to compute the set X in a manner identical to the above method for computing Y. We now state the procedure more precisely.

PROCEDURE ALL-DPs(G).
{Input: connected AT-free graph $G = (V, E)$ with diam$(G) > 3$;
Output: $X \subseteq V$ and $Y \subseteq V$ such that (x, y) is a dominating pair of G if and only if $x \in X$ and $y \in Y$}
begin
  choose an arbitrary vertex w of G;
  LBFS(G, w);
  let x be the vertex numbered last by LBFS(G, w);
  DSETS(G, x);
  $Y = \emptyset$;
  for every $y \in V$ **do begin**
    count $\leftarrow |V| - \text{span}(y)$; {number of vertices not in $\{u | \sigma(u) \geq \text{span}(y)\}$
    for each $u \in N(y)$ **do**
      if $\sigma(u) < \text{span}(y)$ **then** *count* $\leftarrow$ *count* $- 1$;
    if *count* $= 0$ **then** $Y \leftarrow Y \cup \{y\}$
    end;
  let y be the vertex numbered last by LBFS(G, x);
  DSETS(G, y);
  $X = \emptyset$;
  for every $x \in V$ **do begin**
    count $\leftarrow |V| - \text{span}(x)$; {number of vertices not in $\{u | \sigma(u) \geq \text{span}(x)\}$
    for each $u \in N(x)$ **do**
      if $\sigma(u) < \text{span}(x)$ **then** *count* $\leftarrow$ *count* $- 1$;
    if *count* $= 0$ **then** $X \leftarrow X \cup \{x\}$
    end;
  return(X, Y)
  end; {ALL-DPs}

As an illustration, when run with the graph G of Figure 1 as input, Procedure ALL-DPs returns $X = \{a, q\}$, $Y = \{e, f, g, h, i, j, k\}$ or $X = \{e, f, g, h, i, j, k\}$, $Y = \{a, q\}$ (depending upon the initial choice of w).

THEOREM 6.2. *For a connected AT-free graph $G = (V, E)$ with diam(G) > 3, Procedure ALL-DPs computes sets X and Y such that (x, y) is a dominating pair of G if and only if $x \in X$ and $y \in Y$ in $O(|V| + |E|)$ time.*

Proof. We observe that, by Theorem 4.1 and Lemma 5.1, the vertex x in ALL-DPs is guaranteed to be a pokable dominating pair vertex. Similarly, by Theorem 4.1 and Lemma 5.1, the vertex y is a pokable dominating pair vertex. (This follows from the observation that the set of possible numberings produced by DSETS(G, x) is exactly the set of possible numberings of LBFS(G, x), since DSETS is simply LBFS with some additional computations.) Thus, the correctness of Procedure ALL-DPs follows from Theorem 5.3 and Theorem 6.1. Similarly, the complexity of ALL-DPs is the sum of the complexities of LBFS and DSETS plus an $O(|V| + |E|)$ term. □

Let $G = (V, E)$ be a connected AT-free graph with diam$(G) > 3$. Notice that, even though there may be $O(|V|^2)$ dominating pairs in G, Procedure ALL-DPs can compute and represent them in linear time, by virtue of Theorem 6.1. A similar comment applies to the sets $D(v, x)$ for all $v \in V$; even though the sum of the cardinalities of the sets may be $O(|V|^2)$, Procedure DSETS can compute an implicit representation of them in linear time.

We conclude with a corollary, which follows from the fact that some minimum cardinality connected dominating set must be a shortest path between the vertices of a dominating pair (proved in [9]). Once X and Y have been found, a minimum distance dominating pair can be found in linear time by performing a breadth-first search starting at X until a vertex of Y is encountered. In [7], we presented a linear time algorithm to compute a dominating path in an arbitrary connected AT-free graph, but that algorithm does not guarantee a minimum cardinality dominating path. The method of the present paper does guarantee a minimum cardinality dominating path for connected AT-free graphs with diameter greater than 3.

COROLLARY 6.3. *Let $G = (V, E)$ be a connected AT-free graph with diameter greater than 3. A minimum cardinality connected dominating set of G can be computed in $O(|V| + |E|)$ time.* □

7. Conclusions. We have presented a linear time algorithm, based on the well-known lexicographic breadth-first search of [16], for finding a pokable dominating pair in a connected AT-free graph, $G = (V, E)$. The algorithm provides a constructive proof of the existence of pokable dominating pairs in connected AT-free graphs (an existential proof of this fact was given in [9]). It is an improvement over the previously known $O(|V|^3)$ algorithm of [2]. In addition, we extended the dominating pair algorithm to find all dominating pairs in a connected AT-free graph, $G = (V, E)$, with diameter greater than 3. Even though there may be $O(|V|^2)$ dominating pairs, the extended algorithm can compute and implicitly represent them in $O(|V| + |E|)$ time. We remark that the simpler maximum cardinality search (MCS) of Tarjan and Yannakakis [17] cannot take the place of LBFS in these algorithms.

In [8], we presented a different linear time algorithm for finding a dominating pair in a connected AT-free graph. This other algorithm is based on a recursive use of a maximum cardinality breadth-first search. The method does not seem to allow linear time calculation of $D(v, x)$ for all vertices v where x is a pokable dominating pair vertex, or of sets X and Y when the diameter of the graph is greater than 3.

REFERENCES

[1] K. A. Baker, P. C. Fishburn, and F. S. Roberts, *Partial orders of dimension two*, Networks, 2 (1971), pp. 11–28.

[2] H. Balakrishnan, A. Rajaraman, and C. Pandu Rangan, *Connected domination and Steiner set on asteroidal triple-free graphs*, in Workshop on Algorithms and Data Structures, WADS '93, F. Dehne, J.-R. Sack, N. Santoro, and S. Whitesides, eds., Lecture Notes in Comput. Sci., 709, Springer-Verlag, New York, 1993, pp. 131–141.

[3] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North-Holland, Amsterdam, 1976.

[4] K. S. Booth and G. S. Lueker, *Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 335–379.

[5] K. S. Booth and G. S. Lueker, *A linear time algorithm for deciding interval graph isomorphism*, J. ACM, 26 (1979), pp. 183–195.

[6] D. G. Corneil and P. A. Kamula, *Extensions of permutation and interval graphs*, Congr. Numer., 58 (1987), pp. 267–276.

[7] D. G. Corneil, S. Olariu, and L. Stewart, *A linear time algorithm to compute a dominating path in an AT-free graph*, Inform. Process. Lett., 54 (1995), pp. 253–257.

[8] D. G. Corneil, S. Olariu, and L. Stewart, *Computing a dominating pair in an asteroidal triple-free graph in linear time*, in Workshop on Algorithms and Data Structures, WADS '95, S. G. Akl, F. Dehne, J.-R. Sack, and N. Santoro, eds., Lecture Notes in Comput. Sci., 955, Springer-Verlag, New York, 1995, pp. 358–368.

[9] D. G. Corneil, S. Olariu, and L. Stewart, *Asteroidal triple-free graphs*, SIAM J. Discrete Math., 10 (1997), pp. 399–430.

[10] I. Dagan, M. C. Golumbic, and R. Y. Pinter, *Trapezoid graphs and their coloring*, Discrete Appl. Math., 21 (1988), pp. 35–46.

[11] S. Even, A. Pnueli, and A. Lempel, *Permutation graphs and transitive graphs*, J. ACM, 19 (1972), pp. 400–410.

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

[13] M. C. Golumbic, C. L. Monma, and W. T. Trotter, Jr., *Tolerance graphs*, Discrete Appl. Math., 9 (1984), pp. 157–170.

[14] D. Kratsch and L. Stewart, *Domination on cocomparability graphs*, SIAM J. Discrete Math., 6 (1993), pp. 400–417.

[15] C. G. Lekkerkerker and J. C. Boland, *Representation of a finite graph by a set of intervals on the real line*, Fund. Math., 51 (1962), pp. 45–64.

[16] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.

[17] R. E. Tarjan and M. Yannakakis, *Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, SIAM J. Comput., 13 (1984), pp. 566–579.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1298–1309

GUILLOTINE SUBDIVISIONS APPROXIMATE POLYGONAL SUBDIVISIONS: A SIMPLE POLYNOMIAL-TIME APPROXIMATION SCHEME FOR GEOMETRIC TSP, *k*-MST, AND RELATED PROBLEMS *

JOSEPH S. B. MITCHELL†

Abstract. We show that any polygonal subdivision in the plane can be converted into an "m-guillotine" subdivision whose length is at most $(1 + \frac{c}{m})$ times that of the original subdivision, for a small constant c. "m-Guillotine" subdivisions have a simple recursive structure that allows one to search for the shortest of such subdivisions in polynomial time, using dynamic programming. In particular, a consequence of our main theorem is a simple polynomial-time approximation scheme for geometric instances of several network optimization problems, including the Steiner minimum spanning tree, the traveling salesperson problem (TSP), and the k-MST problem.

Key words. approximation algorithms, polynomial-time approximation scheme, traveling salesperson problem, k-MST, Steiner minimal trees, guillotine subdivisions, computational geometry

AMS subject classifications. 68Q25, 68R10, 68U05

PII. S0097539796309764

1. Introduction.

We obtain a simple polynomial-time approximation scheme for geometric instances of some network optimization problems, including the Steiner minimum spanning tree, the traveling salesperson problem (TSP), and the k-MST problem.

The method is based on the concept of an "m-guillotine subdivision," a simple extension of the recent approximation method of Mitchell [9], which considered the case $m = 1$. Roughly speaking, an "m-guillotine subdivision" is a polygonal subdivision with the property that there exists a line ("cut"), whose intersection with the subdivision edges consists of a small number ($O(m)$) of connected components, and the subdivisions on either side of the line are also m-guillotine. The upper bound on the number of connected components allows one to apply dynamic programming to optimize over m-guillotine subdivisions, as there is a succinct specification of how subproblems interact across a cut.

Key to our method is a theorem showing that any polygonal subdivision can be converted into an m-guillotine subdivision by adding a set of edges whose total length is small: at most $\frac{c}{m}$ times that of the original subdivision (where $c = 1, \sqrt{2}$, depending on the metric). Then, using dynamic programming to optimize over an appropriate class of m-guillotine subdivisions, we obtain, for any fixed m, $(1 + \frac{c}{m})$-approximation algorithms that run in polynomial-time ($n^{O(m)}$) for various network optimization problems.

Related work. Over the last few decades, there has been a wealth of research on the problems studied here, both in the graph versions of the problems and in the geometric versions. Almost any standard textbook on algorithms and networks discusses them; e.g., see [4, 6, 12]. For a survey of work on the TSP, refer to the

*Received by the editors September 25, 1996; accepted for publication (in revised form) July 18, 1997; published electronically March 22, 1999. This research was supported in part by Hughes Research Laboratories and NSF grants CCR-9204585 and CCR-9504192.

http://www.siam.org/journals/sicomp/28-4/30976.html

†Department of Applied Mathematics and Statistics, State University of New York, Stony Brook, NY 11794-3600 (jsbm@ams.sunysb.edu).

book [7] edited by Lawler et al. For a survey on approximation algorithms, refer to the recent book [5] edited by Hochbaum.

All of the geometric optimization problems considered here are known to be NP-hard. Polynomial-time approximation algorithms were known, allowing one to get within a constant factor of optimal. However, it has been open as to whether or not one can, in polynomial time, achieve an approximation factor of $(1+\epsilon)$ for any fixed $\epsilon > 0$; i.e., no polynomial-time approximation scheme (PTAS) was known. In particular, no factor better than the Christofides bound of 1.5 was known for the Euclidean TSP.

In this paper, we point out how a minor modification to a previous result [9, 11] (see also [3]) leads to a PTAS for various geometric optimization problems, including the TSP, Steiner tree, and k-MST.

In an exciting recent development, Arora [1] announced that he had found a PTAS for the Euclidean TSP, as well as the other problems considered in this paper, thereby achieving essentially the same results that we report here, using decomposition schemes that are somewhat similar to our own. Arora's remarkable results predate this paper by several weeks. Arora has generalized his method also to higher dimensions, obtaining a running time of $n^{O(\log^{d-2} n)/\epsilon^{d-1}}$ in d dimensions. For the two-dimensional TSP, Arora estimates that his analysis yields a time bound of roughly $n^{100/\epsilon}$; he adds that a more careful analysis should yield roughly $n^{30/\epsilon}$. In this paper, we give analysis giving an explicit exponent: time $O(n^{20m+5})$ to get within factor $(1+\frac{2\sqrt{2}}{m})$ of optimal in the Euclidean planar TSP; in terms of ϵ, the time bound is $O(n^{\frac{40\sqrt{2}}{\epsilon}+5})$.

2. m-Guillotine subdivisions. We follow most of the notation of [9], only somewhat generalized. We consider a polygonal subdivision ("planar straight-line graph") S that has n edges (and hence $O(n)$ vertices and facets). Let E denote the union of the edge segments of S, and let V denote the vertices of S. We can assume (without loss of generality) that S is restricted to the unit square, B (i.e., $E \subset \text{int}(B)$). Then, each facet (2-face) is a bounded polygon, possibly with holes. The *length* of S is the sum of the lengths of the edges of S. If all edges E are horizontal or vertical, then we say that S is *rectilinear*.

A closed, axis-aligned rectangle W is a *window* if $W \subseteq B$. In the following definitions, we fix attention on a given window, W.

A line ℓ is a *cut* for E (with respect to W) if ℓ is horizontal or vertical and $\ell \cap \text{int}(W) \neq \emptyset$. The intersection, $\ell \cap (E \cap \text{int}(W))$, of a cut ℓ with $E \cap \text{int}(W)$ (the restriction of E to the window W) consists of a discrete (possibly empty) set of subsegments of ℓ. (Some of these "segments" may be single points, where ℓ crosses an edge.) The endpoints of these subsegments are called the *endpoints along* ℓ (with respect to W). (The two points where ℓ crosses the boundary of W are not considered to be endpoints along ℓ.) Let ξ be the number of endpoints along ℓ, and let the points be denoted by $p_1, \ldots, p_\xi$, in order along ℓ.

For a positive integer m, we define the *m-span*, $\sigma_m(\ell)$, of ℓ (with respect to W) as follows. If $\xi \leq 2(m-1)$, then $\sigma_m(\ell) = \emptyset$; otherwise, $\sigma_m(\ell)$ is defined to be the (possibly zero-length) line segment, $p_m p_{\xi-m+1}$, joining the mth endpoint, p_m, with the mth-from-the-last endpoints, $p_{\xi-m+1}$. (See Figure 2.1.)

A (horizontal or vertical) cut ℓ is an *m-perfect cut with respect to* W if $\sigma_m(\ell) \subseteq E$. In particular, if $\xi \leq 2(m-1)$, then ℓ is trivially an m-perfect cut (since $\sigma_m(\ell) = \emptyset$). Similarly, if $\xi = 2m-1$, then ℓ is m-perfect (since $\sigma_m(\ell)$ is a single point). Otherwise, if ℓ is m-perfect, and $\xi \geq 2m$, then $\xi = 2m$. See Figure 2.2 for an example.

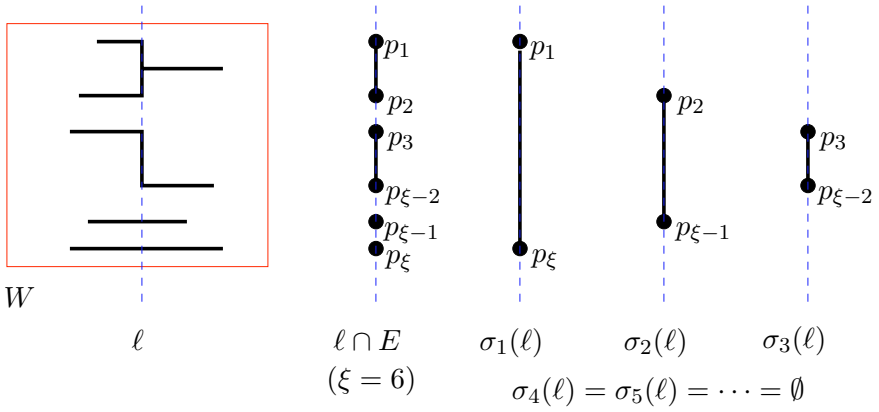


FIG. 2.1. *Definition of m-span.*

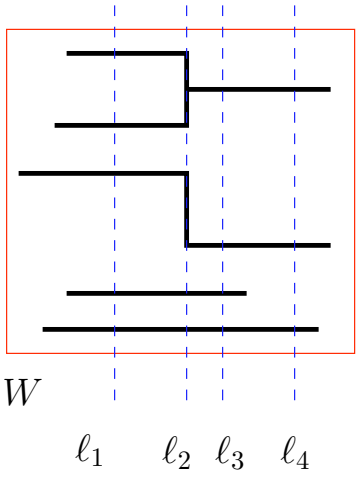


FIG. 2.2. *The vertical cuts $\ell_1, \ell_2, \ell_3, \ell_4$ are 3-perfect (also m-perfect, for $m \geq 4$). The cut ℓ_4 is also 2-perfect (but not 1-perfect).*

Finally, we say that S is an *m-guillotine subdivision with respect to window* W if either (1) $E \cap \text{int}(W) = \emptyset$; or (2) there exists an m-perfect cut, ℓ, with respect to W, such that S is m-guillotine with respect to windows $W \cap H^+$ and $W \cap H^-$, where H^+, H^- are the closed half-planes induced by ℓ. We say that S is an *m-guillotine subdivision* if S is m-guillotine with respect to the unit square, B. A 1-guillotine subdivision is illustrated in Figure 2.3, where "cut" is used to indicate where a 1-perfect cut can be made.

3. The main theorem. For rectilinear subdivisions, the proof of our main theorem directly follows that of [9] with only a very minor change: we use the concept of "m-darkness," in which we require m walls to block light from the boundary. The proof in [9] used $m = 1$; so we copy below the proof from [9] with "m" replacing "1".

THEOREM 3.1. *Let S be a rectilinear subdivision, with edge set E, of length L. Then, for any positive integer m, there exists an m-guillotine rectilinear subdivision, S_G, of length at most $(1 + \frac{1}{m})L$ whose edge set, E_G, contains E.*

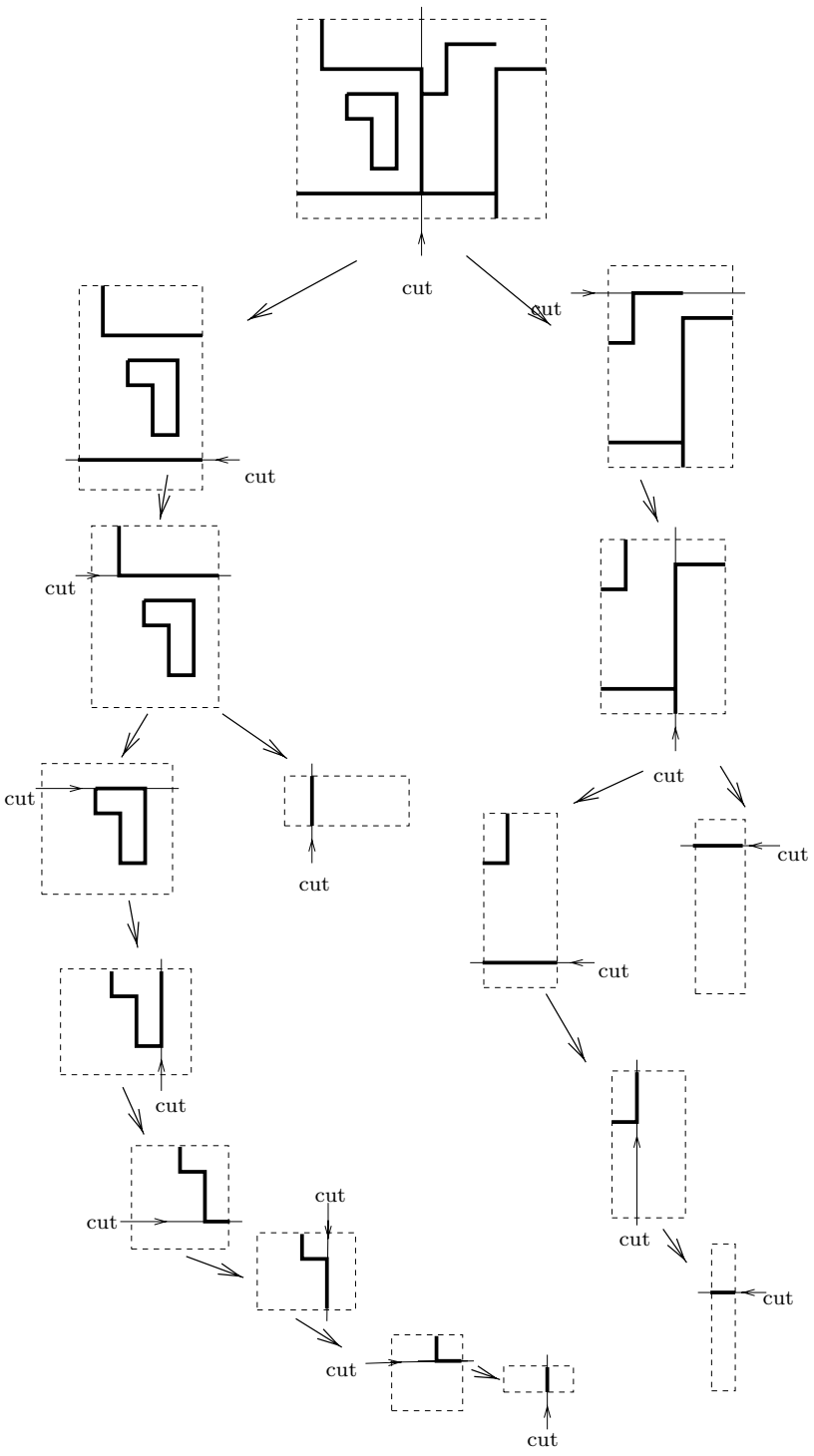


FIG. 2.3. *An example of a* 1*-guillotine rectilinear subdivision.*

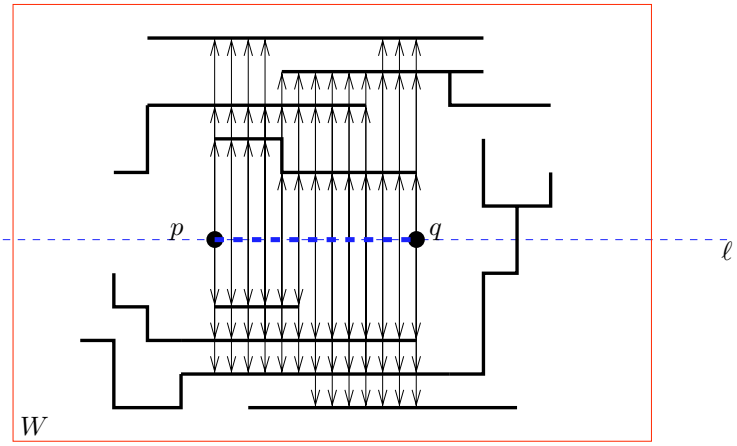


FIG. 3.1. *Subsegment $pq \subset \ell$ is 3-dark; its length is charged to the 3 levels of E that lie above it and the 3 levels of E that lie below it.*

Proof. We will convert S into an m-guillotine subdivision S_G by adding to E a new set of horizontal/vertical edges whose total length is at most $\frac{1}{m}L$. The construction is recursive; at each stage, we show that there exists a cut, ℓ, with respect to the current window W (which initially is the box B), such that we can afford to add the m-span $\sigma_m(\ell)$ to E, while appropriately charging off the length of $\sigma_m(\ell)$. (Once we add $\sigma_m(\ell)$ to E, ℓ becomes an m-perfect cut with respect to W.)

In fact, we will restrict ourselves to a special discrete set, $\mathcal{L}$, of horizontal/vertical cuts, namely those determined by the x- or y-coordinates of original vertices V of the subdivision, or by the midpoints between consecutive x- or y-coordinates of V.

First, note that if an m-perfect cut (with respect to W) exists, then we can simply use it and proceed, recursively, on each side of the cut.

We say that a point p on a cut ℓ is *m-dark with respect to ℓ and W* if, along $\ell^\perp \cap \text{int}(W)$, there are at least m endpoints (strictly) on each side of p, where $\ell^\perp$ is the line through p and perpendicular to ℓ.[1] We say that a subsegment of ℓ is *m-dark* (with respect to W) if all points of the segment are m-dark with respect to ℓ and W.

The important property of m-dark points along ℓ is the following. Assume, without loss of generality, that ℓ is horizontal. The m-dark portion of ℓ is, in general, a union of subsegments of $\ell \cap W$, some endpoints of which may not be themselves m-dark points. (In terms of the *length* (measure) of the m-dark portion, though, the discrete set of endpoints is not relevant.) Let pq be an open subsegment of the m-dark portion of ℓ. Then we can charge the length of pq off to the bottoms of the first m subsegments, $E^+ \subseteq E$, of edges that lie above pq, and the tops of the first m subsegments, $E^- \subseteq E$, of edges that lie below pq (since we know that there are at least m edges "blocking" pq from the top/bottom of W). We charge pq's length half to E^+ (charging each of the m levels of E^+ *from below*, with $\frac{1}{2m}$ units of charge) and half to E^- (charging each of the m levels of E^- *from above*, with $\frac{1}{2m}$ units of charge). In Figure 3.1 we illustrate how a 3-dark subsegment, pq, has its length charged off to the 3 levels of E that are above/below it.

[1] We can think of the edges E as being "walls" that are not very effective at blocking light—light can go through $m-1$ walls but is stopped when it hits the mth wall; then, p on a line ℓ is m-dark if p is not illuminated when light is shone in from the boundary of W, along the direction of $\ell^\perp$.

We call a cut ℓ *favorable* if the total length of the m-dark portion of ℓ is at least as great as the length of the m-span, $\sigma_m(\ell)$. The lemma below shows that a favorable cut always exists (even one in the special discrete set, $\mathcal{L}$). For a favorable cut ℓ, we add its m-span to the edge set (charging off its length, as above), and recurse on each side of the cut, in the two new windows. After a portion of E has been charged on one side, due to a cut ℓ, it will be within m levels of the boundary of the windows on either side of ℓ, and, hence, within m levels of the boundary of any future windows, found deeper in the recursion that contain the portion. Thus, *no portion of E will ever be charged more than once from each side* (top and bottom), so no portion of E will ever pay more than its total length, times $\frac{1}{m}$, in charge ($\frac{1}{2m}$ from each side). Also, the new edges added (the spans $\sigma_m(\ell)$) are never themselves charged, since they lie on window boundaries and cannot therefore serve to make a portion of some future cut m-dark.

(Note too that, in order for a cut ℓ to be favorable but not m-perfect, there must be at least one segment (in fact, at least m segments) of E parallel to ℓ in each of the two open half-planes induced by ℓ; thus, the recursion must terminate in a finite number of steps.)

Since the total length of all m-spans for all favorable cuts is at most $\frac{1}{m}L$, and the total length of all m-spans for all m-perfect cuts is at most L, we are done. □

We now prove our key lemma, which guarantees the existence of a favorable cut. The proof of the lemma uses a particularly simple argument, based on elementary calculus (reversing the order of integration). It appears already in [9], but we repeat it here for completeness.

LEMMA 3.2. *For any subdivision S and any window W, there is a favorable cut.*

Proof. We show that there must be a favorable cut that is either horizontal or vertical.

Let $f(x)$ denote the length of the m-span (with respect to W) of the vertical line through x. (We define $f(x) = 0$, if $x \in [0,1]$ lies outside of the projection of $W \subseteq B$ onto the x-axis.) Then, $\int_0^1 f(x)dx$ is simply the area, A_x, of the set R_x of points of W that are m-dark with respect to horizontal cuts. Similarly, define $g(y)$ to be the length of the m-span of the horizontal line through y, and let $A_y = \int_0^1 g(y)dy$.

Assume, without loss of generality, that $A_x \geq A_y$. We claim that there exists a horizontal favorable cut; i.e., we claim that there exists a horizontal cut, ℓ, such that the length of its m-dark portion is at least as large as the length of its m-span, $\sigma_m(\ell)$. To see this, note that A_x can be computed by switching the order of integration, "slicing" the region R_x horizontally, rather than vertically; i.e., $A_x = \int_0^1 h(y)dy$, where $h(y)$ is the length of the intersection of R_x with a horizontal line through y (i.e., $h(y)$ is the length of the m-dark portion of the horizontal line through y). Thus, since $A_x \geq A_y$, we get that $\int_0^1 h(y)dy \geq \int_0^1 g(y)dy \geq 0$. Thus, it cannot be that for all values of $y \in [0,1]$, $h(y) < g(y)$, so there exists a $y = y^*$ for which $h(y^*) \geq g(y^*)$. The horizontal line through this y^* is a cut satisfying the claim of the lemma. (If, instead, we had $A_x \leq A_y$, then we would get a *vertical* cut satisfying the claim.)

Finally, we note that, in the rectilinear case, f, g, and h are piecewise-constant, with discontinuities corresponding to vertices V of S. Then, we can always select y^* to be at a discontinuity or at a midpoint between two discontinuities. □

General polygonal subdivisions. Consider now a subdivision S whose edges E are *not* rectilinear. A moment's reflection reveals that our charging scheme and the key lemma are quite general and do not really use the rectilinearity of S (or even

the straightness of edges in E). In fact, the proof of the main theorem goes through, almost exactly as before, adding "favorable" cuts that are horizontal or vertical, and charging the added length off to the original length of the subdivision. However, we must address the issue of the discretization of cuts (e.g., in order to specify a dynamic program) and, thereby, the termination of the recursion that converts an arbitrary subdivision to an m-guillotine subdivision.

One approach (as in earlier drafts of this paper) is to use discretization of orientations, and/or discretization onto a sufficiently fine grid. Here, we opt instead to modify slightly our previous definition of m-guillotine subdivision as follows.

Assume, without loss of generality, that no two vertices have a common x- or y-coordinate. We let E_W denote the subset of E consisting of the union of segments of E having at least one endpoint inside (or on the boundary of) W. The *combinatorial type* (*with respect to* E) of a window W is an ordered listing, for each of the four sides of W, of the identities of the (closed) line segments of E_W that intersect it (the side of W, each considered to be an open line segment), as well as the identities of the line segments of E_W that intersect each of the four corners of W. We say that W is *minimal* (with respect to E) if there does not exist a $W' \subset W$, strictly contained in W, having the same combinatorial type as W. By standard critical placement arguments,[2] we see that, since it has four degrees of freedom, a minimal window is determined by four *contact pairs*, defined by a vertex $v \in V$ in contact with a side of W or by a corner of W in contact with a segment of E_W. (Thus, there are at most $O(n^4)$ minimal windows.) For any window W containing at least one vertex of E, we let $\overline{W}$ denote a minimal window, contained within W, having the same combinatorial type as W. (At least one such $\overline{W}$ must exist.)

Now, we say that S is an *m-guillotine subdivision with respect to window W* if either (1) $V \cap \mathrm{int}(W) = \emptyset$; or (2) there exists an m-perfect cut, ℓ, with respect to a minimal window, $\overline{W} \subseteq W$, such that S is m-guillotine with respect to windows $W \cap H^+$ and $W \cap H^-$, where H^+, H^- are the closed half-planes induced by ℓ. (Note that, since $\overline{W}$ is minimal, necessarily windows $W \cap H^+$ and $W \cap H^-$ will each have a combinatorial type different from that of W.)

THEOREM 3.3. *Let S be a polygonal subdivision, with edge set E, of length L. Then, for any positive integer m, there exists an m-guillotine polygonal subdivision, S_G, of length at most $(1 + \frac{\sqrt{2}}{m})L$ whose edge set, E_G, contains E.*

Proof. The proof exactly follows that of the rectilinear case (Theorem 3.1): We use a recursive construction, together with our charging scheme, and Lemma 3.2 applied

[2] In particular, an arbitrary window, W, can be made minimal by the following "shrinking" process: Slide the right side of W inward until the first *event*, when (a) the side hits a vertex, or (b) an endpoint of the side strikes a segment of E_W. In case (a), we have "pinned" the right side of W, and we move next to the top side (then the left side, and the bottom side). In case (b), we next move inwards *both* of the two sides incident on the corner (endpoint) that struck a segment, $s \subset E_W$, while keeping that corner in contact with segment s, until another event (of type (a) or (b)) occurs. (Since we know that $s \in E_W$ has one of its endpoints inside W, we know that as we slide the corner along s, the two sides either both move inward or both move outward.) Again, in case of an event of type (a), we are done moving those two sides (they are both pinned), and we proceed to move inwards other sides (processing unpinned sides in the order right, top, left, bottom). In case of an event of type (b), a second corner of the window has struck a segment, $s' \subset E_W$, and we then proceed to slide *three* sides of W inward simultaneously, while keeping both of the pinned corners sliding on their respective segments of E_W. We may end up having all four sides of W moving inwards at once, with three, or possibly even four (in a degenerate case), corners of W restricted to slide on specific segments of E_W. Since each of these segments of E_W must have one endpoint within W, we know that we will eventually have an event of type (a)—a collision with a vertex. When this happens, all four sides of W have been pinned.

to a minimal window $\overline{W} \subseteq W$, to show that we can convert S into an m-guillotine subdivision S_G by adding to E a new set of horizontal/vertical bridge segments whose total length is at most $\frac{\sqrt{2}}{m}L$. (Here, we do not restrict attention to any special subset of horizontal and vertical cuts (such as $\mathcal{L}$); cuts can be classified according to the combinatorial types of the new windows they create.)

The only difference that we should mention is the origin of the "$\sqrt{2}$" term in the bound. This comes from the fact that each side of an inclined segment of E may be charged *twice*, once vertically and once horizontally. Specifically, the charge assigned to a segment is at most $\frac{1}{m}$ times the sum of the lengths of its x- and y-projections, i.e., at most $\frac{\sqrt{2}}{m}$ times its length. □

4. Some applications. We now discuss how our main theorem can be applied, leading to polynomial-time approximation schemes for some NP-hard optimization problems on a set P of n sites in the plane.

Steiner tree problem: Determine a tree of minimum total length that spans (visits) the set of points P. In this problem, as opposed to the *minimum spanning tree* (MST) problem (solvable exactly in $O(n \log n)$ time for points in the Euclidean plane [13]), the tree is allowed to have vertices ("Steiner points") that are *not* among the points of P.

Steiner k-MST problem: For a given integer k ($k \leq n$), determine a tree, possibly with Steiner points, of minimum total length that spans at least k of the n points in P.

TSP: Determine a *tour* (cycle) of minimum length that visits each point of the set, P, of sites. Because it is of minimum length, a Euclidean TSP tour will necessarily visit each site exactly once, and will be a *simple* polygon (a closed polygonal walk that does not self-intersect). The k-TSP is to determine a minimum-length tour on a subset of at least k of the n points.

k-MST problem: For a given integer k ($k \leq n$), determine a tree, with vertices in the set P, of minimum total length that spans at least k of the n points in P.

4.1. Steiner tree. The dynamic programming algorithm of [9, 11], which computes a 2-approximation to the Steiner k-MST (and hence to Steiner tree, for $k = n$), in the L_1 metric, generalizes immediately to the case of m-guillotine subdivisions. Now, instead of a factor of $1 + \frac{1}{1} = 2$, as in [9] (for the case $m = 1$), we get a factor $1 + \frac{1}{m}$. Exactly the same algorithm works, only now there are up to $2m$ endpoints per side of the rectangular subproblem (instead of 2 per side, as in [9]). Thus, there are $O(k)$ choices of the integer k' that specifies the number of sites that must be visited within a subproblem, $O(n^4)$ choices of rectangle B bounding a subproblem, $O(n^{4 \cdot 2m})$ choices for the endpoints on each side of the rectangle, $O(n^{2m+1})$ choices of cut (and endpoints along the cut), and $O(k)$ choices for how to partition k'. Overall, we get time $O(k^2 n^{10m+5})$. In the case that $k = n$, we do not need to choose the value of k', or how to partition it when we select a cut, since we are forced to visit all sites within a subproblem; thus, the factor of k^2 drops, leaving us with a time bound of $O(n^{10m+5})$.

For the Euclidean metric, essentially the same algorithms work, but we first augment P with a set, G, of candidate (approximate) Steiner points, such that each Steiner point in an optimal tree can be rounded to one of the candidate points, without increasing the size of the tree by much. (In contrast, in the rectilinear case, we can assume that all Steiner points lie (exactly) at vertices of the grid induced by horizontal/vertical lines through points of P; thus, no new candidates need to be added,

since these grid points are potential subdivision vertices in the dynamic programming recursion that searches for a shortest possible m-guillotine subdivision.) One choice of set G that works is to let G be a regular (square) grid of $O(n^2M^2)$ points, with spacing $\frac{diam(P)}{nM}$, for some $M \geq m$. Then, each Steiner point in an optimal tree can be rounded to such a grid point by adding a segment of length less than $\frac{diam(P)}{nM}$; this adds an overall length of only $\frac{diam(P)}{M} \leq \frac{diam(P)}{m} \leq (\frac{1}{m})L^*$ to the optimal tree, where L^* is the length of the optimal tree. Our dynamic programming algorithm, then, searches for a minimum-length m-guillotine subdivision that spans all (or at least k) of the points of P, while allowing vertices of the subdivision to be placed at any of the points $P \cup G$.

COROLLARY 4.1. *Given any fixed positive integer m, and any set of n points in the plane, there is an $n^{O(m)}$-time algorithm to compute a Steiner spanning tree (or Steiner k-MST) whose length is within a factor $(1 + \frac{1}{m})$ of the minimum.*

4.2. TSP. For the TSP, we again apply dynamic programming, since the main theorem gives us an easy way to decompose the problem recursively.

COROLLARY 4.2. *For any fixed positive integer m, there is an $O(n^{20m+5})$ algorithm to compute an approximate TSP whose Euclidean length is within a factor $(1 + \frac{2\sqrt{2}}{m})$ of optimal. (For the L_1 metric, the factor is $(1 + \frac{2}{m})$, and the time bound $O(n^{10m+5})$.)*

Proof. (We present the details for the Euclidean metric; details for the L_1 metric are similar.) Let T^* be a minimum-length TSP tour for P, and let L^* denote its length.

The structure of the proof is typical of many approximation results: (a) We show that T^* can be transformed into a special type of subdivision of length at most $(1 + \frac{2\sqrt{2}}{m})L^*$; (b) we give a dynamic programming algorithm to compute (exactly) a shortest subdivision, having edge set E^*_G, of this special type; and (c) we show how to obtain a tour of P from the resulting subdivision, with the tour having length at most that of the subdivision.

(a) *Transforming T^*.* We know that T^* is a simple polygon with vertices P. Following the proof of Theorem 3.1, we add segments (m-spans of favorable cuts), called "bridges," to the subdivision T^*, in order to make it m-guillotine; *however*, when we add an m-span segment (corresponding to cut ℓ), we *double* it, creating a second copy that lies on top of the first copy. (The reason for the doubling will be made apparent below, in step (c).) Furthermore, we "slide" the doubled bridge, parallel to itself and in the direction that decreases (or does not increase) the length of the bridge, until it lies on a vertical/horizontal line through some point of P. The result is a *pinned double bridge*, "pinned" by this line through a point in P. We let S denote the resulting subdivision, and let E denote its edge set (including the doubled segments). The length of E is at most $(1+\frac{2\sqrt{2}}{m})L^*$, since, by Theorem 3.1, the bridges that we add have total length, prior to doubling, of at most $\frac{\sqrt{2}}{m}L^*$.

(b) *Dynamic programming algorithm.*

Input to Subproblem (see example in Figure 4.1):

1. A rectangle R, corresponding to a minimal window, determined by (up to) four points of P, together with some subset of the $O(m)$ edges defining the boundary information (below).
2. Boundary information specifying $\leq 2m$ crossing segments (each determined by a pair of points in P) that cross the boundary of R, and at most one

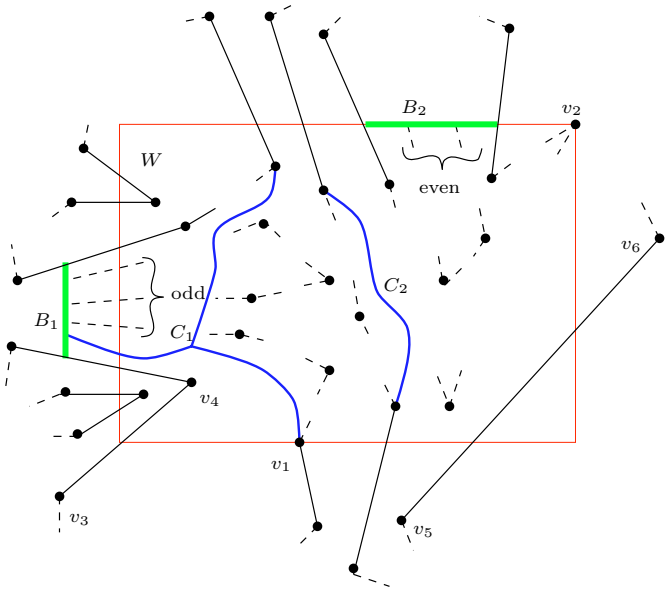


FIG. 4.1. *Example subproblem for TSP dynamic program: The window W is determined by the vertices v_1, v_2, and the segment v_3v_4 (on which its lower left corner is in contact). There are two bridges, B_1 (which is pinned at some coordinate left of the left boundary of W) and B_2 (pinned at the y-coordinate of v_2). Bridge B_1 is required to have an odd number of segments incident on it, while bridge B_2 is required to have an even number. The partition $\mathcal{P}$ specifies the interconnections indicated by trees C_1 and C_2. For vertex v_2, both incident edges are required to be within W; for vertex v_1, only one incident edge must be within W. All points within W, not yet incident on some segment crossing the boundary, are required to have degree two. Short dashed segments indicate a feasible possible set of interconnections.*

(pinned) bridge, per side of R, together with the parity (odd vs. even) of the number of edges of $E^*_G \cap R$ that must be incident on it.

3. Connectivity constraints, given in the form of a partition, $\mathcal{P}$, of the set of crossing segments and bridges, indicating which subsets are required to be connected within subproblem R.

Objective:

Compute a minimum-length m-guillotine subdivision, with edge set E^*_G, such that (1) the subdivision uses only segments joining points of P or *doubled* vertical/horizontal bridge segments that are pinned, (2) all boundary information is satisfied, (3) every point of P within R has degree two, and (4) all connectivity constraints within R are satisfied.

Note that there are $O(n^4 \cdot (n^{4m})^4) = O(n^{16m+4})$ possible inputs (subproblems), since there are $O(n^4)$ choices of R, and $O(n^{4m})$ choices of crossing segments on each of the four sides of R. (The number of possible connectivity constraints and boundary information is constant for fixed m.)

The initial call to the recursion will ask for a solution for the case that R is the bounding box of P, with empty boundary information, and connectivity constraints that simply say that all points of P inside R must be connected (with each point having degree 2).

In the base case, if R has no points of P in its interior, then the subproblem is solved by brute force, since it has only constant size (for fixed m). Otherwise, we can

solve the subproblem recursively, optimizing over all choices associated with a cut:

1. $O(n)$ choices of a horizontal/vertical cut.
2. $O(n^{4m})$ choices of new boundary information on the cut. In particular, we select $\leq 2m$ segments (each determined by a pair of points) that cross the cut, together with the information specifying a possible bridge segment. We require that boundary information of the new subproblems be consistent with boundary information of the given problem.
3. $O(1)$ (for fixed m) choices of connectivity constraints for the two new subproblems determined by the cut, subject to the requirement that these constraints be consistent with the constraints $\mathcal{P}$.

Since there are $O(n \cdot n^{4m})$ choices to make in partitioning a subproblem and there are overall $O(n^{16m+4})$ subproblems, we obtain an overall time complexity of $O(n^{20m+5})$.

(c) *Obtaining a tour.* Given the solution, S_G^*, to the dynamic program, we claim that there is an Eulerian subgraph of E_G^* (the edge set of S_G^*, *including* doubled bridges) that covers P, so that any Eulerian cycle defines a tour of P, and the length of this tour is at most the length of E_G^*.

Now, by the constraints imposed in the subproblems of the dynamic programming algorithm, all vertices of S_G^* have even degree, *except* possibly those that lie along a bridge segment (that is an m-span of some perfect cut). But the fact that bridge segments have their lengths *doubled* allows us to make all vertices along bridges have even degree as well, by keeping an appropriate subset (selection of subsegments) of the second copy of a bridge: Consider the vertices along a single copy of the bridge, and use subsegments from the second copy of the bridge to link, consecutively, those vertices along the bridge that have odd degree. There will necessarily be an even number of odd-degree vertices along a bridge, since we require that a bridge segment be incident to an even number of other segments.

Thus, by deleting some portions of the second copy of bridge segments, we obtain a subdivision covering P all of whose vertices have even degree. Then, an Eulerian cycle on the edges of this subdivision is a tour of P, whose length is at most $(1 + \frac{2\sqrt{2}}{m})L^*$. □

4.3. *k*-MST. Suppose now that we are given an integer $k \leq n$ and asked to find a minimum spanning tree on some subset of k of the n points P, without using Steiner points. Since we are not allowed to introduce Steiner points, e.g., where segments are incident on bridges, we use the same bridge-doubling trick as we did in writing the dynamic program for the TSP. This ensures that the only odd-degree vertices that we end up with in our optimized subdivision are those at original points of P, thereby allowing us to replace Eulerian paths, linking original sites of P, with shortcut paths that bend only at the sites P. (Naturally, we drop the requirement in the dynamic program that the points in P have degree two, and we add the parameter k to the specification of a subproblem.)

5. Conclusion. We suspect that there are many other potential applications and improvements of the results reported here. We do not yet know a characterization of the general class of problems for which our method leads to a PTAS.

While we have concentrated on L_1 and Euclidean metrics, our results generalize to other metrics on points in the plane. We are currently examining possible extensions to higher dimensions.

Finally, we mention some recent results that have been discovered since the time that this paper was originally submitted. Arora [2] has recently obtained a randomized

algorithm with expected running time that is nearly linear in n: $O(n \log^{O(1/\epsilon)} n)$. His new results also allow the d-dimensional problem to be solved in expected time $O(n(\log n)^{(O(\frac{d}{\epsilon}))^{d-1}})$.

Further, in a follow-up to this paper, Mitchell [10] has shown how a variant of m-guillotine subdivisions (termed "grid-rounded m-guillotine subdivisions") yields a (deterministic) PTAS with worst-case time bound $n^{O(1)}$, for any fixed $\epsilon > 0$, for the same set of (two-dimensional) problems we have studied in this paper.

Acknowledgments. I thank Esther Arkin, Rafi Hassin, Samir Khuller, Günter Rote, and the referees for several useful comments and suggestions that improved the presentation of the paper.

REFERENCES

[1] S. Arora, *Polynomial time approximation schemes for Euclidean TSP and other geometric problems*, in Proc. 37th Ann. IEEE Sympos. Found. Comput. Sci., Burlington, VT, IEEE Comput. Soc. Press, Los Alamitos, CA, 1996, pp. 2–12.

[2] S. Arora, *Nearly linear time approximation schemes for Euclidean TSP and other geometric problems*, in Proc. 38th Ann. IEEE Sympos. Found. Comput. Sci., Miami Beach, FL, IEEE Comput. Soc. Press, Los Alamitos, CA, 1997, pp. 554–563.

[3] A. Blum, P. Chalasani, and S. Vempala, *A constant-factor approximation for the k-MST problem in the plane*, in Proc. 27th Ann. ACM Sympos. Theory Comput., Las Vegas, NV, ACM Press, New York, 1995, pp. 294–302.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[5] D. Hochbaum, editor, *Approximation Problems for NP-Complete Problems*, PWS Publications, Boston, MA, 1997.

[6] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New York, 1976.

[7] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds., *The Traveling Salesman Problem*, Wiley, New York, 1985.

[8] C. Mata and J. S. B. Mitchell, *Approximation algorithms for geometric tour and network design problems*, in Proc. 11th Ann. ACM Sympos. Comput. Geom., Vancouver, Canada, ACM Press, New York, 1995, pp. 360–369.

[9] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-MST problem*, in Proc. 7th ACM-SIAM Sympos. Discrete Algorithms, SIAM, Philadelphia, 1996, pp. 402–408.

[10] J. S. B. Mitchell, *Approximation algorithms for geometric optimization problems*, in Proc. 9th Canadian Conference Comput. Geom., Kingston, Canada, 1997, pp. 229–232.

[11] J. S. B. Mitchell, A. Blum, P. Chalasani, and S. Vempala, *A constant-factor approximation for the geometric k-MST problem in the plane*, SIAM J. Comput., 28 (1999), pp. 771–781.

[12] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ, 1982.

[13] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1310–1325

FINE SEPARATION OF AVERAGE-TIME COMPLEXITY CLASSES*

JIN-YI CAI† AND ALAN L. SELMAN†

Abstract. We extend Levin's definition of average polynomial time to arbitrary time-bounds in accordance with the following general principles: (1) It essentially agrees with Levin's notion when applied to polynomial time-bounds. (2) If a language L belongs to DTIME($T(n)$) for some time-bound $T(n)$, then every distributional problem (L, μ) is T on the μ-average. (3) If L does not belong to DTIME($T(n)$) *almost everywhere*, then no distributional problem (L, μ) is T on the μ-average.

We present hierarchy theorems for average-case complexity, for arbitrary time-bounds, that are as tight as the well-known Hartmanis–Stearns hierarchy theorem for deterministic complexity. As a consequence, for every time-bound $T(n)$, there are distributional problems (L, μ) that can be solved using only a slight increase in time but that cannot be solved on the μ-average in time $T(n)$.

Key words. computational complexity, average-time complexity classes, hierarchy, Average-P, logarithmico-exponential functions

AMS subject classifications. 68Q10, 68Q15

PII. S0097539796311715

1. Introduction.

One of the central issues for any complexity-theoretic measure is the question of fine hierarchies. Here we consider this issue for average-case complexity. The average complexity of a problem is, in many cases, a more significant measure than its worst-case complexity. This has motivated a rich area in algorithm research, but Levin [14] was the first to advocate the general study of average-case complexity. An average-case complexity class consists of pairs called distributional problems. Each pair consists of a decision problem and a probability distribution on problem instances. Most papers to date have focused their attention on polynomial time and on the concept of average polynomial time. The primary motivation has concerned the question of whether DistNP $\subseteq$ Average-P, where DistNP and Average-P are the distributional analogues of NP and P, respectively. Many beautiful results have been obtained. Levin, for example, has proved the existence of complete problems in DistNP.

Ben-David et al. [4] were the first to suggest a general formulation of average-case complexity for time-bounds other than polynomials. We will prove a fine hierarchy theorem using their definition for time-bounds that are bounded above by some polynomial. (Our proof uses properties of a class of functions defined by Hardy, called the logarithmico-exponential functions. We present this result in section 3.) However, we will observe that the definition of Ben-David et al. cannot distinguish time-bounds of the form 2^{cn} for different values of c. Thus, there fails to be a fine hierarchy for exponential or more general time-bounds.

Then, we will use the definition of Ben-David et al. as the point of departure from which we develop a new formulation of average-case complexity. A complexity class AVTIME($T(n)$) is to consist of all distributional problems (L, μ) such that L is solvable in "time $T(n)$ on the μ-average." This is the notion that we must make precise. We approach our formulation with the following intuitions in mind. If a language L

*Received by the editors November 4, 1996; accepted for publication (in revised form) October 17, 1997; published electronically March 22, 1999.
http://www.siam.org/journals/sicomp/28-4/31171.html

†Department of Computer Science, University at Buffalo, State University of New York, Buffalo, NY 14260 (cai@cse.buffalo.edu, selman@cse.buffalo.edu). The work of the second author was supported in part by NSF grant CCR-9400229.

belongs to DTIME($T(n)$) for some time-bound $T(n)$, then the distributional problem (L, μ) should belong to the class AVTIME($T(n)$). Furthermore, if L is outside of DTIME($T(n)$) *almost everywhere* (i.e., every Turing machine that accepts L requires more than $T(|x|)$ steps for all but a finite number of input words x), then it should follow that (L, μ) does not belong to AVTIME($T(n)$).

Our definition will satisfy these conditions and in addition will agree with the definitions of Levin [14] and Ben-David et al. [4] when we apply it to polynomial time-bounds and reasonable distributions. Readers who are familiar with Levin's theory of average polynomial time will recall that a naive, intuitive formulation suffers from serious problems. This issue is discussed in detail by previous authors including, notably, Gurevich [8] and Ben-David et al. [4]. Similarly, the path to a correct formulation of average-case complexity for arbitrary time-bounds is intricate. We will develop our new definition in section 4.

We will present a hierarchy theorem for average-case complexity, for arbitrary time-bounds, that is as tight as the well-known Hartmanis–Stearns [11] hierarchy theorem for deterministic complexity. As a consequence, for every time-bound $T(n)$, there are distributional problems (L, μ) that can be solved using only a slight increase in time but that cannot be solved on the μ-average in time $T(n)$.

2. Preliminaries. We assume that all languages are subsets of $\Sigma^* = \{0,1\}^*$, and we assume that Σ^* is ordered by standard lexicographic ordering denoted $\leq$. (We will ignore the empty string ϵ and start with 0. The predecessor of x in this order is denoted by $x-1$.) We use $\subset$ to denote proper inclusion.

2.1. Turing machine running times. Although Turing machine running times are frequently given as functions on the natural numbers, $T : \mathrm{N} \to \mathrm{N}$, we will often need the more accurate view that a Turing machine running time is a function $S : \Sigma^* \to \mathrm{N}$. In this case, the relation between the two interpretations is clear. Namely, $T(n) = \max\{S(x) \mid |x| = n\}$. For two functions T and T', where $T, T' : \mathrm{N} \to \mathrm{N}$, recall that $T'(n) = o(T(n))$ if $\lim_{n\to\infty} T'(n)/T(n) = 0$. Similarly, if $S, S' : \Sigma^* \to \mathrm{N}$, then $S'(x) = o(S(x))$ if $\lim_{|x|\to\infty} S'(x)/S(x) = 0$. We adhere to the customary convention that $T(n) \geq n+1$ ($S(x) \geq |x|+1$) for any Turing machine running time T (S, respectively).

The following proposition is one of the main theorems of Geske, Huynh, and Seiferas [6]. (See also the paper by Geske, Huynh, and Selman [7].)

PROPOSITION 2.1. *If $S(x)$ is fully time constructible, then there is a language $L \in$ DTIME($O(S(x))$) such that for every function S', if $S'(x)\log S'(x) = o(S(x))$, then every Turing machine M that accepts L requires more than $S'(x)$ steps for all but finitely many input strings x.*

2.2. Distributional problems. A *distribution function* $\mu : \{0,1\}^* \to [0,1]$ is a nondecreasing function from strings to the closed interval $[0,1]$ that converges to 1. The corresponding *density function* μ' is defined by $\mu'(0) = \mu(0)$ and $\mu'(x) = \mu(x) - \mu(x-1)$ for $x \neq 0$. Clearly, $\mu(x) = \sum_{y\leq x} \mu'(y)$. For any subset of strings S, we will denote by $\mu(S) = \sum_{x\in S} \mu'(x)$ the probability of the event S. Define $u_n = \mu(\{x \mid |x| = n\})$. For each n, let $\mu'_n(x)$ be the conditional probability of x in $\{x \mid |x| = n\}$. That is, $\mu'_n(x) = \mu'(x)/u_n$, if $u_n > 0$, and $\mu'_n(x) = 0$ for $x \in \{x \mid |x| = n\}$, if $u_n = 0$.

A function μ from Σ^* to $[0,1]$ is *computable in polynomial time* [13] if there is a polynomial time-bounded transducer T such that for every string x and every positive integer n, $|\mu(x) - T(x, 1^n)| < \frac{1}{2^n}$. We restrict our attention to distributions μ that

are computable in polynomial time. If the distribution function μ is computable in polynomial time, then the density function μ' is computable in polynomial time. (The converse is false unless P = NP [8].)

Levin [14] defines a function f from Σ^* to nonnegative reals to be *linear on μ-average* if

$$\sum_{|x|\geq 1} \mu'(x)\frac{f(x)}{|x|} < \infty, \tag{2.1}$$

and f is *polynomial on μ-average* if f is bounded by a polynomial of a function that is linear on μ-average. Thus, a function f is polynomial on μ-average if and only if there is an integer $k > 0$ such that

$$\sum_{|x|\geq 1} \mu'(x)\frac{(f(x))^{1/k}}{|x|} < \infty. \tag{2.2}$$

Finally, Levin defines Average-P to be the class of distributional problems (L, μ), where L is a language and μ is a polynomial time computable distribution, such that L can be decided by some Turing machine M whose running time T_M is polynomial on μ-average.

Starting with Levin, a number of researchers have observed that the more naive notion that for each length n the expectation of the running time T_M of a Turing machine M that accepts L is bounded above by a polynomial in n

$$\sum_{|x|=n} \mu'_n(x)T_M(x) \leq p(n), \tag{2.3}$$

is unsuitable for a number of good reasons [8, 4]. The definition that arises from using (2.3) is not robust under functional composition of algorithms. (There are distributional problems A and B so that A can be solved in average polynomial time by using an oracle B, B can be solved in average (or even deterministic) polynomial time, and A cannot be solved in average polynomial time.) Nor is the definition closed under application of polynomials. (There are functions f that satisfy (2.3) for which f^2 does not.) As a consequence, from (2.3), one loses machine independence of the definition of the class of average polynomial time.

Levin's definition is just as intuitively justified as that given by (2.3). This can be seen as follows: The worst-case time complexity for P requires for all n, and for all x such that $|x| = n$, that $T_M(x) \leq p(n)$. Therefore for the case of bounding complexity on the average by some polynomial $p(n)$, it appears natural to require for all n that (2.3) holds. However, $T_M(x) \leq n^k$ is equivalent to $T_M(x)/n^k \leq 1$, which is also equivalent to $(T_M(x))^{1/k}/n \leq 1$. Thus we might as well take the expectation now, after this manipulation, which results in the established definition. (We will discuss this point further in section 4.)

2.3. Hardy's class of logarithmico-exponential functions. We will need the notion of a class of functions $\mathcal{L}$ defined by Hardy [10], called the *logarithmico-exponential functions*. Every function in $\mathcal{L}$ is a real-valued function of one variable that is defined on all sufficiently large real numbers. The class $\mathcal{L}$ is defined to be the smallest class of functions containing

(i) every constant function $t(x) = c$ for all real c,
(ii) the identity function $t(x) = x$

and that are closed under the following operations:

(i) if $t(x)$ and $s(x)$ belong to $\mathcal{L}$, then so does $t(x) - s(x)$;
(ii) if $t(x)$ belongs to $\mathcal{L}$, then so does $e^{t(x)}$;
(iii) if $t(x)$ is eventually positive and belongs to $\mathcal{L}$, then so does $\ln(t(x))$.

The closure properties are more inclusive than perhaps they first appear. For example, if $t(x)$ and $s(x)$ are in $\mathcal{L}$, then so are $t(x) + s(x)$, $t(x)s(x)$, and $t(x)/s(x)$. Also, functions such as $\sqrt[k]{x} = e^{\frac{1}{k}\ln x}$ and $e^{c\sqrt{\ln x}/\ln\ln x}$ belong to $\mathcal{L}$.

Hardy [10] proved the following facts regarding the *logarithmico-exponential functions*. He showed that every function in $\mathcal{L}$ is either eventually positive or eventually negative or identically zero. Note that it is easily shown by induction that every function $t(x)$ in $\mathcal{L}$ is differentiable and its derivative $t'(x)$ is also in $\mathcal{L}$ (thus infinitely differentiable). Thus, it follows that every function in $\mathcal{L}$ is eventually monotonic. The main theorem of Hardy regarding the logarithmico-exponential functions is that they form an asymptotic hierarchy: for any $t(x)$ and $s(x)$ in $\mathcal{L}$, either $t(x) = o(s(x))$ or $s(x) = o(t(x))$, or there exists a nonzero constant c, such that $\lim_{x\to\infty} t(x)/s(x) = c$.

Let $f^{(\ell)}$ denote the function that iterates ℓ applications of f. That is, $f^{(1)}(x) = f(x)$ and $f^{(\ell+1)}(x) = f(f^{(\ell)}(x))$ for $\ell \geq 1$. Hardy proved [9] that for every function $t \in \mathcal{L}$, if $\lim_{x\to\infty} t(x) = \infty$, then there is some constant ℓ so that $\log^{(\ell)}(x) = o(t(x))$, as well as $t(x) = o(\exp^{(\ell)}(x))$. Informally, a logarithmico-exponential function that goes to infinity cannot increase more slowly than every iterated logarithm function nor faster than every iterated exponential function.

Hardy's purpose in introducing the class of logarithmico-exponential functions was to provide what he called "a scale of infinities." We propose to use only logarithmico-exponential functions as time-bounds in defining average-case complexity classes. Indeed, we propose that for most purposes it suffices to use only logarithmico-exponential functions as time-bounds for complexity classes in general. (To be pedantic for a moment, we believe that it suffices to consider as time-bounds for complexity classes only functions $f : \mathrm{N} \to \mathrm{N}$ that result from first restricting some logarithmico-exponential function to the domain of natural numbers and then further restricting the range to N by taking the floor of the result. For notational ease, we will call these logarithmico-exponential functions as well.) These functions are at the same time sufficiently well behaved and sufficiently expressive for the purpose of bounding time complexity of most meaningful classes of computational problems. While functions such as $x(1+\sin x)$ or $e^{x^2 \sin x}$ that gyrate infinitely often as $x \to \infty$ are excluded, as are functions that are, say, bounded on even length strings but tend to infinity on odd length strings, it is reasonable to assume that bounding the running time, average case or otherwise, of *a class of problems* by such a function is hardly necessary or natural. This is in contrast to the case of bounding the complexity of an individual problem, say, some number theoretic computation where the problem is interesting only for certain lengths. The elegance that results from the exclusion of these pathological cases compensates well for the price we pay for its restriction. However, by restricting to the logarithmico-exponential functions we do lose some very slow growing functions such as $\log^* x$ or the inverse of Ackermann's function. We leave it as an interesting open problem to extend the class of Hardy's logarithmico-exponential functions to include functions such as $\log^* x$ and still retain all its desirable properties.

Our proof of the first hierarchy theorem in the next section depends crucially on the properties of Hardy's logarithmico-exponential functions.

We will need the following lemma.

LEMMA 2.2. *If $t(n)$ belongs to $\mathcal{L}$, $\lim_{n\to\infty} t(n) = +\infty$, and $t(n)$ is polynomially bounded, then there exist a constant c and an integer k such that, for all $n \geq c$ and*

$a > 1$,

$$t(an) < a^k t(n).$$

Proof. Let k be an integer such that $\lim_{n\to\infty} t(n)/n^k = 0$. Since $t(n)$ is polynomially bounded, such an integer exists. Let $q(x) = t(x)/x^k$. Then $q(x)$ belongs to $\mathcal{L}$ and $\lim_{n\to\infty} q(n) = 0$. Since $q(n)$ is eventually positive and approaches 0, it is eventually monotonic decreasing. This is because $q(x)$ is not identically 0, it cannot possibly be monotonic increasing, and these are the only alternatives for functions in $\mathcal{L}$. Thus, for some constant c, if $x \geq c$, then $q(x)$ is monotonic decreasing. It follows that

$$\begin{aligned}\frac{t(an)}{a^k t(n)} &= \frac{q(an)(an)^k}{a^k q(n) n^k} \\ &= \frac{q(an)}{q(n)} \\ &< 1\end{aligned}$$

for all $n \geq c$ and for all $a > 1$. (Note that c does not depend on a.) This implies that $t(an) < a^k t(n)$. □

3. The first hierarchy theorem. Ben-David et al. [4] propose the following definition.

DEFINITION 3.1 (Ben-David et al.). *For a time complexity function $T : \mathrm{N} \to \mathrm{N}$, a function f is T* on *μ-average if f is bounded by T of a function that is linear on μ-average; i.e., $f(x) \leq T(\ell(x))$, where ℓ is linear on μ-average.* AverDTime($T(n)$) *denotes the class of distributional problems (L, μ), where L is a language and μ is a polynomial time computable distribution, such that L can be decided by some Turing machine M whose running time T_M is T on μ-average.*

If T is monotonically increasing and thus invertible, then f is T on μ-average if and only if

$$\sum_{|x|\geq 1} \mu'(x) \frac{T^{-1}(f(x))}{|x|} < \infty. \tag{3.1}$$

This definition is a direct adaptation of Levin's notion of average polynomial time, where the time-bound T is polynomially bounded. Indeed, Average-P = $\bigcup_k$ AverDTime(n^k).

3.1. Inadequacy of Definition 3.1. Definition 3.1 for time-bounds T beyond polynomial time has serious difficulties, which we will now explain.

It follows from the result of Geske, Huynh, and Seiferas [6], our Proposition 2.1, that there exists a language $L \in$ DTIME(2^{2^n}) that cannot be recognized in time 3^n almost everywhere—every Turing machine that accepts L requires more than 3^n steps on all but some finite number of inputs. Yet, it follows easily from the definition that every distributional problem that belongs to AverDTime(2^{2^n}) also belongs to AverDTime(2^n). This is inconceivable. How can a language L require more than 3^n time almost everywhere but be 2^n on the μ-average for *every* distribution μ?

To see that AverDTime(2^{2^n}) $\subseteq$ AverDTime(2^n), let M accept L in time T_M, where T_M is $2^{2n} = 4^n$ on μ-average. Then, by definition,

$$\sum_{|x|\geq 1} \mu'(x) \frac{\log_4(T_M(x))}{|x|} < \infty.$$

Thus,

$$2\sum_{|x|\geq 1}\mu'(x)\frac{\log_4(T_M(x))}{|x|}=\sum_{|x|\geq 1}\mu'(x)\frac{\log_2(T_M(x))}{|x|}<\infty$$

also; thus according to Definition 3.1, T_M is also 2^n on μ-average.

The same argument implies that $\mathrm{AverDTime}(2^n)=\mathrm{AverDTime}(c^n)$ for all constants $c>0$. This is another weakness of the definition. Usually a time complexity class should be defined in such a way that it is sufficiently fine to distinguish varying inherent time complexities of problems. In other words, one likes to have a fine time hierarchy theorem. The fact that $\mathrm{AverDTime}(2^n)=\mathrm{AverDTime}(4^n)$ prevents us from having such a *fine* theorem.

Previous researchers have raised the question of hierarchies for average time but in all cases have been implicitly stymied by the inadequacy that we illuminate here. For example, Li and Vitányi [15] and Ben-David et al. [4] obtain results demonstrating languages for which average-case and worst-case complexity are the same. However, both use distributions that cannot be computed in polynomial time, and the latter uses a nonstandard notion of worst-case complexity. Reischuk and Schindelhauer [20] introduce a concept of average-time complexity according to "rankable" distributions that possess a tight hierarchy. Their concept is different than Levin's notion. Belanger and Wang [3] contains a discussion. In addition, Belanger and Wang [3] obtain a weak hierarchy theorem for the notion of average time given by Definition 3.1.

We believe that it is interesting and useful to have a meaningful, robust definition of average exponential time. Our notion of average-time complexity is supported by the hierarchy theorem that we will develop in section 4, an important test of its meaningfulness and robustness. With regard to complexity theory studies, average-case exponential time is interesting for the same reasons as worst-case exponential time. For example, in general one would like to understand why certain problems are complete and to grasp the inherent properties that make a set complete. Natural complete problems are not as common for exponential time as they are for smaller classes such as NP. However, exponential time permits techniques and constructions that are not possible with smaller time-bounds. For this reason, much more is known about complete problems for exponential time than is currently known about NP complete problems [12]. We can anticipate the same situation for average-case complexity. With regard to applications of average-case complexity theory, here too, average-case exponential time is worth exploring. For example, any distributional problem that is complete for average exponential time is *provably* intractable even in the average case. Therefore, such a problem might be useful as a basis for cryptographic protocols.

3.2. First hierarchy theorem. We can prove a hierarchy theorem when we restrict our attention to functions that are bounded above by some polynomial. More precisely, we shall require that the time complexity bounds T and T' belong to Hardy's class of logarithmico-exponential functions $\mathcal{L}$ and that they are bounded above by a polynomial. Under these conditions, we show that if $T'(n)\log T'(n)=o(T(n))$, then there is a distributional problem (L,μ) in $\mathrm{AverDTime}(T(n))$ that does not belong to $\mathrm{AverDTime}(T'(n))$.

THEOREM 3.2. *Let T,T' : N $\to$ N be logarithmico-exponential functions and assume that T is fully time constructible. Assume that $T'(n)\log T'(n)=o(T(n))$ and that $T'(n)$ is bounded above by a polynomial. Then*

$$\mathrm{AverDTime}(T'(n))\subset\mathrm{AverDTime}(T(n)).$$

The general idea of the proof is to use Proposition 2.1 to obtain a language L that is in DTIME($T(n)$) but almost everywhere not in DTIME($T'(n)$). Then we use properties of the logarithmico-exponential functions to show that (L, μ) belongs to AverDTime($T(n)$) but does not belong to AverDTime($T'(n)$).

We need the following lemmas.

LEMMA 3.3. *Let T be a logarithmico-exponential and fully time constructible function. If $L \in$ DTIME($O(T(n))$), then for every polynomial time computable distribution μ, $(L, \mu) \in$ AverDTime($T(n)$).*

Proof. By the hypotheses, $T(n) \geq n + 1$ for all n, and T is monotonically increasing. Since T is logarithmico-exponential, either (i) $n = o(T(n))$ or (ii) for some constant $c \geq 1$, $T(n) \leq cn$ for all sufficiently large n. If case (i) holds, then the well-known linear-speedup theorem applies [11]. In this case $L \in$ DTIME($T(n)$) and the result follows immediately. If case (ii) holds, then $L \in$ DTIME(cn). Let M be a Turing machine that witnesses $L \in$ DTIME(cn), and let T_M be the running time of M. By definition, T_M is linear on μ-average. Thus, by Definition 3.1 and the fact that $T(n) \geq n + 1$, T_M is T on μ-average. This completes the proof. □

LEMMA 3.4. *If $a, b > 0$ and $1/h = 1/a + 1/b$, then $\min(a, b)/2 \leq h \leq \min(a, b)$.*

Proof.

$$h = \frac{1}{\frac{1}{a} + \frac{1}{b}} = \frac{ab}{a+b} = \min(a,b) \cdot \frac{\max(a,b)}{a+b} \leq \min(a,b).$$

Also,

$$\frac{1}{h} \leq \frac{2}{\min(a,b)},$$

and therefore $h \geq \min(a, b)/2$. □

As a corollary, if $a(n)$ and $b(n)$ are functions such that both $a(n), b(n) \to +\infty$, then $h(n) \to +\infty$ also, where

$$h(n) = \frac{a(n)b(n)}{a(n) + b(n)},$$

but no faster than either $a(n)$ and $b(n)$. Furthermore, if both $a(n), b(n) \in \mathcal{L}$, the class of logarithmico-exponential functions, then so is $h(n)$.

Now we prove our theorem.

Proof. Since $T'(n) = o(T(n))$, it follows that

$$\text{AverDTime}(T'(n)) \subseteq \text{AverDTime}(T(n)).$$

We need to show that the classes are distinct. Without loss of generality, we assume that $\lim_{n\to\infty} T'(n) = +\infty$. Otherwise, the theorem is trivial.

Define sequences α_n and β_n by

$$\alpha_n = \frac{T(n)}{T'(n) \log T'(n)}$$

and

$$\beta_n = \frac{\log T'(n)}{\log \log T'(n)}.$$

Then, both

$$\alpha_n \text{ and } \beta_n \to +\infty.$$

Take $B_n = \sqrt{\alpha_n \beta_n/(\alpha_n + \beta_n)}$. By Lemma 3.4, $B_n \to +\infty$, and yet $B_n = o(\alpha_n)$ and $B_n = o(\beta_n)$.

Define $S(n)$ by $S(n) = B_n \cdot T'(n)$. We claim that

$$\lim_{n\to\infty} \frac{S(n)\log S(n)}{T(n)} = 0.$$

In fact,

$$\begin{aligned}\frac{S(n)\log S(n)}{T(n)} &= \frac{B_n \cdot T'(n)\cdot(\log B_n + \log T'(n))}{T(n)} \\ &= B_n \log B_n \frac{T'(n)}{T(n)} + B_n \frac{1}{\alpha_n} \\ &= B_n \frac{1}{\alpha_n} + \frac{B_n \log B_n}{\log T'(n)} \cdot \frac{1}{\alpha_n}.\end{aligned}$$

We have $B_n/\alpha_n \to 0$. Also, $B_n/\beta_n \to 0$, which implies that $\log B_n \le \log \beta_n \le \log\log T'(n)$, so for the second term, we have

$$\frac{B_n \log B_n}{\log T'(n)} = o\left(\beta_n \cdot \frac{\log\log T'(n)}{\log T'(n)}\right) = o(1).$$

Thus, Proposition 2.1 applies: there is a language $L \in \mathrm{DTIME}(O(T(n)))$ such that for every Turing machine M that accepts L there is a constant n_0 such that the running time T_M requires more than $S(|x|)$ steps for all inputs of length $\ge n_0$. That is, $T_M(x) > S(|x|)$ for all x such that $|x| \ge n_0$. By Lemma 3.3, $(L,\mu) \in \mathrm{AverDTime}(T(n))$ for every polynomial time computable distribution μ. Now our task is to define a polynomial time computable distribution μ such that $(L,\mu) \notin \mathrm{AverDTime}(T'(n))$.

By our assumption that T' is bounded by a polynomial and belongs to $\mathcal{L}$, by Lemma 2.2, there is an integer k and a constant c such that $T'(an) < a^k T'(n)$, for all $a > 1$ and all $n \ge c$. In particular $T'(B_n^{1/k} \cdot n) < B_n \cdot T'(n)$. Since $T' \in \mathcal{L}$ and $\lim_{n\to\infty} T'(n) = +\infty$, it follows that T' is monotonically increasing. Hence,

$$T'^{-1}(B_n \cdot T'(n)) \ge B_n^{1/k} \cdot n. \tag{3.2}$$

Now, let $b_n = B_n^{1/k}$. Since $b_n \in \mathcal{L}$ and $b_n \to +\infty$, there is some constant ℓ so that $\log^{(\ell)}(n) = o(b_n)$. For this value ℓ, there is some value n_ℓ so that for all $n \ge n_\ell$, the expression

$$\frac{1}{n\log n \log\log n \cdots (\log^{(\ell)} n)^2}$$

is defined. Define

$$a_n = \frac{1}{n\log n \log\log n \cdots (\log^{(\ell)} n)^2}$$

for all $n \ge n_\ell$. Define $a_n = 1$ otherwise. Then, the series $\sum_{n=1}^{\infty} a_n$ converges, but the series $\sum_{n=1}^{\infty} a_n b_n$ diverges.

Let μ be the distribution function whose density function is defined by

$$\mu'(x) = \frac{1}{s} \cdot a_{|x|} \cdot \frac{1}{2^{|x|}},$$

where $\sum_{n=1}^{\infty} a_n = s$.

Clearly, μ is polynomial time computable.

Let M be a Turing machine that accepts L. For all x such that $|x| \geq n_0$, $T_M(x) > S(|x|) = B_{|x|}T'(|x|)$. Hence, by (3.2),

$$T'^{-1}(T_M(x)) \geq B_n^{1/k} \cdot |x| = b_{|x|}|x|.$$

Thus,

$$\sum_{|x|\geq \max(n_0,c)} \mu'(x) \cdot \frac{T'^{-1}(T_M(x))}{|x|} \geq \sum_{n\geq \max(n_0,c)} \frac{a_n b_n}{s} = \infty,$$

so $(L,\mu) \notin \text{AverDTime}(T'(n))$. □

COROLLARY 3.5. *For $c \geq 1$ and $\epsilon > 0$,* $\text{AverDTime}(n^c) \subset \text{AverDTime}(n^{c+\epsilon})$.

4. The new definition and second hierarchy theorem. We have shown that there is a problem with the existing definition of average-case complexity for time-bounds beyond those that are bounded by a polynomial. Now we will develop a new definition of "T on the μ-average," based on the following guiding principles.

1. Our definition should be essentially the same as Levin's notion when we apply it to polynomial time-bounds.

2. If L belongs to $\text{DTIME}(T(n))$ for some time-bound T, then any distributional problem (L,μ) is T on the μ-average.

3. If L is not in $\text{DTIME}(T(n))$ almost everywhere, then, for any distibutional problem (L,μ), L is not T on the μ-average.

To begin, let us revisit the intuitive justification that we discussed in section 2.2. Let $T \in \mathcal{L}$ be some fully time constructible function, let T_M be some Turing machine running time, and let μ be some polynomial time computable distribution, for which we want to say that T_M is T on the μ-average. As earlier, we might want to say that $T_M(x) \leq T(|x|)$ for a μ-average x. Equivalently, we want $T^{-1}(T_M(x)) \leq |x|$, or

$$\frac{T^{-1}(T_M(x))}{|x|} \leq 1,$$

for a μ-average x. Thus, let us require that the expectation be bounded above by 1:

$$\mathcal{E}\left[\frac{T^{-1}(T_M(x))}{|x|}\right] = \sum_{|x|\geq 1} \mu'(x) \cdot \frac{T^{-1}(T_M(x))}{|x|} \leq 1. \tag{4.1}$$

At this point Levin and subsequent researchers, including Ben-David et al., took (4.1) to say that the expectation $\mathcal{E}$ over all x is finite. But, the requirement of (4.1) adds nothing new. One can always modify M so that, for some fixed but arbitrarily long initial segment of inputs, M takes little time. However, the tail of a convergent sum can be made arbitrarily small, so the total sum with respect to the new machine is bounded by 1.

Since we are primarily concerned with the asympotic behavior of algorithms on average inputs, we want to avoid the possibility that the sum in the expectation is dominated by an initial segment. For this reason, it is perfectly reasonable, with identical justification, to require that for all $n \geq 1$ the expectation of $\frac{T^{-1}(T_M(x))}{|x|}$, for all x of length $|x| \geq n$, be bounded above by 1:

$$\mathcal{E}_{|x|\geq n}\left[\frac{T^{-1}(T_M(x))}{|x|}\right] = \sum_{|x|\geq n} \mu'_{\geq n}(x) \cdot \frac{T^{-1}(T_M(x))}{|x|} \leq 1, \tag{4.2}$$

where $\mu'_{\geq n}$ is the conditional probability distribution on $\{z \mid |z| \geq n\}$. That is, let $W_n = \mu(\{z \mid |z| \geq n\})$; for x of length $|x| \geq n$,

$$\mu'_{\geq n}(x) = \mu'(x)/W_n \text{ if } W_n > 0 \quad \text{and} \quad \mu'_{\geq n}(x) = 0 \text{ if } W_n = 0.$$

Thus (4.2) is equivalent to requiring that, for all $n \geq 1$,

$$\sum_{|x|\geq n} \mu'(x) \cdot \frac{T^{-1}(T_M(x))}{|x|} \leq W_n. \tag{4.3}$$

This is the condition that we will take for our definition.

Comparing this with the simpler requirement that

$$\sum_{|x|\geq 1} \mu'(x) \cdot \frac{T^{-1}(T_M(x))}{|x|} < \infty, \tag{4.4}$$

we require not only that the infinite sum converges but that it converges at a certain rate. Note that $W_n \to 0$ as $n \to \infty$.

A persistent criticism of the theory of average-time complexity is that for any reasonable definition of "default" distribution on the set of all positive integers, such as $u_n = 1/n^2$ or $1/n^3$, inevitably a disproportionate weight of the total distribution is on the first few (or few dozen?) integers. For instance, in (4.4), the first terms might be somewhat dominating, thus masking the true asymptotic behavior. The requirement that we impose in (4.3), which stipulates that each sum is bounded above, explicitly addresses this criticism. As it turns out, we will demonstrate in Theorem 4.2 that our definition agrees with Levin's definition for polynomial time-bounds and reasonable distributions. Thus, this criticism is indeed unfounded for those cases. However, this phenomenon of masking the asymptotic behavior by the concentration of measures on the initial few strings is magnified at superpolynomial time-bounds; in this case, the criticism is valid and our formulation corrects the problem.

Thus, we arrive at our definition.

DEFINITION 4.1. *For any time constructible function $T(n) \in \mathcal{L}$, a function f is T on the μ-average*[1] *if, for all $n \geq 1$,*

$$\sum_{|x|\geq n} \mu'(x) \cdot \frac{T^{-1}(f(x))}{|x|} \leq W_n. \tag{4.5}$$

AVTIME($T(n)$) *denotes the class of distributional problems (L, μ), where L is a language and μ is a polynomial time distribution, such that L can be decided by some Turing machine M whose running time T_M is T on the μ-average.*

To summarize, we departed from the previous definition by imposing two new criteria. We insist (i) that the expectation given in (4.1) is bounded by 1, rather than asking only that it be finite, and we insist (ii) that *each* conditional expectation given in (4.2) is bounded (indeed, bounded by 1). If we were to have added either one of these requirements without the other, the result would have been trivially equivalent to the previous definition. We have already shown that requirement (i) alone adds nothing new. To see that requiring each conditional expectation to converge adds nothing new, we simply note that every tail series of a convergent series converges as well. Thus, it is the combination of the two modifications, which amounts to restricting the *rate of convergence*, that adds something new.

[1] We are redefining this expression, and henceforth all references to this expression will refer to the meaning given herein.

4.1. Equivalence theorem. Before proceeding to establish that Definition 4.1 satisfies our guiding principles, we introduce the following Condition W. Condition W will limit our consideration to distributions that do not put too much weight on the first few strings. (For example, we don't consider $u_n = 2^{-n}$ to be an acceptable default distribution on the natural numbers.)

CONDITION W. *There exists $s > 0$ such that $W_n = \Omega\left(\frac{1}{n^s}\right)$.*

All the usual distributions that have been used in the past in this area satisfy Condition W. These include various uniform distributions on graph properties, as well as distributions used by Levin [14], Gurevich [8], and others to prove their average-case hard problems. We also note that all the previous work on average-case NP-hard problems is not altered by our reformulation of average-time complexity.

Now we prove that average polynomial time under Levin's definition is unchanged by our new definition, for distributional problems that satisfy Condition W.

THEOREM 4.2. *Let μ be a polynomial time computable distribution that satisfies Condition W. Then (L, μ) belongs to* Average-P *if and only if (L, μ) belongs to* $\bigcup_k \mathrm{AVTIME}(n^k)$.

Proof. Since our definition requires at least as much as the old definition, the inclusion in one direction is trivial. So, let μ be a polynomial time computable distribution that satisfies Condition W, let M accept L with running time T_M, and let k be a positive integer such that

$$\sum_{|x|\geq 1} \mu'(x)\frac{(T_M(x))^{1/k}}{|x|} = C < \infty.$$

Define $p(n) = (Cn)^k$, and observe that

$$\sum_{|x|\geq 1} \mu'(x)\frac{p^{-1}(T_M(x))}{|x|} \leq 1.$$

Since μ satisfies Condition W, there exists $s > 0$ such that $W_n = \Omega\left(\frac{1}{n^s}\right)$. Define the polynomial q by $q(n) = p(n^c)$ for $c > s + 2$. We will show that

$$\sum_{|x|\geq n} \mu'(x)\frac{q^{-1}(T_M(x))}{|x|} \leq W_n$$

for all but a finite number of n. Observe that this suffices to complete our proof. Namely, let n_0 be a fixed positive integer; if (4.5) holds for all $n \geq n_0$, then as we explained above, we can modify M to contain a lookup table in order to quickly decide all strings of length smaller than n_0. By doing so, we make the first terms of the sum sufficiently small so that the nth sum is bounded by W_n for all $n \geq 1$.

Now our goal is to estimate the sum

$$\sum_{k\geq n}\sum_{|x|=k} \mu'(x)\cdot\frac{q^{-1}(T_M(x))}{k}$$

for all $n \geq 1$.

Note that $q^{-1}(y) = (p^{-1}(y))^{1/c}$. Recall that $u_k = \mu(\{z \mid |z| = k\})$. By convexity, for all k such that $u_k > 0$,

$$\sum_{|x|=k} \frac{\mu'(x)}{u_k}\cdot\frac{q^{-1}(T_M(x))}{k} \leq \left[\sum_{|x|=k} \frac{\mu'(x)}{u_k}\cdot\frac{p^{-1}(T_M(x))}{k^c}\right]^{1/c}.$$

Thus,

$$\begin{aligned}\sum_{k\geq n}\sum_{|x|=k}\mu'(x)\cdot\frac{q^{-1}(T_M(x))}{k} &= \sum_{\substack{k\geq n\\ u_k>0}}\sum_{|x|=k}\mu'(x)\cdot\frac{q^{-1}(T_M(x))}{k}\\ &= \sum_{\substack{k\geq n\\ u_k>0}} u_k \sum_{|x|=k}\frac{\mu'(x)}{u_k}\cdot\frac{q^{-1}(T_M(x))}{k}\\ &\leq \sum_{\substack{k\geq n\\ u_k>0}} u_k \left[\sum_{|x|=k}\frac{\mu'(x)}{u_k}\cdot\frac{p^{-1}(T_M(x))}{k^c}\right]^{1/c}.\end{aligned}$$

Since for all $n \geq 1$,

$$\sum_{|x|=n}\mu'(x)\cdot\frac{p^{-1}(T_M(x))}{n}\leq 1,$$

we have

$$\begin{aligned}\sum_{k\geq n}\sum_{|x|=k}\mu'(x)\cdot\frac{q^{-1}(T_M(x))}{k} &\leq \sum_{\substack{k\geq n\\ u_k>0}} u_k\left[\frac{1}{u_k}\cdot\frac{1}{k^c}\cdot k\right]^{1/c}\\ &= \sum_{k\geq n}\left(\frac{u_k}{k}\right)^{1-1/c}.\end{aligned}$$

By Hölder's inequality,

$$\sum_{k\geq n}\left(\frac{u_k}{k}\right)^{1-1/c}\leq\left(\sum_{k\geq n}u_k\right)^{1/p}\left(\sum_{k\geq n}\frac{1}{k^{c-1}}\right)^{1/q},$$

where $1/p = 1 - 1/c$ and $q = c$. Hence,

$$\sum_{k\geq n}\sum_{|x|=k}\mu'(x)\cdot\frac{q^{-1}(T_M(x))}{k}\leq W_n^{1-1/c}\cdot\left(\sum_{k\geq n}\frac{1}{k^{c-1}}\right)^{1/c}.$$

Finally, we have

$$\begin{aligned}\sum_{k\geq n}\sum_{|x|=k}\mu'(x)\cdot\frac{q^{-1}(T_M(x))}{k} &\leq W_n^{1-1/c}\cdot\left(\int_{n-1}^{\infty}\frac{1}{x^{c-1}}dx\right)^{1/c}\\ &= W_n\cdot\left(\frac{1}{(c-2)n^{c-2}W_n}\right)^{1/c}.\end{aligned}$$

As we have taken $c > s + 2$, and $W_n = \Omega\left(\frac{1}{n^s}\right)$, the last term is at most W_n for almost all n. □

4.2. Second hierarchy theorem. Now we verify that our definition satisfies the remaining guiding principles, and we prove a tight hierarchy theorem for AVTIME classes.

THEOREM 4.3. *Let $T \in \mathcal{L}$ be fully time constructible. If $L \in \mathrm{DTIME}(T(n))$, then for every polynomial time computable distribution μ, $(L, \mu) \in \mathrm{AVTIME}(T(n))$.*

The proof is easy: for any Turing machine that accepts L in time T, the ratio $\frac{T^{-1}(T_M(x))}{|x|}$ is ≤ 1 for every input x.

THEOREM 4.4. *Let $T \in \mathcal{L}$ be fully time constructible and suppose that $L \notin$ DTIME$(T(n))$ almost everywhere. Then, for every polynomial time computable distribution μ, $(L, \mu) \notin$ AVTIME$(T(n))$.*

Again, the proof is easy. For any Turing machine that accepts L, choose n_M so that $T_M(x) > T(|x|)$ for all x such that $|x| \geq n_M$. Observe that the ratio $\frac{T^{-1}(T_M(x))}{|x|}$ is > 1 for every input x such that $|x| \geq n_M$. The proof follows immediately by observing that

$$\sum_{|x| \geq n_M} \mu'(x) \cdot \frac{T^{-1}(T_M(x))}{|x|} > W_{n_M}. \tag{4.6}$$

The fact that these theorems follow immediately attests to the naturalness of Definition 4.1.

THEOREM 4.5. *Let $T, T' : \mathrm{N} \to \mathrm{N}$ be logarithmico-exponential functions and assume that T and T' are fully time constructible. Assume $T'(n) \log T'(n) = o(T(n))$. Then*

$$\mathrm{AVTIME}(T'(n)) \subset \mathrm{AVTIME}(T(n)).$$

Further, there is a language L such that for every polynomial time computable distribution μ, $(L, \mu) \in$ AVTIME$(T(n))$, but $(L, \mu) \notin$ AVTIME$(T'(n))$.

When we use Proposition 2.1, the proof follows immediately from Theorems 4.3 and 4.4. This result is as tight as the well-known Hartmanis–Stearns hierarchy theorem [11] for deterministic time.

COROLLARY 4.6. *For all $c \geq 1$ and for all $\epsilon > 0$,* AVTIME$(n^c) \subset$ AVTIME$(n^{c+\epsilon})$. *For all $c > 1$ and for all $\epsilon > 0$,* AVTIME$(c^n) \subset$ AVTIME$((c+\epsilon)^n)$.

In analogy with traditional complexity theory, consider the following average-case complexity classes:

(i) AVP $= \bigcup_{k \geq 1}$ AVTIME(n^k).
(ii) AVE $= \bigcup_{k \geq 1}$ AVTIME(2^{cn}).
(iii) AVEXP $= \bigcup_{k \geq 1}$ AVTIME(2^{n^k}).

COROLLARY 4.7. AVP $\subset$ AVE *and* AVE $\subset$ AVEXP.

A language L is *bi-immune* to a complexity class $\mathcal{C}$ if L is infinite, no infinite subset of L belongs to $\mathcal{C}$, and no infinite subset of $\overline{L}$ belongs to $\mathcal{C}$. Balcázar and Schöning [1] proved that for every time constructible function T, L does not belong to DTIME$(T(n))$ almost everywhere if and only if L is bi-immune to DTIME$(T(n))$.

Recall that DistNP is the class of distributional problems (L, μ) such that $L \in$ NP and μ is computable in polynomial time [14, 4]. The important open question that motivates studies of average-time complexity classes is the question of whether DistNP $\subseteq$ AVP.

COROLLARY 4.8. *If* NP *contains a language L that is bi-immune to* P*, then for every polynomial time computable distribution μ, the distributional problem (L, μ) belongs to* DistNP *but does not belong to* AVP.

The proof follows from Theorem 4.4. A weaker conclusion is known to follow from a weaker hypothesis. To wit, Lutz and Mayordomo [16] proved that if NP contains a language L that is bi-immune to P, then E $\neq$ NE. Ben David et al. [4] proved that if E $\neq$ NE, then there is a tally language L in NP $-$ P such that the distributional problem (L, ν) belongs to DistNP but does not belong to Average-P, where $\nu'(1^n) = n^{-2}$.

Furthermore, Mayordomo [17] proved that if NP does not have p-measure 0, then NP contains a language that is bi-immune to P. Thus, the following corollary follows immediately.

COROLLARY 4.9. *If* NP *does not have p-measure* 0*, then there is a language* L *such that, for every polynomial time computable distribution* μ*, the distributional problem* (L, μ) *belongs to* DistNP *but does not belong to* AVP.

Schuler and Yamakami [21] have proved independently that if L is bi-immune to P, then there is a polynomial time computable distribution μ such that (L, μ) does not belong to Average-P. They observed from this result that if DistNP $\subseteq$ Average-P, then NP has p-measure 0. By Theorem 4.2, $(L, \mu) \notin$ Average-P for every bi-immune set L and every polynomial time computable distribution μ that satisfies Condition W. Thus, if NP does not have p-measure 0, then there is a language L such that for every polynomial time computable distribution μ that satisfies Condition W, the distributional problem (L, μ) belongs to DistNP but does not belong to Average-P.

4.3. Pathological distributions. We arrived at our current definition of average-case complexity for arbitrary time-bounds after a careful analysis of the intuitive justifications and after considering the demands of a well-formed complexity theory. The new definition is supported by the equivalence theorem, Theorem 4.2, and the second hierarchy theorem, Theorem 4.5.

Here we briefly address the exceptional cases where the distribution does not satisfy Condition W, so that our equivalence theorem, Theorem 4.2, does not apply. We show in this case that our notion of polynomial on the μ-average indeed *differs* from that of Levin. Thus, the restriction to distributions that satisfy Condition W in Theorem 4.2 is essential.

To illustrate, let's consider a (pathological) distribution μ where $u_n = 1/2^n$; i.e., all strings of length n have total measure $1/2^n$. It follows from the theorem of Geske, Huynh, and Seiferas [6], our Proposition 2.1, that there is a language L that is decidable in time $2^n/n$ but that almost everywhere requires more than, say, $2^n/n^3$ steps. Then according to Levin's definition, the distributional problem (L, μ) is in Average-P; indeed it is linear on the μ-average according to the definition in [4], since

$$\sum_{n \geq 1} \frac{1}{2^n} \frac{2^n}{n^2} < \infty.$$

However, according to our definition, by Theorem 4.4 the problem (L, μ) is not in time $2^n/n^3$ on the μ-average. Thus, the two definitions differ.

We believe that distributions, such as this μ, that fail to satisfy Condition W are pathological. Such distributions put too much weight on short strings, so that the problem we are really dealing with becomes essentially a finitary problem, not one with an asymptotic behavior. However, if we must consider such distributions in the context of average-case analysis, we still stand by our guiding principle that a language that requires more than polynomial time almost everywhere is not polynomial time on the average for any distribution.

Earlier we informally wrote of "default" distributions such as $u_n = 1/n^2$ or $1/n^3$. (More formally $u_n = 1/(\zeta(2)n^2)$ or $u_n = 1/(\zeta(3)n^3)$, where $\zeta(s) = \sum_{n=1}^{\infty} 1/n^s$ is the Riemann zeta function.) Indeed, we had in mind distributions that satisfy Condition W. It is not merely a matter of convenience that these are the common default distributions. As we show by the example here, Levin's notion of average polynomial time yields counterintuitive results when applied to pathological distributions; i.e., mere convergence of $\sum_{n=1}^{\infty} u_n < \infty$ is not sufficient.

4.3.1. Worst-case average case. In a discussion of a preliminary draft of this paper, Rackoff [19] suggested the following more stringent requirement as a possible definition for a distributional problem (L, μ) to be T on the μ-average: There exists a Turing machine M that accepts L such that, for all n, the running time T_M satisfies

$$\sum_{|x|=n} \mu'(x) \cdot \frac{T^{-1}(T_M(x))}{|x|} \leq u_n. \tag{4.7}$$

Clearly, if T_M satisfies (4.7) it also satisfies our definition in (4.3).

For the notion of polynomial time on average, Gurevich [8] has shown that the definition given by (4.7) is equivalent to Levin's, and therefore to ours, for distributions that satisfy the additional condition (call it condition W*) that there exists $s > 0$ such that $u_n = \Omega\left(\frac{1}{n^s}\right)$.

It is not hard to show that the class of distributional problems that are polynomial on the μ-average is not the same as the class that (4.7) defines for distributions that do not satisfy condition W*.

The notion expressed in (4.7) is an interesting one. For example, since it refines ours, one can prove a fine hierarchy theorem. Also, it reflects the intuitive notion of average-case problems when it is important to bound the average hardness of a problem for every length n. This is especially relevant in areas such as cryptography and number theoretic problems. In a sense it is a hybrid requirement, best described as the worst-case bound (over all lengths n) of the average-case complexity (within each length n).

4.4. Final comments. Levin [14] provided central notions toward the development of a theory of *average polynomial time*. One is the class Average-P, which provides a classification of *easy* problems on average; the other is the class Dist-NP, which, together with complete distributional problems under appropriate reductions, provides a *hardness* notion. In this paper we have focused on extending the classification to arbitrary time-bounds. We refer the reader to papers by Belanger, Pavan, and Wang [2] and Pavan and Selman [18] for research on issues concerning reductions and complete problems as they relate to the new definitions we have given here.

5. Random-access machines. In order to illustrate our technique one more time, we complete this paper by giving a hierarchy theorem for random-access machines.

THEOREM 5.1. *Let $T, T' : \mathrm{N} \to \mathrm{N}$ be logarithmico-exponential functions and assume that T and T' are fully time constructible. Assume that $T'(n) = o(T(n))$. Then there is a language L such that for every polynomial time computable distribution μ there is a random-access machine M that decides L, whose running time T_M is $O(T)$ on the μ-average, but for every random-access machine M' that decides L, the running time of M' is* not *T' on the μ-average.*

Proof. It is known [6] that there is a language L that is decided by a random-access machine in time $O(T(n))$ such that the running time of every random-access machine M' that decides L exceeds $T'(n)$ almost everywhere. (This result is an almost-everywhere version of a hierarchy theorem of Cook and Reckow for random-access machines [5].) Let μ be any polynomial time computable distribution that satisfies Condition W. It is immediately apparent that the running time of M is $O(T(n))$ on the μ-average. However, it is also immediately apparent, as in the proof of Theorem 4.4, that the running time of M' is *not* T' on the μ-average. □

Acknowledgments. We thank Steve Cook, Leonid Levin, Charlie Rackoff, and Jie Wang for helpful comments and discussions on a preliminary draft of this paper. We also thank the two anonymous referees for their comments and criticisms, which have improved the paper.

REFERENCES

[1] J. Balcázar and U. Schöning, *Bi-immune sets for complexity classes*, Math. Systems Theory, 18 (1985), pp. 1–10.

[2] J. Belanger, A. Pavan, and J. Wang, *Reductions do not preserve fast convergence rates in average time*, Algorithmica, 23 (1999), pp. 363–378.

[3] J. Belanger and J. Wang, *Rankable distributions do not provide harder instances than uniform distributions*, in Proceedings of the First Annual International Computing and Combinatorics Conference, Lecture Notes in Comput. Sci. 959, Springer-Verlag, Berlin, 1995, pp. 410–419.

[4] S. Ben-David, B. Chor, O. Goldreich, and M. Luby, *On the theory of average case complexity*, J. Comput. System Sci., 44 (1992), pp. 193–219.

[5] S. Cook and R. Reckow, *Time bounded random access machines*, J. Comput. System Sci., 7 (1973), pp. 353–375.

[6] J. Geske, D. Huynh, and J. Seiferas, *A note on almost-everywhere-complex sets and separating deterministic-time-complexity classes*, Inform. Comput., 92 (1991), pp. 97–104.

[7] J. Geske, D. Huynh, and A. Selman, *A hierarchy theorem for almost everywhere complex sets with application to polynomial complexity degrees*, in Proceedings of the Fourth Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 247, Springer-Verlag, Berlin, 1987, pp. 125–135.

[8] Y. Gurevich, *Average case completeness*, J. Comput. System Sci., 42 (1991), pp. 346–398.

[9] G. Hardy, *Properties of logarithmico-exponential functions*, Proc. London Math. Soc., 10 (1911), pp. 54–90.

[10] G. Hardy, *Orders of Infinity. The Infinitärcalcül of Paul du Bois-Reymond,* Reprint of the 1910 Edition, Cambridge Tracts in Math. and Math. Phys. 12, Hafner Publishing Co., New York, 1971.

[11] J. Hartmanis and R. Stearns, *On the computational complexity of algorithms*, Trans. Amer. Math. Soc., 117 (1965), pp. 285–306.

[12] S. Homer, *Structural properties of complete problems for exponential time*, in Complexity Theory Retrospective II, L. Hemaspaandra and A. Selman, eds., Springer-Verlag, New York, 1997, pp. 135–153.

[13] K. Ko, *On self-reducibility and weak P-selectivity*, J. Comput. System Sci., 26 (1983), pp. 209–211.

[14] L. Levin, *Average case complete problems*, SIAM J. Comput., 15 (1986), pp. 285–286.

[15] M. Li and P. Vitányi, *Average case complexity under the universal distribution equals worst-case complexity*, Inform. Process. Lett., 42 (1992), pp. 145–149.

[16] J. Lutz and E. Mayordomo, *Cook versus Karp-Levin: Separating completeness notions if NP is not small*, in Proceedings of the Eleventh Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 755, Springer-Verlag, Berlin, 1994, pp. 415–426.

[17] E. Mayordomo, *Almost every set in exponential time is P-bi-immune*, Theoret. Comput. Sci., 136 (1994), pp. 487–506.

[18] A. Pavan and A. Selman, *Complete distributional problems, hard languages, and resource-bounded measure*, Theoret. Comput. Sci., to appear.

[19] C. Rackoff, *personal communication*.

[20] R. Reischuk and C. Schindelhauer, *Precise average case complexity*, in Proceedings of the Tenth Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 665, Springer-Verlag, Berlin, 1993, pp. 400–409.

[21] R. Schuler and T. Yamakami, *Sets computable in polynomial time on the average*, in Proceedings of the First Annual International Computing and Combinatorics Conference, Lecture Notes in Comput. Sci. 959, Springer-Verlag, Berlin, 1995, pp. 650–661.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1326–1346

BUCKETS, HEAPS, LISTS, AND MONOTONE PRIORITY QUEUES*

BORIS V. CHERKASSKY†, ANDREW V. GOLDBERG‡, AND CRAIG SILVERSTEIN§

Abstract. We introduce the heap-on-top (hot) priority queue data structure that combines the multilevel bucket data structure of Denardo and Fox with a heap. Our data structure has superior operation bounds than either structure taken alone. We use the new data structure to obtain an improved bound for Dijkstra's shortest path algorithm. We also discuss a practical implementation of hot queues. Our experimental results in the context of Dijkstra's algorithm show that this implementation of hot queues performs very well and is more robust than implementations based only on heap or multilevel bucket data structures.

Key words. data structures, priority queues, shortest paths

AMS subject classifications. 05C85, 06Q25, 90C35

PII. S0097539796313490

1. Introduction.

A priority queue is a data structure that maintains a set of elements and supports operations `insert`, `decrease-key`, and `extract-min`. Priority queues are fundamental data structures with many applications. Typical applications include graph algorithms (e.g., [14]) and event simulation (e.g., [5]).

An important subclass of priority queues, used in applications such as event simulation and in Dijkstra's shortest path algorithm [13], is the class of *monotone* priority queues. In monotone priority queues, the extracted keys form a monotone, nondecreasing sequence. In this paper we develop a new data structure for monotone priority queues.

Consider an event simulation application, in which we use a monotone priority queue to maintain a set of items. Item keys are times at which the items will be processed. At each step, we extract an item with the smallest key t and process it. This precipitates several events $e_1, \ldots, e_i$, of nonnegative event durations $t_1, \ldots, t_i$. Each event j causes an item with key $t + t_j$ to be inserted into the queue. We denote the maximum event duration by C.

We refer to priority queues whose operations are more efficient when the number of elements on the queue is small as *s-heaps*.[1] For example, in a binary heap containing n elements, all priority queue operations take $O(\log n)$ time, so the binary heap is an s-heap. (Operation bounds may depend on parameters other than the number of elements.) The fastest implementations of s-heaps are described in [4, 14, 22].

*Received by the editors December 1, 1996; accepted for publication (in revised form) March 19, 1998; published electronically March 30, 1999. A previous version of this paper appeared in *Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, 1997, pp. 83–92.
http://www.siam.org/journals/sicomp/28-4/31349.html

†Central Econ. and Math. Inst., Krasikova St. 32, 117418, Moscow, Russia (cher@cemi.msk.su). This work was done while the author was visiting NEC Research Institute.

‡InterTrust Technologies Corp., 460 Oakmead Parkway, Sunnyvale, CA 94086 (goldberg@intertrust.com, http://www.intertrust.com/star/goldberg). This work was done while the author was at NEC Research Institute, Princeton, NJ.

§Computer Science Department, Stanford University, Stanford, CA 94305 (csilvers@cs.stanford.edu). This author was supported by the Department of Defense and an ARCS fellowship, with partial support from NSF Award CCR–9357849 and matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

[1]Here "s" stands for "size."

Alternative implementations of priority queues use buckets (e.g., [2, 8, 11, 12]). Operation times for bucket-based implementations depend on the maximum event duration C and are not very sensitive to the number of elements. See [3] for a related data structure.

s-heaps are particularly efficient when the number of elements on the s-heap is small. Bucket-based priority queues are particularly efficient when the maximum event duration C is small. Furthermore, some of the work done in bucket-based implementations can be amortized over elements in the buckets, so bucket-based priority queues actually have better bounds if the number of elements is large. In this sense, s-heaps and buckets complement each other.

We introduce *heap-on-top priority queues (hot queues)*, which combine the multilevel bucket data structure of Denardo and Fox [11] and a monotone s-heap.[2] The resulting implementation takes advantage of the best performance features of both data structures. In order to describe hot queues, we give an alternative and more insightful description of the multilevel bucket data structure. Independently, a similar description has been given in [18].

We illustrate efficiency of hot queues by implied bounds for Dijkstra's shortest path algorithm; detailed operation bounds appear in section 4. In the following discussion, n is the number of vertices, m the number of arcs, C the largest arc length, and ϵ any positive constant. We assume integral arc lengths.

The best time bounds in the standard RAM model of computation appear in [2, 3, 14]. Out of these algorithms, the algorithm of Ahuja et al., based on the radix heap data structure, is closest to our algorithm. Their algorithm runs in $O(m + n(\log C)^{\frac{1}{2}})$ time. Fredman and Willard [15] showed that better bounds are possible in a stronger RAM model; the fastest currently known algorithm [22] in this model runs in $O(m \log\log n)$ time.

Efficiency of a hot queue depends on that of the heap used to implement it. Using Fibonacci heaps [14], we match the radix heap bounds in the standard model. Our data structure, however, is simpler. In a stronger model, we use the s-heap of Thorup [22] to obtain better monotone priority queue bounds. In particular, we get an $O(m + n(\log C)^{\frac{1}{3}+\epsilon})$ expected time implementation of Dijkstra's shortest path algorithm.

Hot queues work well both in theory and in practice. This is often not the case for low-complexity data structures, including priority queues. Asymptotic bound improvements often come at the expense of constant factors, which come to dominate the running time for all reasonable problem sizes. A preliminary version of the hot queue data structure [6] did not perform well in practice. Based on experimental feedback, we modified the data structure to be more practical. As an added bonus, hot queues became simpler. We also developed implementation techniques that make hot queues efficient in practice.

We compare the implementation of hot queues to implementations of multilevel buckets and d-heaps [23] in the context of Dijkstra's shortest paths algorithm. Our experimental results show that hot queues perform best overall and are more robust than either of the other two data structures. This is especially significant because a multilevel bucket implementation of Dijkstra's algorithm compared favorably with other implementations of the algorithm in a previous study [8] and was shown to

[2] Actually, our data structure can use any heap. However, hot queues are designed so that the number of elements on the heap is small, so s-heaps lead to the best bounds.

be very robust [16]. For many problem classes, the hot queue implementation of Dijkstra's algorithm is the best both in theory and in practice.

This paper is organized as follows. Section 2 introduces basic definitions. Our description of the multilevel bucket data structure appears in section 3. Section 4 describes hot queues and their application to Dijkstra's algorithm. Details of our implementation, including heuristics we used, are described in section 5. Section 6 describes our experimental setup, including the problem families we use. Section 7 contains experimental results. Section 8 contains concluding remarks.

2. Preliminaries. A *priority queue* is a data structure that maintains a set of elements and supports operations `insert`, `decrease-key`, and `extract-min`. We assume that elements have *keys* used to compare the elements, and we denote the key of an element u by $\rho(u)$. Unless mentioned otherwise, we assume that the keys are integral. By the value of an element we mean the key of the element. The `insert` operation adds a new element to the queue. The `decrease-key` operation assigns a smaller value to the key of an element already on the queue. The `extract-min` operation removes a minimum element from the queue and returns the element. We denote the number of `insert` operations in a sequence of priority queue operations by N.

To gain intuition about the following definition, think of event simulation applications where keys correspond to event durations. An *event* is an `insert` or a `decrease-key` operation on the queue. Given an event, let v be the element that is inserted into the queue or has its key decreased. Let u be the most recent element extracted from the queue. The *event duration* is $\rho(v) - \rho(u)$. In Dijkstra's shortest path algorithm, event durations correspond to arc lengths. We denote the maximum event duration by C. A *monotone* priority queue is a priority queue for monotone applications. To make these definitions valid for the first insertion, we assume that during initialization, a special element is inserted into the queue and deleted immediately afterward. Without loss of generality, we assume that the value of this element is zero. (If it is not, we can subtract this value from all element values.)

In this paper, by *s-heap* we mean a priority queue that is sensitive to the number of elements in the queue and is more efficient if the number of elements on the queue is small.

We call a sequence of operations on a priority queue *balanced* if the sequence starts and ends with an empty queue. In particular, implementations of Dijkstra's shortest path algorithm produce balanced operation sequences.

All logarithms in this paper are base two.

Hot queues work in the *word RAM* model of computation (see, e.g., [1]): they need array addressing and the following unit-time word operations: addition, subtraction, comparison, and arbitrary shifts (which are equivalent to multiplication and division by powers of two). However, hot queues use s-heaps, and some heaps assume a *strong RAM* model where certain other word operations take unit time. The most common operations are AC0 operations and multiplication; different variants allow different operations. See [15, 22].

The multilevel bucket data structure, one of the main building blocks of hot queues, works in the word RAM model. In section 3, we describe a strong RAM variant of this data structure that uses word operations including bitwise logical operations and the operation of finding the index of the most significant bit in which two words differ. The latter operation is in AC0; see [10] for a discussion of a closely related operation. The use of this more powerful model does not improve the amor-

tized operation bounds, but it improves some worst case bounds and simplifies the description.

In the context of Dijkstra's algorithm, we assume that we are given a directed graph with n vertices, m arcs, and integral arc lengths in the range $[0, \ldots, C]$, where C fits in one machine word.

3. Multilevel buckets. In this section we describe the k-level bucket data structure of Denardo and Fox [11]. By using a strong RAM model, we give a simpler description of this data structure than that given in [11]. We treat the element keys as base-Δ numbers for a certain parameter Δ. Consider a bucket structure B that contains k levels of buckets, where k is a positive integer. Except for the top level, a level contains an array of Δ buckets. The top level contains infinitely many buckets. Each top level bucket corresponds to an interval $[i\Delta^k, (i+1)\Delta^k - 1]$. We chose Δ so that at most Δ consecutive buckets at the top level can be nonempty; we need to maintain only these buckets.[3]

We denote bucket j at level i by $B(i, j)$; i ranges from 1 (bottom level) to k (top), and j ranges from 0 to $\Delta - 1$. A bucket contains a set of elements in a way that allows constant-time insertion and deletion, e.g., in a doubly linked list.

Given k, we choose Δ as small as possible subject to two constraints. First, each top level bucket must span a key range of at least $(C+1)/\Delta$. Then, by the definition of C, keys of elements in B belong to at most Δ consecutive top level buckets. Second, Δ must be a power of two so that we can manipulate base-Δ numbers efficiently using RAM operations on words of bits. With these constraints in mind, we set Δ to the smallest power of two greater than or equal to $(C+1)^{1/k}$.

We maintain μ, the key of the latest element extracted from the queue. Consider the base-Δ representation of the keys and an element u in B. Let μ_{i-j} denote the ith through jth least significant digits of μ. $\mu_{i-\infty}$ denotes the ith and higher least significant digits, while μ_i denotes the ith least significant digit alone. Similarly, u_i denotes the ith least significant digit of $\rho(u)$, and likewise for the other definitions. By the definitions of C and Δ, μ and the k least significant digits of the base-Δ representation of $\rho(u)$ uniquely determine $\rho(u)$. Thus, $\rho(u) = \mu + (u_{0-k} - \mu_{0-k})$ if $u_{0-k} > \mu_{0-k}$ and $\mu + \Delta^k + (u_{0-k} - \mu_{0-k})$ otherwise.

Let i be the index of the most significant digit in which $\rho(u)$ and μ differ, or 1 if $\rho(u) = \mu$. (The least significant digit index is 1.) Given μ and u with $\rho(u) \geq \mu$, we denote the *position of u with respect to μ* by (i, u_i). If u is inserted into B, it is inserted into $B(i, u_i)$. For each element in B, we store its position. If an element u is in $B(i, j)$, then only the i least significant digits of $\rho(u)$ differ from the corresponding digits of μ. Furthermore, $u_i = j$.

The fact that keys of all elements on the queue are at least μ implies the following lemma.

LEMMA 3.1. *For every level i, buckets $B(i, j)$ for $0 \leq j < \mu_i$ are empty.*

At each level i, we maintain the number of elements at this level. We also maintain the total number of elements in B.

The `extract-min` operation can change the value of μ. As a side effect, positions of some elements in B may change. Suppose that a minimum element is deleted and the value of μ changes from μ' to μ''. By definition, keys of the elements on the queue after the deletion are at least μ''. Let i be the position of the most significant digit in which μ' and μ'' differ. If $i = 1$ (μ' and μ'' differ only in the last digit), then for

[3] The simplest way to implement the top level is to "wrap around" modulo Δ.

any element in B after the deletion its position is the same as before the deletion. If $i > 1$, then the elements in bucket $B(i, \mu_i'')$ with respect to μ' are exactly those whose position is different with respect to μ''. These elements have a longer prefix in common with μ'' than with μ', and therefore they belong to a lower level with respect to μ''.

The *bucket expansion* procedure moves these elements to their new positions. The procedure removes the elements from $B(i, \mu_i'')$ and puts them into their positions with respect to μ''. The two key properties of bucket expansions are as follows:

- After the expansion of $B(i, \mu_i'')$, all elements of B are in correct position with respect to μ''.
- Every element of B moved by the expansion is moved to a lower level.

Now we are ready to describe the multilevel bucket implementation of the priority queue operations.

`insert:` To insert an element u, compute its position (i, j) and insert u into $B(i, j)$.

`decrease-key:` Decrease the key of an element u in position (i, j) as follows. Remove u from $B(i, j)$. Set $\rho(u)$ to the new value and insert u as described above.

`extract-min:` (We need to find and delete the minimum element, update μ, and move elements affected by the change of μ.)

Find the lowest nonempty level i. Find j, the first nonempty bucket at level i. If $i = 1$, delete an element from $B(i, j)$, set $\mu = \rho(u)$, and return u. (In this case all element positions remain the same.)

If $i > 1$, examine all elements of $B(i, j)$ and delete a minimum element u from $B(i, j)$. Set $\mu = \rho(u)$ and expand $B(i, j)$. Return u.

Next we deal with efficiency issues.

LEMMA 3.2. *Given μ and u, we can compute the position of u with respect to μ in constant time.*

Proof. By definition, $\Delta = 2^\delta$ for some integer δ, so each digit in the base-Δ representation of $\rho(u)$ corresponds to δ bits in the binary representation. It is straightforward to see that if we use appropriate masks and the fact that the index of the first bit in which two words differ can be computed in constant time [10], we can compute the position in constant time. □

Iterating through the levels, we can find the lowest nonempty level in $O(k)$ time. Using binary search, we can find the level in $O(\log k)$ time. We can do even better using the power of the strong RAM model.

LEMMA 3.3. *If $k \le \log C$, then the lowest nonempty level of B can be found in $O(1)$ time.*

Proof. Define D to be a k-bit number with $D_i = 1$ if and only if level i is nonempty. If $k \le \log C$, D fits into one RAM word, and we can set a bit of D and find the index of the first nonzero bit of D in constant time. □

As we will see, the best bounds are achieved for $k \le \log C$.

A simple way of finding the first nonempty bucket at level i is to go through the buckets. This takes $O(\Delta)$ time.

LEMMA 3.4. *We can find the first nonempty bucket at a level in $O(\Delta)$ time.*

Remark. One can do better [11]. Divide buckets at every level into groups of size $\lceil \log C \rceil$, each group containing consecutive buckets. For each group, maintain a $\lceil \log C \rceil$-bit number with bit j equal to 1 if and only if the jth bucket in the group is not empty. We can find the first nonempty group in $O(\Delta / \log C)$ time and the first nonempty bucket in the group in $O(1)$ time. This construction gives a $\log C$ factor improvement for the bound of Lemma 3.4. By iterating this construction p times, we

get an $O(p + (\Delta/\log^p C))$ bound. Note that we can use word-size groups instead of $\lceil \log C \rceil$-size groups.

Although the above observation improves the multilevel bucket operation time bounds for small values of k, the bounds for the optimal value of k do not improve. To simplify the presentation, we use Lemma 3.4, rather than its improved version, in the rest of the paper.

THEOREM 3.5. *Amortized bounds for the multilevel bucket implementation of priority queue operations are as follows:* $O(k)$ *for* `insert`, $O(1)$ *for* `decrease-key`, *and* $O(C^{1/k})$ *for* `extract-min`.

Proof. The `insert` operation takes $O(1)$ worst case time. We assign it an amortized cost of k because we charge moves of elements to a lower level to the insertions of the elements.

The `decrease-key` operation takes $O(1)$ worst case time, and we assign it an amortized cost of $O(1)$.

The worst case time of the `extract-min` operation is $O(\log k + \Delta)$ plus the cost of bucket expansions. The cost of a bucket expansion is proportional to the number of elements in the bucket. This cost can be charged to the corresponding `insert` operations, because, except for the minimum element, each element examined during a bucket expansion is moved to a lower level. An element can move down at most $k - 1$ times. We charge the $\log k$ factor to the insertions as well. This completes the proof since $\Delta = O(C^{1/k})$. □

Note that in any sequence of operations the number of `insert` operations is at least the number of `extract-min` operations. In a balanced sequence, the two numbers are equal, and we can modify the above proof to obtain the following result.

THEOREM 3.6. *For a balanced sequence, amortized bounds for the multilevel bucket implementation of priority queue operations are as follows:* $O(1)$ *for* `insert`, $O(1)$ *for* `decrease-key`, $O(k + C^{1/k})$ *for* `extract-min`.

Remark. We can easily obtain an $O(1)$ implementation of the `delete` operation for multilevel buckets. Given a pointer to an element, we delete it by removing the element from the list of elements in its bucket.

For $k = 1$, the `extract-min` bound is $O(C)$. For $k = 2$, the bound is $O(\sqrt{C})$. The best bound of $O(\log C/\log\log C)$ is obtained for $k = \lceil \log C/2\log\log C \rceil$.

Remark. The original multilevel bucket data structure [11] works in the word RAM model of computation. This implementation maintains ranges for each bucket level and finds element positions by first finding the appropriate level (using sequential or binary search) and then computing the element's bucket index (using shifts). For this implementation, some operations take longer; for example, `insert` is not constant time. However, the amortized bounds of Theorem 3.5 are valid.

Remark. The k-level bucket data structure uses $\Theta(kC^{1/k})$ space for the buckets.

A major bottleneck of the multilevel bucket implementation is bucket scans. A natural idea is to maintain nonempty bucket indices in a heap. A variant of this idea leads to radix heaps, which maintain a heap containing all elements, with element positions serving as keys. The number of distinct keys is $O(kC^{1/k})$. Radix heaps use a modification of Fibonacci heaps with amortized $O(1)$ `insert` and `decrease-key` operations, and amortized $O(\log N')$ `extract-min` operation, where N' is the number of distinct keys on the heap. Setting $k = \sqrt{\log C}$ gives the following operation costs for the balanced case: $O(1)$ for `insert` and `decrease-key`, and $O(\sqrt{\log C})$ for `extract-min`.

We use a different idea to improve on multilevel buckets. We scan a bucket level

only if many elements "went through" this level and charge bucket scans to these elements.

4. Hot queues. A *hot queue* uses an s-heap H and a multilevel bucket structure B. Intuitively, the hot queue data structure works like the multilevel bucket data structure, except we do not expand a bucket containing fewer than t elements, where t is a parameter set to optimize performance. Elements of the bucket are copied into H and processed using the s-heap operations. If the number of elements in the bucket exceeds t, the bucket is expanded. In the analysis, we charge scans of buckets at the lower levels to the elements in the bucket during the expansion into these levels.

A k-level hot queue uses the k-level bucket structure. For technical reasons, we must add an additional *special* level $k+1$, which is needed to account for scanning of buckets at level k. Only two buckets at the top level can be nonempty at any time, μ_{k+1} and $\mu_{k+1}+1$. Note that if the queue is nonempty, then at least one of the two buckets is nonempty. Thus bucket scans at the special level add a constant amount to the work of processing an element found. We use wrap around at level $k+1$ instead of k.

An *active* bucket is the bucket whose elements are in H. At most one bucket is active at any time, and H is empty if and only if there is no active bucket. We denote the active bucket by $B(a,b)$. We make a bucket active by making H into a heap containing the bucket elements and inactive by resetting H to an empty heap. (Elements of the active bucket are both in the bucket and in H.)

To describe the details of hot queues, we need the following definitions. We denote the number of elements in $B(i,j)$ by $c(i,j)$. Given μ, $i : 1 \le i \le k+1$, and $j : 0 \le j < \Delta$, we say that an element u is *in the range of* $B(i,j)$ if $u_{(i+1)-\infty} = \mu_{(i+1)-\infty}$ and $u_i = j$. Using word operations, we can check if an element is in the range of a bucket in constant time.

We maintain the invariant that μ is in the range of $B(a,b)$ if there is an active bucket. The detailed description of the queue operations is as follows.

`insert:` If H is empty or if the element u being inserted is not in the range of the active bucket, we insert u into B as in the multilevel case. Otherwise u belongs to the active bucket $B(a,b)$. If $c(a,b) < t$, we insert u into H and $B(a,b)$. If $c(a,b) \ge t$, we make $B(a,b)$ inactive, add u to $B(a,b)$, and expand the bucket.

`decrease-key:` Decrease the key of an element u as follows. If u is in H, decrease the key of u in H. Otherwise, let (i,j) be the position of u in B. Remove u from $B(i,j)$. Set $\rho(u)$ to the new value and insert u as described above.

`extract-min:` If H is not empty, extract and return the minimum element of H. Otherwise, proceed as follows. Find the lowest nonempty level i. Find the first nonempty bucket at level i by examining buckets starting from $B(i,\mu_i)$. If $i = 1$, delete an element from $B(i,j)$, set $\mu = \rho(u)$, and return u. If $i > 1$, examine all elements of $B(i,j)$ and delete a minimum element u from $B(i,j)$. Set $\mu = \rho(u)$. If $c(i,j) > t$, expand $B(i,j)$. Otherwise, make $B(i,j)$ active. Return u.

Correctness of the hot queue operations follows from the correctness of the multilevel bucket operations, Lemma 3.1, and the observation that if u is in H and v is in B but not in H, then $\rho(u) < \rho(v)$.

Note that at any time, H contains a (possibly empty) subset of the smallest elements of the hot queue. Consider a time interval when H is nonempty. It is easy to see that during each time interval while H is nonempty, the sequence of operations

TABLE 4.1
The bounds for hot queues using Fibonacci heaps. The best bounds are obtained with $k = \log^{\frac{1}{2}} C$ *and* $t = 2^{\log^{\frac{1}{2}} C}$. *Radix heaps achieve the same bounds but are more complicated.*

Fibonacci heaps	Heap bounds	Hot queue bounds	Hot queue, best k and t
`insert`	$O(1)$	$O(k + \frac{kC^{1/k}}{t})$	$O(\log^{\frac{1}{2}} C)$
`decrease-key`	$O(1)$	$O(1)$	$O(1)$
`extract-min`	$O(\log N)$	$O(\log t)$	$O(\log^{\frac{1}{2}} C)$

TABLE 4.2
The bounds for hot queues using Thorup's heaps. Here ϵ *is any constant. The best bounds are obtained with* $k = \log^{\frac{1}{3}} C$ *and* $t = 2^{\log^{\frac{2}{3}} C}$.

Thorup's heaps	Heap bounds	Hot queue bounds	Hot queue, best k and t
`insert`	$O(1)$	$O(k + \frac{kC^{1/k}}{t})$	$O(\log^{\frac{1}{3}} C)$
`decrease-key`	$O(1)$	$O(1)$	$O(1)$
`extract-min`	$O(\log^{\frac{1}{2}+\epsilon} N)$	$O(\log^{\frac{1}{2}+\epsilon} t)$	$O(\log^{\frac{1}{3}+\epsilon} C)$

on H is monotone. Thus we can use a monotone s-heap H.

LEMMA 4.1. *The cost of finding the first nonempty bucket at a level, amortized over the* `insert` *operations, is* $O(k\Delta/t)$.

Proof. It is enough to bound the number of empty buckets scanned during the search. We scan an empty bucket at level i at most once during the period of time that $\mu_{i-\infty}$ remains unchanged. This can happen only if a higher-level bucket has been expanded during the period that $\mu_{i-\infty}$ does not change. We charge bucket scans to insertions into the queue of the elements contained in the expanded bucket. For each level scan, we charge Δ to a group of over t elements, and each element's share is less than Δ/t. Each time we charge an element it moves down at least one level, so the total charge for an element is less then $k\Delta/t$. □

THEOREM 4.2. *Let* $I(N)$, $D(N)$, *and* $X(N)$ *be the time bounds for monotone heap* `insert`, `decrease-key`, *and* `extract-min` *operations. Then amortized times for the hot queue operations are as follows:* $O(k+I(t)+\frac{kC^{1/k}}{t})$ *for* `insert`, $O(D(t)+I(t))$ *for* `decrease-key`, *and* $O(X(t))$ *for* `extract-min`.

Proof. The result follows from Lemma 4.1, Theorem 3.5, and the fact that the number of elements in H never exceeds t. □

Remark. All bounds are valid only when $t \le N$. For $t > N$, one should use an s-heap instead of a hot queue.

The bounds for Fibonacci heaps are given in Table 4.1. The bounds for Thorup's heaps are in Table 4.2. Similarly to Theorem 3.6, we can get bounds for a balanced sequence of operations.

THEOREM 4.3. *Let* $I(N)$, $D(N)$, *and* $X(N)$ *be the time bounds for heap* `insert`, `decrease-key`, *and* `extract-min` *operations, and consider a balanced sequence of the hot queue operations. The amortized bounds for the operations are as follows:* $O(I(t))$ *for* `insert`, $O(D(t) + I(t))$ *for* `decrease-key`, *and* $O(k + X(t) + \frac{kC^{1/k}}{t})$ *for* `extract-min`.

Using Fibonacci heaps, we get $O(1)$, $O(1)$, and $O(k + \log t + \frac{kC^{1/k}}{t})$ amortized bounds for the queue operations. Consider `extract-min`, the only operation with a nonconstant bound. Setting $k = 1$ and $t = \frac{C}{\log C}$, we get an $O(\log C)$ bound. Setting $k = 2$ and $t = \frac{\sqrt{C}}{\log C}$, we get an $O(\log C)$ bound. Setting $k = \log^{\frac{1}{2}} C$ and $t = 2^{\log^{\frac{1}{2}} C}$,

we get an $O(\log^{\frac{1}{2}} C)$ bound.

Remark. Consider the 1- and 2-level implementations. Although the time bounds are the same, the 2-level implementation has two advantages: it uses less space, and its time bounds remain valid for a wider range of values of C.

Using Thorup's heaps and setting $k = \log^{\frac{1}{3}} C$ and $t = 2^{\log^{\frac{2}{3}} C}$, we get $O(1)$, $O(1)$, and $O(\log^{\frac{1}{3}+\epsilon} C)$ expected amortized time bounds for the queue operations `insert`, `decrease-key`, and `extract-min`, respectively.

The above time bounds allow us to get an improved bound on Dijkstra's shortest path algorithm. The running time of Dijkstra's algorithm is dominated by a balanced sequence of priority queue operations that includes $O(n)$ `insert` and `extract-min` operations and $O(m)$ `decrease-key` operations (see, e.g., [21]). The maximum event duration for this sequence of operations is C. The hot queue bounds immediately imply the following result.

THEOREM 4.4. *On a network with n vertices, m arcs, and integral lengths in the range $[0, C]$, the shortest path problem can be solved in $O(m + n(\log C)^{\frac{1}{3}+\epsilon})$ expected time.*

This improves the deterministic bound of $O(m + n \log^{\frac{1}{2}} C)$ achieved using radix heaps [2].

Remark. Like multilevel buckets, hot queues do not need a strong RAM model unless the s-heap used needs it.

Space bounds. Suppose $\overline{n}$ is the maximum number of elements in a k-level hot queue and assume that the underlying s-heap requires constant space per element. Then the additional space needed for the hot queue is $O(kC^{1/k}+\overline{n})$. Here the first term reflects the space used by the buckets and the second term reflects the space needed to maintain element lists and the $O(t)$ space used by the s-heap. For comparison, the multilevel buckets require the same space minus the s-heap space.

The number of buckets needed for $C = 2^{60}$ by a 4-level hot queue is 2^{15}. The space needed for these buckets is comparable with cache sizes of current computers.

From a theoretical point of view, it is interesting to obtain a per-element space bound. We get such a bound by "lazy allocation": only the two special-level buckets are allocated unless the other buckets are needed, i.e., unless $\overline{n} > t$. Then the per-element space bound is constant unless the buckets are allocated, in which case the bound is $O(\frac{kC^{1/k}}{\overline{n}}) = O(\frac{kC^{1/k}}{t})$. Then, since k and $\frac{C^{1/k}}{t}$ are small, the per-element space is small as well. For example, for the $\sqrt{\log C}$-level hot queue using Fibonacci heaps, the per-element bound is $O(\sqrt{\log C})$. The corresponding space bounds for the faster hot queues described above are even better.

5. Implementation details. Our previous papers [8, 16] describe implementations of multilevel buckets. Our implementation of hot queues augments the multilevel bucket implementation of [16]. See [16] for details of the multilevel bucket implementation.

Consider a k-level hot queue. As in the multilevel bucket implementation, we set Δ to the smallest power of two greater than or equal to $C^{1/k}$. Based on the analysis of section 4 and experimental results, we set t, the maximum size of an active bucket, to $\lceil C^{1/k}/(4 \log C) \rceil$.

The number of elements in an active bucket is often small. For example, for $k = 3$ and $C = 100,000,000$, we have $t = 5$. We take advantage of this fact by maintaining elements of an active bucket in a sorted list instead of an s-heap until operations on the list become expensive. At this point we switch to an s-heap. We use a 4-heap, which worked best in our tests.

To implement priority queue operations using a sorted list, we use a doubly linked list sorted in nondecreasing order. Our implementation is tuned for the shortest path application. In this application, the number of `decrease-key` operations on the elements of the active bucket tends to be very small and elements inserted into the list or moved by the `decrease-key` operation tend to be close to the beginning of the list. A different implementation may be better for a different application.

The `insert` operation searches for the element's position in the list and puts the element at that position. One can start the search in different places. Our implementation starts the search at the beginning of the list. Starting at the end of the list or at the point of the last insertion, may work better in some applications.

The `extract-min` operation removes the first element of the list.

The `decrease-key` operation removes the element from the list, finds its new position, and puts the element in that position. Our implementation starts the search from the beginning of the list. Starting at the previous position of the element, at the end of the list, or at the place of the last insertion may work better in some applications.

When a bucket becomes active, we put its elements in a list if the number of elements in the bucket is below T_1 and in an s-heap otherwise. (Our code uses $T_1 = 2,000$.) We switch from the list to the heap using the following rule, suggested by Satish Rao [20]: Switch if `insert` and `decrease-key` operations examine more than $T_2 = 5,000$ list elements from the time the current bucket became active. An alternative, which may work better in some applications but which performed worse in ours, is to switch when the number of elements in the list exceeds T_1. Once we started using a heap, we continued using it until the next bucket expansion.

6. Experimental setup. Our experiments were conducted on a computer with a 166 MHz Pentium Pro processor running Solaris x86 2.5.1. The machine has 96 MHz of main memory and all problem instances fit into main memory. Our code was written in C++ and compiled using `gcc 2.7.2.2` with the `-O6` compilation option.

We made an effort to make our code efficient. In particular, we set the bucket array sizes to be powers of two. This allows us to use word shift operations when computing bucket array indices.

We report experimental results for five types of directed graphs. Two of the graph types were chosen to exhibit the properties of the algorithm at two extremes: one where the paths from the start vertex to other vertices tend to be order $\Theta(n)$, and one in which the path lengths are order $\Theta(1)$. The third graph type is random graphs. The fourth type was constructed to have a lot of `decrease-key` operations in the active bucket. This is meant to test the robustness of our implementations when we violate the assumption (section 5) that there are few `decrease-key` operations. The fifth type of graph is meant to be easy or hard for a specific implementation with a specific number of bucket levels.

We tested each type of graph on seven implementations: d-heaps, with d=4; k-level buckets, with k ranging from 1 to 3; and k-level hot queues, with k ranging from 1 to 3. Each of these has parameters to tune, and the results we show are for the best parameter values we tested.

Most of the problem families we use are the same as in our previous paper [16]. The next two sections describe the problem families.

6.1. Graph types. Two types of graphs we explored were grids produced using the GRIDGEN generator [8]. These graphs can be characterized by a length x and width y. The graph is formed by constructing x layers, each of which is a path of

TABLE 6.1
The graph types used in our experiments; p is the number of buckets at each level.

Name	Type	Description	Salient feature
long grid	grid	16 vertices high $n/16$ vertices long	path lengths are $\Theta(n)$
wide grid	grid	$n/16$ vertices high 16 vertices long	path lengths are $\Theta(1)$
random	random	degree 4	path lengths are $\Theta(\log n)$
cycle	cycle	$d(i,j) = (i-j)^2$	results in many `decrease-key` operations
hard	two paths	$d(S, \text{path } 1) = 0$ $d(S, \text{path } 2) = p-1$	vertices occupy first and last buckets in bottom level bins

length y. We order the layers, as well as the vertices within each layer, and we connect each vertex to its corresponding vertex on adjacent layers. All the vertices on the first layer are connected to the source.

The first type of graph we used, the LONG GRID, has a constant width—16 vertices in our tests. We used graphs of different lengths, ranging from 512 to 32,768 vertices. The arcs had lengths chosen independently and uniformly at random in the range from 1 to C. C varied from 1 to 100,000,000.

The second type of graph we used was the WIDE GRID type. These graphs have length limited to 16 layers, while the width can vary from 512 to 32,768 vertices. C was the same as for LONG GRIDS.

The third type of graph includes random graphs with uniform arc length distribution. A random graph with n vertices has $4n$ arcs.

The fourth type of graph is the only type that is new compared with [16]. These are based on a cycle of n vertices, numbered 1 to n. In addition, each vertex is connected to $d-1$ distinct random vertices.

The length of an arc (i,j) is equal to $2k^{1.5}$, where k is the number of arcs on the cycle path from i to j. The fifth type of graph includes HARD graphs. These are parameterized by the number of vertices, the desired number of levels k, and a maximum arc length C. From C we compute p, the number of buckets in each level assuming the implementation has k levels. The graphs consist of two paths connected to the source. The vertices in each path are at distance p from each other. The distance from the source to path 1 is 0; vertices in this path will occupy the first bucket of bottom level bins. The distance from the source to path 2 is $p-1$, making these vertices occupy the last bucket in each bottom level bin. In addition, the source is connected to the last vertex on the first path by an arc of length 1, and to the last vertex of the second path by an arc of length C.

A summary of our graph types appears in Table 6.1.

6.2. Problem families. For each graph type, we examined how the relative performance of the implementations changed as we increased various parameters. Each type of modification constitutes a *problem family.* The families are summarized in Table 6.2. In general, each family is constructed by varying one parameter while holding the others constant. Different families can vary the same parameter, using different constant values. For instance, one problem family modifies x as $C = 16$, another modifies x as $C = 10,000$, and a third modifies x as $C = 100,000,000$.

7. Experimental results. The results of this section should be taken in proper context. The 4-heap implementation has the *worst case* operation bound of $O(\log_4 n)$ with a relatively small constant. For most shortest path problems, only a fraction of

TABLE 6.2

The problem families used in our experiments. C is the maximum arc length; x and y the length and width, respectively, of grid graphs; d is the degree of vertices in the cycle graph; and p is the number of buckets at each level.

Graph type	Graph family	Range of values	Other values
long grid	Modifying C	$C = 1$ to $1,000,000$	$x = 8192$
	Modifying x	$x = 512$ to $32,768$	$C = 16$
			$C = 10,000$
			$C = 100,000,000$
	Modifying C and x	$x = 512$ to $32,768$	$C = x$
wide grid	Modifying C	$C = 1$ to $1,000,000$	$y = 8192$
	Modifying y	$y = 512$ to $32,768$	$C = 16$
			$C = 10,000$
			$C = 100,000,000$
	Modifying C and y	$y = 512$ to $32,768$	$C = y$
random graph	Modifying C	$C = 1$ to $1,000,000$	$n = 131,072$
	Modifying n	$n = 8192$ to $524,288$	$C = 16$
			$C = 10,000$
			$C = 100,000,000$
	Modifying C and n	$n = 8192$ to $524,288$	$C = n$
cycle	Modifying n	$n = 300$ to $1,200$	$d = 20$
			$d = 200$
hard	Modifying C	$C = 100$ to $10,000,000$	$n = 131,072, p = 2$
			$n = 131,072, p = 3$

vertices are on the heap at any given time. The problems in our study are big enough to establish the relative performance of the codes but not big enough to see a drastic difference in performance.

A previous study [7] suggested that the multilevel bucket implementations compare well with other implementations of Dijkstra's algorithm. This motivated a further study of the multilevel buckets [16] and the discovery of hot queues. The problem families presented here are designed to test the multilevel bucket and hot queue codes at extremes, and not to test these codes against the 4-heap.

Another study [16] showed that the multilevel buckets are efficient and robust except on a limited class of problems which cause a large number of empty bucket scans. One cannot expect hot queues to outperform the buckets when the number of such scans is moderate. The data presented below show that the additional complexity of hot queues does not significantly increase the constant factors in our implementations. Hot queues are competitive with multilevel buckets when the latter scan a moderate number of empty buckets and outperform the buckets when the number is large. The improved robustness of hot queues comes at essentially no cost to typical performance.

In the following discussion of our computational results, we present data for some, but not all, problem families from Table 6.2. Qualitative results for the other problem families can be interpolated from the data we present. An earlier technical report [9] contains data for all listed problem families. Our codes are publicly available, and interested readers can run their own experiments.

We tabulate our experimental data. In all the tables, k denotes the implementation: "h" for heap, "bi" for buckets with i levels, and "hi" for hot queue with i levels. In addition to reporting running time, we report counts for operations that give insight into algorithm performance. For the heap implementation, we count the total number of `insert` and `decrease-key` operations. For the bucket implementations, we count the number of empty buckets examined (*empty operations*). For the hot

TABLE 7.1
The performance on long grids as the grid length increases for $C = 16$.

k	nodes	8193	16385	32769	65537	131073	262145	524289
h	**time**	**0.03 s**	**0.06 s**	**0.12 s**	**0.23 s**	**0.46 s**	**0.91 s**	**1.82 s**
	heap ops.	18533	37124	74224	148476	296799	593577	1187401
b1	**time**	**0.03 s**	**0.05 s**	**0.10 s**	**0.20 s**	**0.39 s**	**0.77 s**	**1.55 s**
	empty	175	327	664	1303	2585	5120	10377
b2	**time**	**0.03 s**	**0.05 s**	**0.11 s**	**0.21 s**	**0.42 s**	**0.83 s**	**1.66 s**
	empty	129	248	502	990	1937	3861	7823
b3	**time**	**0.03 s**	**0.06 s**	**0.11 s**	**0.22 s**	**0.44 s**	**0.89 s**	**1.77 s**
	empty	85	157	325	631	1237	2459	5009
h1	**time**	**0.03 s**	**0.05 s**	**0.10 s**	**0.20 s**	**0.41 s**	**0.81 s**	**1.62 s**
	empty	167	308	635	1250	2471	4899	9932
	act. bucket	3	3	4	3	4	3	3
h2	**time**	**0.03 s**	**0.05 s**	**0.11 s**	**0.21 s**	**0.42 s**	**0.84 s**	**1.69 s**
	empty	128	245	498	984	1926	3840	7791
	act. bucket	5	5	5	4	6	4	7
h3	**time**	**0.03 s**	**0.06 s**	**0.11 s**	**0.22 s**	**0.45 s**	**0.89 s**	**1.78 s**
	empty	74	139	288	553	1089	2160	4380
	act. bucket	70	124	268	517	1031	1950	4206

TABLE 7.2
The performance on long grids as the grid length increases for $C = 100,000,000$.

k	nodes	8193	16385	32769	65537	131073	262145	524289
h	**time**	**0.03 s**	**0.06 s**	**0.12 s**	**0.24 s**	**0.47 s**	**0.94 s**	**1.87 s**
	heap ops.	18861	37751	75550	151025	302002	604132	1208105
b1	**time**	**0.63 s**	**1.55 s**	**4.20 s**	**9.00 s**	**27.84 s**	**137.39 s**	**399.30 s**
	empty	12065884	30372824	83348762	179705524	563449790	2820960848	3932274035
b2	**time**	**0.03 s**	**0.07 s**	**0.13 s**	**0.26 s**	**0.52 s**	**1.10 s**	**2.45 s**
	empty	119761	302294	594361	1321307	2639569	7066638	21872187
b3	**time**	**0.03 s**	**0.07 s**	**0.13 s**	**0.26 s**	**0.53 s**	**1.10 s**	**2.27 s**
	empty	51586	119623	256824	516687	1213386	4116075	10405346
h1	**time**	**0.04 s**	**0.08 s**	**0.15 s**	**0.26 s**	**0.51 s**	**1.03 s**	**2.05 s**
	empty	0	0	0	0	0	0	0
	act. bucket	9607	19212	38454	76910	153640	307515	615035
h2	**time**	**0.03 s**	**0.06 s**	**0.12 s**	**0.23 s**	**0.46 s**	**0.98 s**	**1.98 s**
	empty	24239	53988	144058	288742	770482	2944406	5590656
	act. bucket	1171	1994	3023	5996	9150	43279	250373
h3	**time**	**0.03 s**	**0.06 s**	**0.12 s**	**0.23 s**	**0.46 s**	**0.91 s**	**1.80 s**
	empty	12117	19554	38965	47720	108401	226167	462957
	act. bucket	3065	6870	12983	27484	51835	97991	154045

queue implementations, we count the number of empty operations and the number of `insert` and `decrease-key` operations on the active bucket.

For some problem families we also plot the data. The plots help to identify crossover points and asymptotic trends. In the plots, we use a logarithmic scale whenever appropriate. With one exception (the HARD-2 problem family), we omit the curves for the `b3` and `h3` implementations. These curves would always be close to those for `b2` and `h2`, respectively.

We were unable to run 1-level bucket and hot queue implementations on some problems because of memory limitations. We leave the corresponding table entries blank.

Network structure and arc length distribution determine the distribution of the priority queue operations. We organize the discussion according to graph types.

7.1. Long grids. For long grids, during a shortest path computation the priority queue contains a small number of elements. Because of this, the heap performs very well.

When C is small, all codes perform similarly, with the growth rate that is very close to linear. (See Table 7.1.) As C grows, the relative performance of `b1` becomes worse, due to the large number of empty operations it executes. In Table 7.2 we show results for $C = 100,000,000$. One can see that `b1` performs very poorly. In contrast, the dependence of `h1` on C is slight. For 2 and 3 levels, hot queues perform slightly better than the corresponding bucket codes.

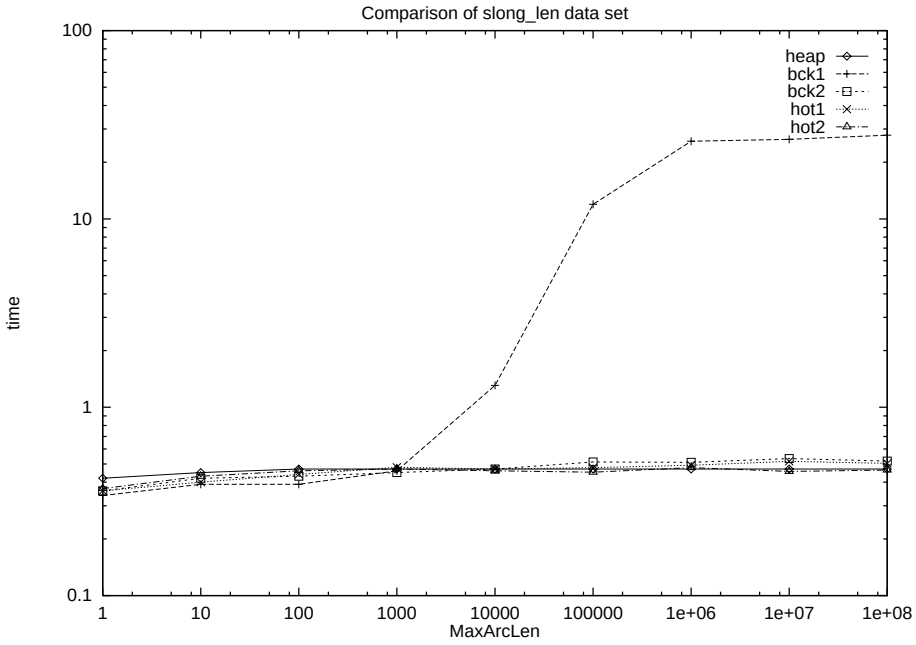


FIG. 7.1. *The performance on long grids as C increases.* $n = 131,073$.

TABLE 7.3
The performance on long grids as C increases. $n = 131,073$.

k	MaxArcLen	1	10	100	1000	10000
h	**time**	**0.42 s**	**0.45 s**	**0.47 s**	**0.47 s**	**0.47 s**
	heap ops.	262143	293966	301124	301916	301996
b1	**time**	**0.34 s**	**0.39 s**	**0.39 s**	**0.46 s**	**1.31 s**
	empty	0	346	164121	2492467	26194603
b2	**time**	**0.36 s**	**0.42 s**	**0.43 s**	**0.45 s**	**0.47 s**
	empty	0	169	112918	487717	1377280
b3	**time**	**0.36 s**	**0.43 s**	**0.47 s**	**0.48 s**	**0.50 s**
	empty	0	169	66895	209566	505269
h1	**time**	**0.36 s**	**0.40 s**	**0.44 s**	**0.48 s**	**0.47 s**
	empty	0	316	160231	505444	0
	act. bucket	3	3	5	118910	156758
h2	**time**	**0.37 s**	**0.43 s**	**0.46 s**	**0.47 s**	**0.46 s**
	empty	0	166	93807	40378	113032
	act. bucket	3	10	8165	56133	32072
h3	**time**	**0.37 s**	**0.43 s**	**0.47 s**	**0.48 s**	**0.48 s**
	empty	0	166	19900	58450	229964
	act. bucket	3	12	44872	35397	36976

k	MaxArcLen	100000	1000000	9999994	99999937
h	**time**	**0.47 s**	**0.47 s**	**0.47 s**	**0.47 s**
	heap ops.	302001	302002	302002	302002
b1	**time**	**11.95 s**	**25.87 s**	**26.44 s**	**27.84 s**
	empty	236114678	522926842	534441194	563449790
b2	**time**	**0.51 s**	**0.51 s**	**0.53 s**	**0.52 s**
	empty	2636417	2349095	3213545	2639569
b3	**time**	**0.52 s**	**0.52 s**	**0.54 s**	**0.53 s**
	empty	1108578	911898	1547187	1213386
h1	**time**	**0.48 s**	**0.49 s**	**0.52 s**	**0.50 s**
	empty	0	0	0	0
	act. bucket	153196	148773	157053	153640
h2	**time**	**0.45 s**	**0.48 s**	**0.46 s**	**0.46 s**
	empty	395585	883981	599877	770482
	act. bucket	14597	8344	11346	9150
h3	**time**	**0.46 s**	**0.47 s**	**0.46 s**	**0.46 s**
	empty	175194	74555	226660	108401
	act. bucket	57888	46121	57082	51835

Table 7.3 shows the dependence on C for long grids. With the exception of `b1`, the performance is robust. Even 1-level hot queues had consistent performance across a wide range of C values. On this family, `h`, `h2`, and `h3` are the best codes, and `b2` and `b3` perform somewhat worse than their hot queue counterparts.

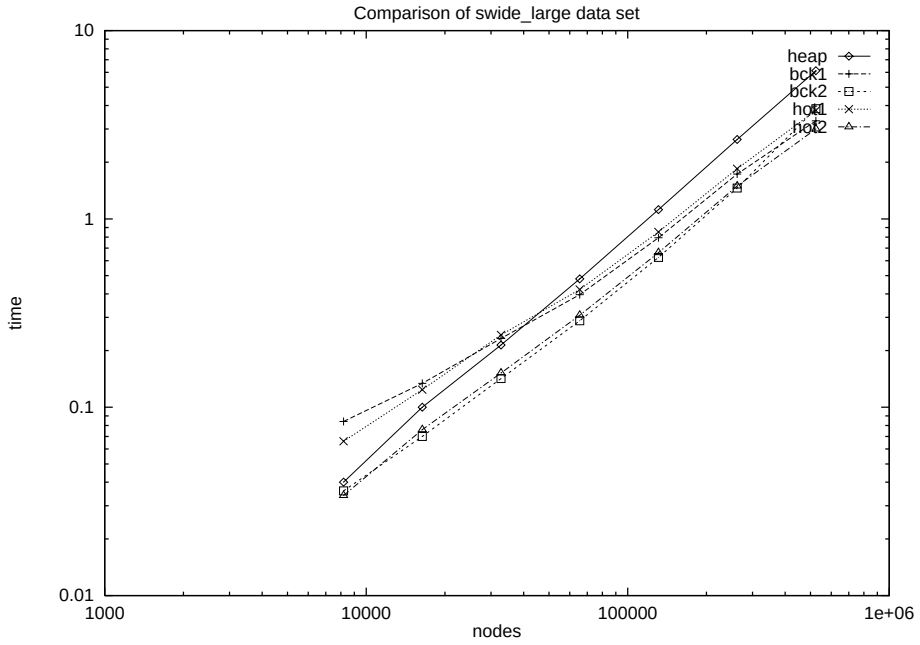


FIG. 7.2. *The performance on wide grids as the grid width increases.* $C = 100,000,000$.

TABLE 7.4
The performance on wide grids as the grid width increases. $C = 100,000,000$.

k	nodes	8193	16385	32769	65537	131073	262145	524289
h	**time**	**0.04 s**	**0.10 s**	**0.21 s**	**0.48 s**	**1.12 s**	**2.64 s**	**6.13 s**
	heap ops.	18935	37871	75656	151330	302768	603635	1207172
b1	**time**	**0.08 s**	**0.13 s**	**0.23 s**	**0.40 s**	**0.79 s**	**1.73 s**	**3.32 s**
	empty	519767	676382	898934	966873	1527813	3901062	5620110
b2	**time**	**0.04 s**	**0.07 s**	**0.14 s**	**0.29 s**	**0.62 s**	**1.46 s**	**3.85 s**
	empty	254236	439268	667632	769506	1241470	6213185	37317210
b3	**time**	**0.03 s**	**0.07 s**	**0.14 s**	**0.30 s**	**0.65 s**	**1.44 s**	**3.16 s**
	empty	43744	110229	243210	423860	835800	3507272	12187746
h1	**time**	**0.07 s**	**0.12 s**	**0.24 s**	**0.42 s**	**0.85 s**	**1.85 s**	**3.77 s**
	empty	29512	78997	257215	751682	1215514	2841845	5804273
	act. bucket	6030	12404	14835	3859	5343	15860	26190
h2	**time**	**0.03 s**	**0.08 s**	**0.15 s**	**0.31 s**	**0.66 s**	**1.49 s**	**3.00 s**
	empty	117655	358586	623911	737503	1185921	5023437	6993795
	act. bucket	1457	892	468	317	473	8638	105214
h3	**time**	**0.03 s**	**0.07 s**	**0.14 s**	**0.31 s**	**0.68 s**	**1.45 s**	**3.05 s**
	empty	9899	23825	85944	268493	705023	2301808	5420782
	act. bucket	2551	5926	10198	10779	7409	24449	62292

Operation counts give insight into performance of the code. Consider, for instance, Table 7.3. For `b1`, the number of empty bucket operations grows with C. For $C = 1,000$, the number of empty operations exceeds the graph size $(n + m)$ and soon thereafter these operations dominate the running time. Note that at the values of C around $1,000$, operation distribution for the corresponding hot queue code, `h1`, changes. For the smaller values of C, `h1` performs almost no active bucket operations because t is small and the active bucket is usually expanded. For the larger values of C, the top level bucket is usually not expanded and the code performs no empty operations. Different hot queue codes have different trade-offs between empty operations and active bucket operations.

The data presented above show that, on long grids, hot queue codes, especially `h2` and `h3`, perform very well and are very robust. Other tests we conducted, including those where C grows with n, confirm this conclusion.

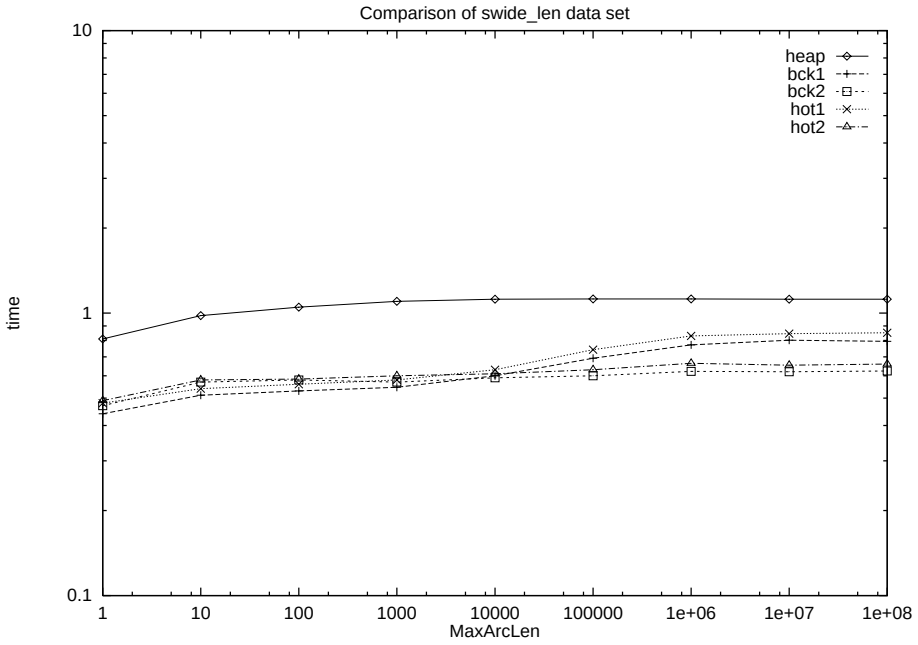


FIG. 7.3. *The performance on wide grids as C increases. $n = 131,073$.*

TABLE 7.5
The performance on wide grids as C increases. $n = 131,073$.

k	MaxArcLen	1	10	100	1000	10000	100000	1000000	9999994	99999937
h	**time**	**0.81 s**	**0.98 s**	**1.05 s**	**1.10 s**	**1.12 s**	**1.12 s**	**1.12 s**	**1.12 s**	**1.12 s**
	heap ops.	262143	294257	301848	302672	302758	302767	302768	302768	302768
b1	**time**	**0.44 s**	**0.51 s**	**0.53 s**	**0.55 s**	**0.60 s**	**0.69 s**	**0.77 s**	**0.80 s**	**0.79 s**
	empty	0	1	29	895	22778	577430	1413606	1439151	1527813
b2	**time**	**0.47 s**	**0.57 s**	**0.58 s**	**0.57 s**	**0.59 s**	**0.60 s**	**0.62 s**	**0.62 s**	**0.62 s**
	empty	0	0	18	372	14931	479394	1133272	1186499	1241470
b3	**time**	**0.48 s**	**0.59 s**	**0.62 s**	**0.62 s**	**0.62 s**	**0.63 s**	**0.65 s**	**0.65 s**	**0.65 s**
	empty	0	0	11	248	11682	373938	719020	853076	835800
h1	**time**	**0.48 s**	**0.54 s**	**0.56 s**	**0.58 s**	**0.63 s**	**0.74 s**	**0.83 s**	**0.85 s**	**0.85 s**
	empty	0	1	22	483	22777	474794	1135801	1240804	1215514
	act. bucket	3	3	4	38	290	2176	4135	4811	5343
h2	**time**	**0.49 s**	**0.58 s**	**0.58 s**	**0.60 s**	**0.61 s**	**0.63 s**	**0.66 s**	**0.65 s**	**0.66 s**
	empty	0	0	15	252	14157	464270	1092710	1131198	1185921
	act. bucket	3	4	4	19	29	161	404	440	473
h3	**time**	**0.50 s**	**0.58 s**	**0.61 s**	**0.63 s**	**0.64 s**	**0.66 s**	**0.68 s**	**0.67 s**	**0.68 s**
	empty	0	0	6	176	10564	354793	554144	772748	705023
	act. bucket	3	4	7	22	158	1294	11694	3744	7409

7.2. Wide grids and random graphs. For wide grids and random graphs, most of the time during a shortest path computation the priority queue contains a large number of elements. The distribution of the keys depends on the arc length distribution. For the same arc length distribution, computational results for the random graphs are similar to those for wide grids, and we present only the latter. For wide grids, `h` is consistently the worst performer because of the large heap size.

The key distribution is nonclustered and, when C is not much larger than the number of elements on the heap, one would expect bucket codes to perform well. The data presented in Table 7.4 confirm the expected behavior for large enough values of n. For $C = 100,000,000$, the 1-level bucket and hot queue codes outperform the heap code even for the smallest n.

Table 7.5 gives data for wide grids as C grows. All codes are robust; `b1` and `h1` show a slight dependence of C but outperform `h` for all values of C we test.

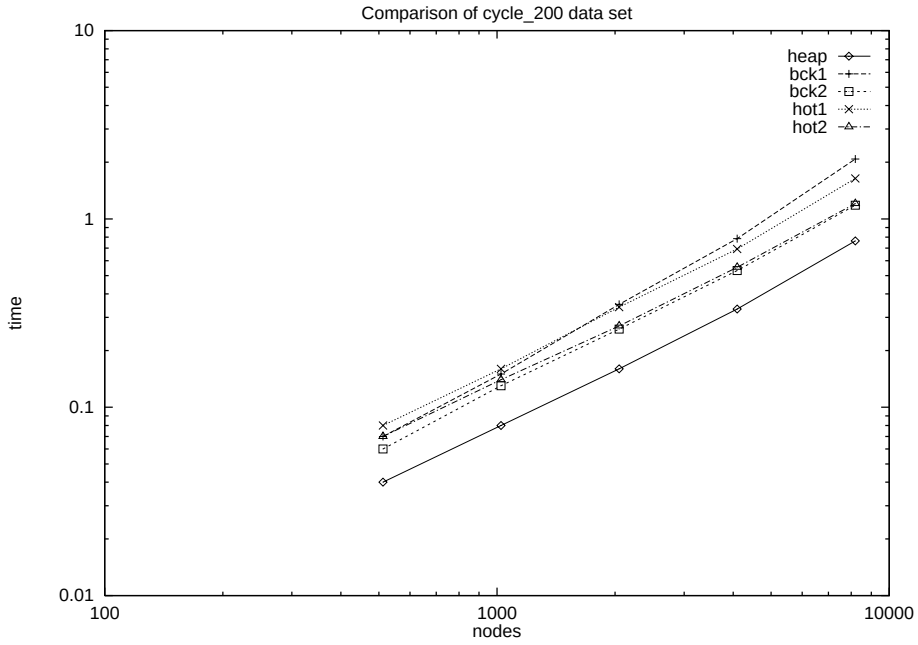


FIG. 7.4. *The performance on cycle graphs as n increases. Degree d = 200.*

TABLE 7.6
The performance on cycle graphs as n increases. Degree d = 200.

k	nodes	512	1024	2048	4096	8192
h	**time**	**0.04 s**	**0.08 s**	**0.16 s**	**0.33 s**	**0.76 s**
	heap ops.	51855	103927	207893	415548	831172
b1	**time**	**0.07 s**	**0.15 s**	**0.35 s**	**0.79 s**	**2.08 s**
	empty	0	0	0	0	0
b2	**time**	**0.06 s**	**0.13 s**	**0.26 s**	**0.53 s**	**1.18 s**
	empty	0	0	0	0	0
b3	**time**	**0.06 s**	**0.12 s**	**0.24 s**	**0.48 s**	**1.07 s**
	empty	0	0	0	0	0
h1	**time**	**0.08 s**	**0.16 s**	**0.34 s**	**0.69 s**	**1.64 s**
	empty	0	0	0	0	0
	act. bucket	51344	102904	205846	411453	822981
h2	**time**	**0.07 s**	**0.14 s**	**0.27 s**	**0.55 s**	**1.21 s**
	empty	0	0	0	0	0
	act. bucket	3	3	6	22	53
h3	**time**	**0.07 s**	**0.13 s**	**0.26 s**	**0.52 s**	**1.13 s**
	empty	0	0	0	0	0
	act. bucket	84	11	229	74	1371

Further experiments confirm that on wide grids and random graphs, even if C grows with the graph size, all bucket and hot queue codes perform well, outperforming the heap for all but the smallest graphs.

7.3. Cycle graphs. For many shortest path problems, including our grid and random graph problems, the number of `decrease-key` operations is much less than the worst case bound of m. The average-case analysis of [17] gives a theoretical explanation for this phenomena.

For the cycle graph problems, the number of `decrease-key` operations is large. Table 7.6 gives data for the cycle family with $d = 200$. Cycle graphs cause many `decrease-key` operations and no empty operations. For $d = 200$, `decrease-key` operations dominate the running time.

Experimental results for this problem family are surprising. One would expect `b1` to perform the best, since it is not hampered by examining many empty buckets

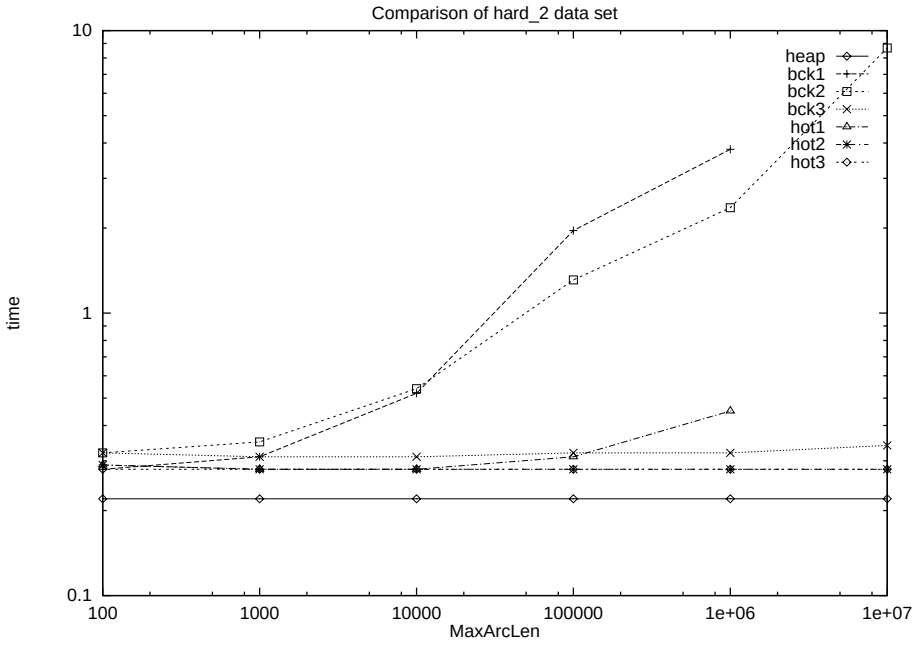


FIG. 7.5. *A graph designed to be hard for the* 2*-level bucket implementation.* $n = 131,072$.

TABLE 7.7
A graph designed to be hard for the 2*-level bucket implementation.* $n = 131,072$.

k	MaxArcLen	100	1000	10000	100000	1000000	10000000
h	**time**	**0.22 s**	**0.22 s**	**0.22 s**	**0.22 s**	**0.22 s**	**0.22 s**
	heap ops.	262143	262143	262143	262143	262143	262143
b1	**time**	**0.28 s**	**0.31 s**	**0.52 s**	**1.96 s**	**3.80 s**	
	empty	917488	1966048	8257408	33422848	66976768	
b2	**time**	**0.32 s**	**0.35 s**	**0.54 s**	**1.31 s**	**2.36 s**	**8.68 s**
	empty	917488	1966048	8257408	33422848	66976768	268300288
b3	**time**	**0.32 s**	**0.31 s**	**0.31 s**	**0.32 s**	**0.32 s**	**0.34 s**
	empty	3	7	131085	393209	393209	917489
h1	**time**	**0.29 s**	**0.28 s**	**0.28 s**	**0.31 s**	**0.45 s**	
	empty	0	0	0	0	0	
	act. bucket	122880	129024	130560	130816	131008	
h2	**time**	**0.29 s**	**0.28 s**	**0.28 s**	**0.28 s**	**0.28 s**	**0.28 s**
	empty	0	0	0	0	0	0
	act. bucket	126976	129024	130560	130944	131008	131056
h3	**time**	**0.28 s**	**0.28 s**	**0.28 s**	**0.28 s**	**0.28 s**	**0.28 s**
	empty	0	0	0	0	0	0
	act. bucket	129024	130560	130816	130944	131040	131056

and has constant-time `decrease-key` performance. Likewise, one would expect `h`, with its $O(\log n)$ implementation of `decrease-key`, to perform the worst. However, `h` is the fastest overall, while `b1` is the slowest.

To explain this phenomenon, we measured the average number of heap levels an element goes up during a `decrease-key` operation. This number is less then two even for the biggest problem size. As a result, for these problems the `decrease-key` operation in the heap is more efficient than in the buckets.

To decrease a key in the `b1` implementation, one needs to remove an element from a list, compute its new position, and insert it into a list. This is more work than moving an element up one level in our array-based implementation of the heap.

The 2- and 3-level bucket implementations perform better than the 1-level implementation. This is because in many cases a `decrease-key` operation does not move the element. We must compute the new bucket position in any case (as a witness

that the element does not move), and the work involved in this computation is non-negligible compared with the work to perform a `decrease-key` operation in the heap. Because of this, the 2- and 3-level buckets perform worse than the heap.

Next we discuss the hot queue implementations. Compared with `b1`, `h1` moves fewer elements from one bucket to another. This is because some of the operations happen within the active bucket, while others take place in the large, special top level bucket. Because of this, `h1` is a little faster. The `h2` and `h3` codes perform similarly to the corresponding bucket codes.

7.4. Hard problems. Hard problems were designed to separate the performance of various multilevel bucket implementations. Experimental results on these problems motivated the development of hot queues, which were designed to improve performance on such problems. Table 7.7 shows that, indeed, hot queues perform much better on HARD-2 than their bucket-only counterparts. With at most two elements on the heap at any time, the heap implementation is the most efficient on the hard problems.

For the hot queue implementations, no bucket is expanded and the action is confined to the two special top level buckets. Thus hot queues perform well, although not quite as well as the heap. The only exception is `h1` for the largest value of C it could handle, where its running time is significantly greater than for other values of C. We attribute this to the time taken to initialize the buckets, which we do even if the buckets are never used.

The HARD-2 problems are hard for `b1` and `b2`, and, as expected, these implementations are asymptotically worse than the other codes on this family. One can also design hard problems for buckets with more levels. In particular, we tested hard problems for 3-level buckets, with predictable results: `b3` performed asymptotically worse, although the difference between its performance and that of `h` and the hot queue codes was not as drastic as for the HARD-2 problems.

8. Concluding remarks. The hot queue data structure can use any monotone s-heap. Concurrently with our work, Raman [18] developed a deterministic s-heap that implies a deterministic hot queue that is almost as fast as that described in our paper; in particular, this result implies an $O(m + n(\log C)^{\frac{1}{3}} \log\log C)$-time deterministic implementation of Dijkstra's algorithm. In a latter paper, Raman [19] develops a faster randomized s-heap that leads to a faster randomized hot queue and to an improved shortest path bound.

The hot queue data structure combines the best features of heaps and multilevel buckets in a natural way. In theory, if C is very small compared with N, the data structure performs as the multilevel bucket structure. If C is very large, the data structure performs as the heap used in it. For intermediate values of C, the data structure performs better than either the heap or the multilevel bucket structure.

Implementing the hot queue data structure efficiently is nontrivial. Our implementation uses carefully chosen parameter values as well as the sorted list data structure for small heaps. Our experiments show that, in the context of Dijkstra's algorithm, the resulting implementations with appropriate number of levels are more robust than the heap or the multilevel bucket implementations. Our previous studies [8, 16] have shown that multilevel bucket implementations of Dijkstra's algorithm compare favorably with other implementations. The fact that in the current study, hot queues were always competitive with, and in some cases more efficient than, multilevel buckets is very interesting, in spite of the fact that the performance difference is often small.

The 3-level structure is very robust for a very wide range of values of C. For small values of C, 1- or 2-level hot queues may be more efficient, but the win is not big. It is possible that for extremely huge values of C more 3 levels are better. A variable number of levels, dependent on C, may be used in practice. Our data suggest that performance of the resulting data structure will not be sensitive to the exact switchover values.

The 1-level hot queue can be viewed as a robust version of the calendar queue [5], a data structure developed for event simulation applications. It would be interesting to see how hot queues perform for these applications.

Acknowledgments. We would like to thank Bob Tarjan and Rajeev Raman for stimulating discussions and insightful comments, Satish Rao for suggesting an adaptive strategy for switching from lists to heaps, and Harold Stone for useful comments on a draft of this paper.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.

[2] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan, *Faster algorithms for the shortest path problem*, J. Assoc. Comput. Mach., 37 (1990), pp. 213–223.

[3] P. V. E. Boas, R. Kaas, and E. Zijlstra, *Design and implementation of an efficient priority queue*, Math. Systems Theory, 10 (1977), pp. 99–127.

[4] G. S. Brodal, *Worst-case efficient priority queues*, in Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1996, pp. 52–58.

[5] R. Brown, *Calendar queues: A fast $O(1)$ priority queue implementation for the simulation event set problem*, Comm. Assoc. Comput. Mach., 31 (1988), pp. 1220–1227.

[6] B. V. Cherkassky and A. V. Goldberg, *Heap-on-Top Priority Queues*, Tech. report 96-4, NEC Research Institute, Princeton, NJ, 1996. Available online at http://www.neci.nj.nec.com/tr/index.html

[7] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, *Shortest paths algorithms: Theory and experimental evaluation*, in Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1994, pp. 516–525.

[8] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, *Shortest paths algorithms: Theory and experimental evaluation*, Math. Programming, 73 (1996), pp. 129–174.

[9] B. V. Cherkassky, A. V. Goldberg, and C. Silverstein, *Buckets, Heaps, Lists, and Monotone Priority Queues*, Tech. report 96-070, NEC Research Institute, Princeton, NJ, 1996.

[10] R. Cole and U. Vishkin, *Deterministic coin tossing with applications to optimal parallel list ranking*, Inform. and Control, 70 (1986), pp. 32–53.

[11] E. V. Denardo and B. L. Fox, *Shortest-route methods:* 1. *Reaching, pruning, and buckets*, Oper. Res., 27 (1979), pp. 161–186.

[12] R. B. Dial, *Algorithm* 360*: Shortest path forest with topological ordering*, Comm. Assoc. Comput. Mach., 12 (1969), pp. 632–633.

[13] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[14] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. Assoc. Comput. Mach., 34 (1987), pp. 596–615.

[15] M. L. Fredman and D. E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. System Sci., 48 (1994), pp. 533–551.

[16] A. V. Goldberg and C. Silverstein, *Implementations of Dijkstra's algorithm based on multi-level buckets*, in Lecture Notes in Econom. and Math. Systems 450, P. M. Pardalos, D. W. Hearn, and W. W. Hages, eds., Springer-Verlag, New York, 1997, pp. 292–327.

[17] K. Noshita, *A theorem on the expected complexity of Dijkstra's shortest path algorithm*, J. Algorithms, 6 (1985), pp. 400–408.

[18] R. Raman, *Priority queues: Small, monotone and trans-dichotomous*, in Proc. 4th Annual European Symposium Algorithms, Lecture Notes in Comput. Sci. 1136, Springer-Verlag, New York, 1996, pp. 121–137.

[19] R. Raman, *Recent results on single-source shortest paths problem*, SIGACT News, 28 (1997), pp. 81–87.

[20] S. Rao, *personal communication,* 1996.
[21] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, 1983.
[22] M. Thorup, *On RAM priority queues*, in Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1996, pp. 59–67.
[23] J. W. J. Williams, *Algorithm* 232 *(Heapsort)*, Comm. Assoc. Comput. Mach., 7 (1964), pp. 347–348.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1347–1363

DISTRIBUTED ANONYMOUS MOBILE ROBOTS: FORMATION OF GEOMETRIC PATTERNS*

ICHIRO SUZUKI† AND MASAFUMI YAMASHITA‡

Abstract. Consider a system of multiple mobile robots in which each robot, at infinitely many unpredictable time instants, observes the positions of all the robots and moves to a new position determined by the given algorithm. The robots are anonymous in the sense that they all execute the same algorithm and they cannot be distinguished by their appearances. Initially they do not have a common x-y coordinate system. Such a system can be viewed as a distributed system of anonymous mobile processes in which the processes (i.e., robots) can "communicate" with each other only by means of their moves. In this paper we investigate a number of formation problems of geometric patterns in the plane by the robots. Specifically, we present algorithms for converging the robots to a single point and moving the robots to a single point in finite steps. We also characterize the class of geometric patterns that the robots can form in terms of their initial configuration. Some impossibility results are also presented.

Key words. distributed algorithms, anonymous robots, mobile robots, multiagent systems, formation of geometric patterns

AMS subject classification. 68Q99

PII. S009753979628292X

1. Introduction.

Suppose that a schoolteacher wants her 100 children in the playground to form a circle so that, for instance, they can play a game. She might draw a circle on the ground as a guideline or even give each child a specific position to move to. What if the teacher does not provide such assistance? Even without such assistance, the children may still be able to form a sufficiently good approximation of a circle if each of them moves adaptively based on the movement of other children and knowledge of the shape of a circle. If successful, this method can be called a *distributed* solution to the circle formation problem for children.

A similar distributed approach can be used for controlling a group of multiple mobile robots. The main idea is to let each robot execute a simple algorithm and plan its motion adaptively based on the observed movement of other robots, so that the robots as a group will achieve the given goal. The objective of this paper is to give a formal discussion on the power and limitations of the distributed control method in the context of the formation problems of geometric patterns in the plane.

The problem of forming an approximation of a circle having a given diameter by identical mobile robots was first discussed by Sugihara and Suzuki [13].[1] Assuming

*Received by the editors November 13, 1996; accepted for publication (in revised form) July 24, 1997; published electronically March 30, 1999. This work was supported in part by a Scientific Research Grant-in-Aid from the Ministry of Education, Science and Culture in Japan (0668032, 07243219, 076803604), the National Science Foundation under grant IRI-9307506, the Office of Naval Research under grant N00014-94-1-0284, and an endowed chair supported by Hitachi Ltd. at the Faculty of Engineering Science, Osaka University. An earlier version of some of the results contained in this paper appears as "Formation and agreement problems for anonymous mobile robots" in *Proc. 31st Annual Allerton Conference on Communication, Control, and Computing*, University of Illinois, Urbana, 1993, pp. 93–102.

http://www.siam.org/journals/sicomp/28-4/28292.html

†Department of Electrical Engineering and Computer Science, University of Wisconsin–Milwaukee, P.O. Box 784, Milwaukee, WI 53201 (suzuki@cs.uwm.edu).

‡Department of Electrical Engineering, Faculty of Engineering, Hiroshima University, Kagamiyama, Higashi-Hiroshima 739, Japan. Presently with Department of Computer Science and Communication Engineering, Kyushu University, Fukuoka, 812-8581, Japan (mak@csce.kyushu-u.ac.jp).

[1] In our terminology, the problem they consider is a convergence problem for a circle.

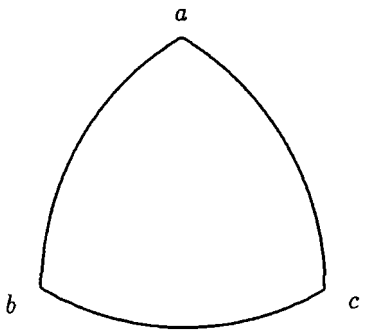


FIG. 1.1. *Reuleux's triangle.*

that the positions of the robots are the only information available, they proposed a simple heuristic distributed algorithm (to be executed independently by all robots), which, according to simulation results, sometimes brings the robots to a pattern reminiscent of a Reuleaux's triangle (Figure 1.1) rather than a circle. Tanaka [16] later improved their algorithm and demonstrated, using simulation, that his new algorithm avoids this problem and generates a better approximation of a circle. In essence, in his algorithm each robot simply adjusts its position regarding the midpoint of the positions of the nearest and farthest neighbors as the center of the circle to which the robots are converging, while moving away from its nearest neighbor if the distance to that midpoint is approximately equal to the given target radius. Figure 1.2 shows the behavior of 50 robots executing his algorithm starting from an initial distribution generated randomly. This extremely simple algorithm demonstrates the potential of the distributed method. The circle formation problem was also discussed recently by Debest [2] from the viewpoint of self-stabilization. A system is said to be *self-stabilizing* if it recovers from any finite number of transient errors [12], and thus self-stabilizing robot algorithms are robust against a finite number of sensor and control errors.

Formation problems of geometric patterns are closely related to certain agreement problems. Agreement on a common x-y coordinate system by the robots, for instance, can greatly reduce the complexity of motion coordination algorithms; e.g., convergence toward a single point can easily be solved by moving all the robots toward point $(0, 0)$ of the common coordinate system. However, such a simple solution is not possible if the robots have only their own local coordinate systems, whose origins may or may not agree. It is sometimes assumed in the literature, therefore, that either there exists a global coordinate system or that some navigation devices (e.g., a variety of potential functions [18], compasses [3], or beacons and lighthouses [4]) are available to compensate for the lack of such a system. Note here that the agreement problem on a common coordinate system can (partially) be reduced to certain formation problems: If the robots can form (i.e., gather at) a single point, then they can agree to use that point as the origin of the common coordinate system. Similarly, formation of a circle implies agreement on both the origin and the unit distance (i.e., the center and the radius of the circle). Formation of a symbol "$\top$" implies agreement on the origin, the unit distance, and the positive x-direction, i.e., agreement on a common x-y coordinate system.

Related work on the distributed robot control method includes the following. Wang and Beni [17] considered a cellular robotic system consisting of a large number

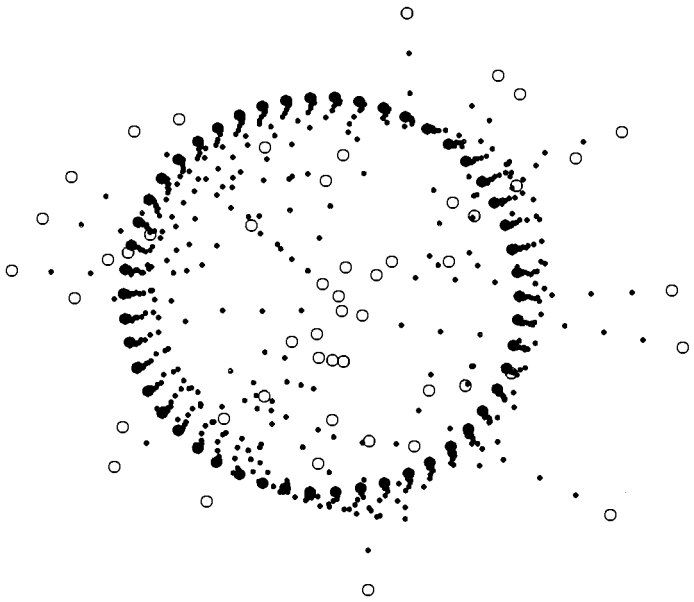


FIG. 1.2. *Hollow circles are the initial positions of* 50 *robots. Solid circles are their final positions after execution of Tanaka's algorithm. Small dots represent their intermediate positions.*

of robots that operate in a cellular space under distributed control. They discussed the problem of generating certain one- and two-dimensional cellular patterns using distributed control and showed how the technique can be applied to the design of sensor arrays and escape systems. Fukuda and Nakagawa [6] and Kawauchi, Inaba, and Fukuda [7] considered a dynamically reconfigurable robotic system called CEBOT, which consists of many simple cells that can detach and combine autonomously to change its overall shape, depending on the task and the environment. Kokaji [8] and Murata, Kurokawa, and Kokaji [10] designed self-reorganizing systems called Fractal Machine and Fractum, respectively, based on a similar idea (but unlike CEBOT, these systems consist of homogeneous units) and discussed dynamic reconfiguration based on a set of local rules. Fujimura [5] investigated how planning algorithms, knowledge about the environment, and action intervals of the robots affect the overall performance of two robots moving toward their respective goal positions while avoiding collision. Sugihara and Suzuki [13], [14], and Suzuki and Yamashita [15] considered formation and agreement problems for anonymous mobile robots in the plane. Work by others includes swarm intelligence [1] and collective behavior of multiple robots [9], [11].

The main emphasis of most of the work mentioned above has been on the development of heuristic algorithms for various problems, and rigorous proofs of the correctness of these algorithms have not been given. In contrast, as we stated earlier in this paper, we conduct a formal investigation on the power and limitations of the distributed control method.

We model a robot as a mobile processor with infinite memory and a sensor for detecting the positions of other robots[2] that repeatedly becomes *active* at infinitely

[2]We assume that the sensor cannot measure the velocity or acceleration of other robots and that other navigation devices such as compasses and beacons are not available.

many unpredictable time instants. (At other times it is *inactive*.) We assume that initially the robots do not have a common x-y coordinate system and that the local x-y coordinate systems of the robots may not agree on the location of the origin, the unit distance, or the direction of the positive x-axis. Each time a robot becomes active, using its sensor it observes the positions of all the robots in terms of its own local x-y coordinate system and moves to a new position determined by the given deterministic algorithm.[3] The algorithm is *oblivious* if the new position is determined only from the positions of the robots observed at that time instant. Otherwise, it is *nonoblivious*, and the new position may depend also on the observations made in the past. Note that oblivious algorithms are self-stabilizing by definition. To simplify the discussion and bring forth the fundamental issues of the problem, in this paper we assume that (1) the initial positions of the robots are all distinct, (2) the time it takes for a robot to move to its new position is negligibly small, and (3) a robot is a point (so two robots can occupy the same position simultaneously and never collide). The robots are anonymous in the sense that (1) they do not know their identifiers, (2) they all use the same algorithm for determining the next position, and (3) they cannot be distinguished by their appearances.

Let π be a predicate describing a geometric pattern, such as a point, a regular polygon, a line segment, etc. On the one hand, we say that an algorithm ψ solves the *convergence problem* for π if the robots' distribution converges to one that satisfies π, regardless of the number n of robots, their initial distribution, and the timing with which they become active. On the other hand, we say that ψ solves the *formation problem* for π if the robots eventually reach a distribution that satisfies π in a finite number of steps, regardless of n, their initial distribution, and the timing with which they become active. (See section 2 for formal definitions of these concepts.)

We begin with a simple problem of converging the robots toward a single point. (That is, this is the convergence problem for a predicate π that describes a point. Note that the process of convergence need not terminate in finite steps.) Note again that since the robots do not have a common x-y coordinate system, we cannot simply use an algorithm such as "move toward the origin $(0, 0)$." For this problem we give a simple oblivious algorithm.

We also consider the formation problem for a point, in which the robots must form (i.e., gather at) a single point in finite steps. We show that this problem can be solved by a nonoblivious algorithm for any $n \geq 2$ and by an oblivious algorithm for any $n \geq 3$, but it is not solvable by any oblivious algorithm for the case $n = 2$, where n is the total number of robots.

Finally, we characterize the class of geometric patterns for which the formation problem is solvable in our model. We do so by first examining the class of patterns that the robots can form, starting from a given initial configuration. Our main observation is that since the robots may happen to become active simultaneously all the time (i.e., their motions turn out to be synchronized) and (by definition) algorithms are required to solve the given problem regardless of the timing with which the robots become active, the robots may not be able to break the "symmetry" that exists in their initial distribution by executing an algorithm (which is deterministic by definition). Based on this and using techniques that have been developed for anonymous complete networks in [19], [20], we prove that the formation problem is solvable by an algorithm (in the sense defined above) only for two patterns: a point and a regular n-gon. The

[3] In this paper we do not consider nondeterministic algorithms that allow a robot to randomly select its next position from two or more candidates.

algorithm we present for the formation of a regular n-gon is nonoblivious. Whether an oblivious algorithm exists for this problem remains open.

We present necessary definitions and basic assumptions in section 2. Convergence and formation problems for a point are discussed in section 3. Section 4 gives a characterization of the class of geometric patterns that the robots can form in our model. Discussions and concluding remarks are presented in section 5.

2. Definitions and basic assumptions. We formalize the concepts described in section 1. Let $r_1, r_2, \ldots, r_n$ be the robots in a two-dimensional space. (The subscript i of r_i is used for convenience of explanation. The robots do not know their identifiers.) We denote by $Z_i = (o_i, d_i, u_i)$, $1 \leq i \leq n$, the local x-y coordinate system of r_i, where o_i, d_i, and u_i denote the position of the origin, direction of the positive x-axis, and size of the unit distance, respectively, under Z_i. It is possible that $Z_i \neq Z_j$ for some i and j, but the robots are assumed to have a common sense of orientation so that in each Z_i, the positive y-direction is 90 degrees counterclockwise from the positive x- direction. As we describe below, all robot positions that r_i observes and computes are given in terms of Z_i.

We assume discrete time $0, 1, 2, \ldots$ and let $p_i(t)$ be the position of r_i at time instant t, where $p_i(0)$ is the initial position of r_i. We assume that $p_1(0), p_1(0), \ldots, p_n(0)$ are all distinct. Define $P(t) = \{p_i(t) | 1 \leq i \leq n\}$ to be the multiset of the positions of the robots at time t. ($P(t)$ is a multiset, since we assume that two robots can occupy the same position simultaneously.) For any point p, we denote by $[p]_j$ the position of p given in terms of Z_j and define $[P(t)]_j = \{[p_i(t)]_j | 1 \leq i \leq n\}$. Thus $[P(t)]_j$ shows how r_j views the distribution $P(t)$ in terms of its own Z_j. Note that if $Z_j \neq Z_k$, then it is possible that $[P(t)]_j \neq [P(t)]_k$; i.e., r_j and r_k may observe distribution $P(t)$ differently. However, $[P(t)]_j = [P(t)]_k$ may hold even if $p_j(t) \neq p_k(t)$. In this case, r_j and r_k are located at different positions, but $P(t)$ looks identical to them.

At each time instant t, every robot r_i is either *active* or *inactive*. Without loss of generality we assume that at least one robot is active at every time instant. We use A_t to denote the set of active robots at t, and call the sequence $\mathcal{A} = A_0, A_1, \ldots$ an *activation schedule*. We assume that every robot becomes active at infinitely many time instants, but no additional assumptions are made on the timing with which the robots become active. Thus $\mathcal{A}$ need satisfy only the condition that every robot appears in infinitely many A_t's. Note that a special case is when every robot appears in A_t for every t; in this case we say that the robots are *synchronized*.

The algorithm that a robot uses is a function ψ such that, for any given sequence $(Q_1, p_1), (Q_2, p_2), \ldots, (Q_m, p_m)$ of pairs of a multiset Q_ℓ of points and a point $p_\ell \in Q_\ell$, $\psi((Q_1, p_1), (Q_2, p_2), \ldots, (Q_m, p_m))$ is a point such that the distance between p_m and $\psi((Q_1, p_1), (Q_2, p_2), \ldots, (Q_m, p_m))$ is at most 1. The position of a robot at $t \geq 1$ is determined by $P(0)$, $\mathcal{A}$, and ψ, as follows.

For any $t \geq 0$, if $r_i \notin A_t$ (r_i is inactive), then $p_i(t+1) = p_i(t)$; i.e., r_i does not move. If $r_i \in A_t$ (r_i is active), then let $0 \leq t_1 \leq t_2 \leq \cdots \leq t_m = t$ be the time instants when r_i has been active, and for each $1 \leq \ell \leq m$, let $Q_\ell = [P(t_\ell)]_i$ and $p_\ell = [p_i(t_\ell)]_i$ be the distribution that r_i observed and the position of r_i at t_ℓ, respectively. (Note that Q_ℓ and p_ℓ are given in terms of Z_i.) Then $p_i(t+1) = p$, where p is the point such that $[p]_i = \psi((Q_1, p_1), (Q_2, p_2), \ldots, (Q_m, p_m))$. That is, r_i moves to point $\psi((Q_1, p_1), (Q_2, p_2), \ldots, (Q_m, p_m))$ of Z_i. By the restriction on ψ stated above, the maximum distance that r_i can move in one step is the unit distance 1 of Z_i, which corresponds to some physical distance $\epsilon_i > 0$. Note that every robot is then capable of moving over distance at least $\epsilon = \min\{\epsilon_1, \epsilon_2, \ldots, \epsilon_n\} > 0$ in one step.

That is, r_i observes the distribution of the robots only when it is active, and its next position depends only on ψ and the distributions that r_i has observed so far. The p_ℓ in pair (Q_ℓ, p_ℓ) shows that r_i is always aware of its current position in Z_i. Algorithm ψ is said to be *oblivious* if $\psi((Q_1,p_1),(Q_2,p_2),\ldots,(Q_m,p_m)) = \psi((Q_m,p_m))$ for any $(Q_1,p_1),(Q_2,p_2),\ldots,(Q_m,p_m)$. In this case, the move of a robot depends only on the current configuration of the robots. Otherwise, ψ is *nonoblivious*. Note that the robots are anonymous in the following sense: (1) function ψ is common to all the robots, (2) the identifier i of robot r_i is not an argument of ψ, and (3) $[P(t)]_i$ contains only the positions of the robots (but not their identities).

Let π be a predicate over the set of multisets of points that is invariant under any rotation, translation, and uniform scaling. For example, π might be true iff the given points are on the circumference of a circle or on a line segment. For such π, we consider two types of problems: the *convergence problem* and the *formation problem*. An algorithm ψ is said to solve the convergence problem for π if, as t goes to infinity, $P(t)$ converges to a distribution that satisfies π, regardless of the number n of robots, initial distribution $P(0)$, and activation schedule $\mathcal{A}$. In contrast, in the formation problem the robots must reach a distribution satisfying π in finite steps and "halt." That is, an algorithm ψ is said to solve the formation problem for π if there exists some time instant t' such that $P(t')$ satisfies π and $p_i(t') = p_i(t'+1) = \cdots$ for all $1 \le i \le n$, regardless of n, $P(0)$, and $\mathcal{A}$. Since the robots have no knowledge of the underlying coordinate system, the robots can only converge to or form a pattern similar to the given goal pattern. The restriction on π stated above was introduced for this reason. All predicates discussed in the following sections satisfy this condition.

3. Convergence and formation problems for a point. Formally, the problem of converging the robots to a point is stated as the convergence problem for predicate π_{point}, where $\pi_{point}(p_1,\ldots,p_n) =$ true iff $p_i = p_j$ for any $1 \le i,j \le n$. We call this problem C-POINT. The corresponding formation problem for π_{point} is called F-POINT. Note that in F-POINT, all robots must occupy a single point in finite steps, whereas in C-POINT they need only converge to a single point. These are perhaps some of the simplest problems one could consider. Nevertheless, the discussions presented in this section can serve as an introduction to the technical results given in the rest of the paper. An algorithm that solves F-POINT also solves C-POINT.

For convenience, we present all algorithms by giving an informal description of the behavior of the robots executing it, instead of giving a formal definition of function ψ. Converting the informal description into a formal definition of ψ is straightforward.

It is easy to show that the following oblivious algorithm $\psi_{c-point(2)}$ solves C-POINT for the case $n = 2$.

ALGORITHM $\psi_{c-point(2)}$—OBLIVIOUS.

Each time r_i becomes active, it moves toward[4] the midpoint m of its current position and that of the other robot r_j. □

Suppose that we modify $\psi_{c-point(2)}$ so that each robot moves toward the position of the other robot. Then the two robots will continue to swap their positions if they are mutually reachable in one step and always become active simultaneously. (Recall that we assume robots never collide with each other.) Thus this modified algorithm does not solve C-POINT for $n = 2$.

Note that if exactly one robot becomes active at every time instant, then the

[4]Unless otherwise stated, "a robot moves toward point p" means that "a robot moves to the point p' closest to p that is reachable in one step from the current position." Of course, $p = p'$ if p is reachable in one step.

two robots executing $\psi_{c-point(2)}$ will never occupy the same point. Thus oblivious algorithm $\psi_{c-point(2)}$ does not solve F-POINT for $n = 2$. In fact, we have the following theorem.

THEOREM 3.1. *There is no oblivious algorithm for solving F-POINT for the case* $n = 2$.

Proof. Suppose that there is an oblivious algorithm ψ that solves F-POINT for two robots r_i and r_j. Note that since ψ is oblivious, the moves of the robots depend only on Z_i, Z_j and their current positions.

We first show that there exist distinct positions p and q of r_i and r_j, respectively, such that either (1) ψ moves r_i from p to q and r_j from q to q, or (2) ψ moves r_i from p to p and r_j from q to p. (That is, ψ moves exactly one robot to the position of the other if both robots become active simultaneously.) To see this, assume that such positions do not exist. Consider a scenario $\mathcal{S}$ in which r_i and r_j, located at distinct positions p and q, respectively, at time $t-1$ occupy the same position r at time t. Now we show that we can modify this scenario and obtain another scenario in which the robots never occupy the same position simultaneously. There are two cases.

Case 1. Both r_i and r_j are active at time $t-1$ in $\mathcal{S}$. By assumption, $r \neq p$ and $r \neq q$. Thus if exactly one robot, say, r_i, happens to be active at $t-1$, then at time t, r_i is located at r and r_j at q, where $r \neq q$.

Case 2. Exactly one robot is active at $t-1$ in $\mathcal{S}$. Suppose that r_i is active at $t-1$ but r_j is not. Then $r = q$. So if both robots happen to be active at $t-1$, then at time t, r_i is located at q and r_j at some point s, where by assumption $s \neq q$.

Using this argument repeatedly, we can construct an infinite sequence of moves in which the robots never occupy the same position simultaneously. (We can do so in such a way that each robot becomes active infinitely many times, since either of the robots can be chosen to be inactive in Case 1.) So ψ does not solve F-POINT. This is a contradiction.

Now consider an initial distribution $P(0) = \{p, q\}$ in which r_i and r_j are at p and q, respectively, and ψ moves r_i from p to q, and r_j from q to q; see Figure 3.1(a). (The case in which ψ moves r_j to the positions of r_i is similar.) Now, by modifying Z_i through translation and rotation, we can construct another configuration in which r_i observes distribution $P(0)$ the same way as r_j; i.e., $[P(0)]_i = [P(0)]_j$ and $[p]_i = [q]_j$; see Figure 3.1(b). Then ψ moves r_i and r_j in the same manner in the new configuration and, of course, ψ moves r_j in the same manner in both configurations (namely, from q to q). Therefore, in the new configuration ψ moves r_i from p to p and r_j from q to q. Then, since ψ is oblivious, both robots remain in their respective initial positions forever. Thus ψ does not solve F-POINT. This is a contradiction. □

However, F-POINT can be solved for two robots by the following *nonoblivious* algorithm $\psi_{f-point(2)}$.[5]

ALGORITHM $\psi_{f-point(2)}$—NONOBLIVIOUS.

When r_i becomes active for the first time, it translates and rotates its coordinate system[6] Z_i so that

1. r_i is at $(0, 0)$ of Z_i, and
2. the other robot r_j is on the positive y-axis of Z_i, say, at $(0, a)$ for some $a > 0$.

[5] If all robots are *known* to be active at every time instant (i.e., the robots are "synchronous"), then a simple oblivious algorithm that moves both robots toward the midpoint of their current positions solves F-POINT for two robots.

[6] Formally, r_i cannot modify Z_i in our framework, but the effect of such a transformation can easily be simulated within the framework.

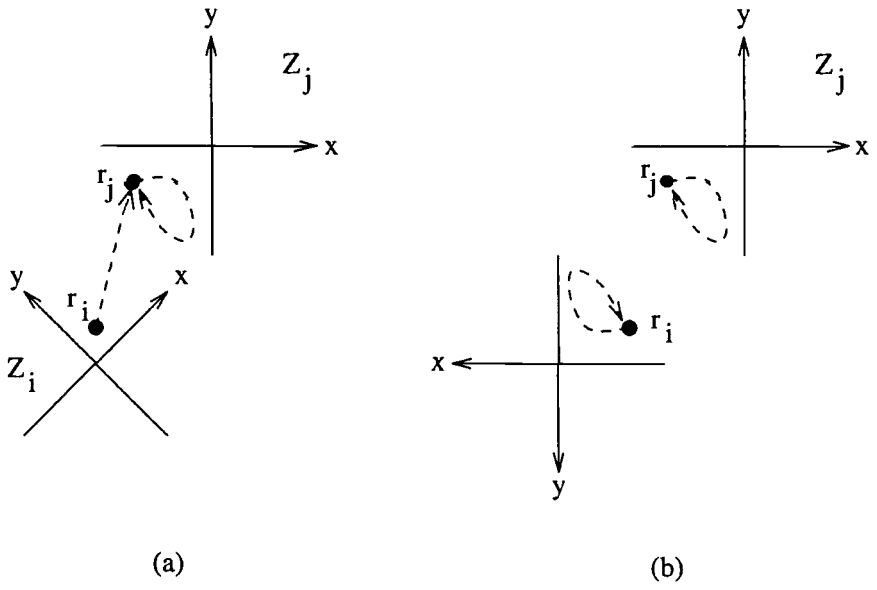


FIG. 3.1. (a) r_i *moves but* r_j *does not.* (b) *After modification of* Z_i.

Then it moves in the positive x direction of Z_i, over any nonzero distance. It then continues to move in the same direction each time it becomes active until it observes that the position of r_j has changed twice.

Now, r_i knows line ℓ that contains the first two distinct positions of r_j that r_i has observed. (Note that by symmetry ℓ is the x-axis of r_j's coordinate system Z_j.) Then using Lemma 3.2, r_i finds the initial position of r_j and moves to the midpoint of the initial positions of r_i and r_j. □

Lemma 3.2, which follows immediately from the description of $\psi_{f-point(2)}$, shows that robots r_i and r_j executing $\psi_{f-point(2)}$ eventually find out which of them became active first for the first time and what their initial distribution was.

LEMMA 3.2. *Let* t_i *and* t_j *be the time instants at which* r_i *and* r_j, *respectively, become active for the first time in* $\psi_{f-point(2)}$. *Then the following hold.*

1. *The trajectory of* r_i *and the trajectory of* r_j *are parallel iff* $t_i = t_j$. *In this case, each robot sees the other robot at its initial position at* $t_i (= t_j)$ (*Figure* 3.2(a)).
2. *The trajectory of* r_j *intersects the negative* x*-axis of* Z_i *iff* $t_i < t_j$. *In this case,* r_i *sees* r_j *at its initial position, and* r_i*'s initial position is the foot of the perpendicular drop from* r_j*'s initial position to the line containing the trajectory of* r_i (*Figure* 3.2(b)).
3. *The trajectory of* r_i *intersects the negative* x*-axis of* Z_j *iff* $t_j < t_i$. *In this case,* r_j *sees* r_i *at its initial position, and* r_j*'s initial position is the foot of the vertical drop from* r_i*'s initial position to the line containing the trajectory of* r_j.

THEOREM 3.3. *Algorithm* $\psi_{f-point(2)}$ *solves problem F-POINT for* $n = 2$.

Proof. A key observation is the following: When r_i observes that the position of r_j has changed twice, r_j must have already observed that r_i's position has changed at least once and thus r_j knows where the x-axis of Z_i is. Similarly, r_j will know that r_i knows where the x-axis of Z_j is. Then the correctness of $\psi_{f-point(2)}$ follows from Lemma 3.2. □

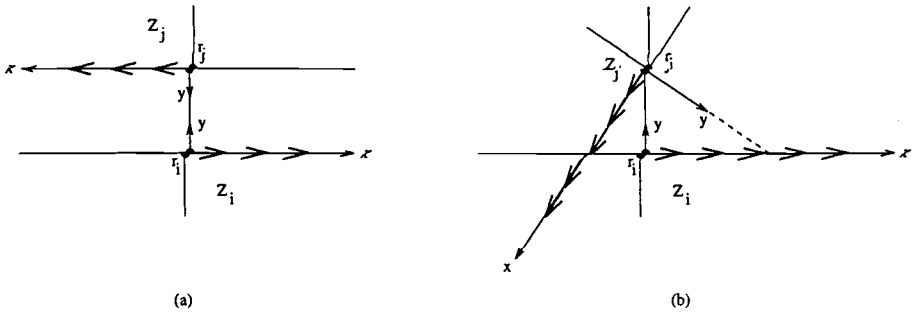


FIG. 3.2. *Illustration for* $\psi_{f-point(2)}$*;* (a) $t_i = t_j$, (b) $t_i < t_j$.

Finally, we have the following result on F-POINT and C-POINT for $n \geq 3$.

THEOREM 3.4. *There is an oblivious algorithm for solving F-POINT (and thus C-POINT) for* $n \geq 3$.

Proof. It suffices to give an oblivious algorithm $\psi_{f-point(n)}$ that solves F-POINT. The idea is the following. Starting from distinct initial positions, we move the robots in such a way that eventually there will be exactly one position, say, p, that two or more robots occupy. Once such a distribution is reached, all robots that are not located at p move toward p in such a way that no two robots will occupy the same position at any location other than p. Then all robots eventually occupy p, solving F-POINT.

Such a distribution can be obtained if each robot, each time it becomes active, determines which of the following cases applies and moves to a new position (or remains stationary) as specified. Since a robot's action is based only on the current robot distribution, this strategy can be implemented as an oblivious algorithm.

Case 1. $n = 3$; p_1, p_2, and p_3 denote the positions of the three robots.

1.1. If $n = 3$ and p_1, p_2, and p_3 are collinear with p_2 in the middle, then the robots at p_1 and p_3 move toward p_2 while the robot at p_2 remains stationary. Then eventually two robots occupy p_2.

1.2. If $n = 3$ and p_1, p_2, and p_3 form an isosceles triangle with $|\overline{p_1p_2}| = |\overline{p_1p_3}| \neq |\overline{p_2p_3}|$, then the robot at p_1 moves toward the foot of the perpendicular drop from its current position to $\overline{p_2p_3}$ in such a way that the robots do not form an equilateral triangle at any time, while the robots at p_2 and p_3 remain stationary. Then eventually the robots become collinear and the problem is reduced to part 1.1.

1.3. If $n = 3$ and the lengths of the three sides of triangle $p_1p_2p_3$ are all different, say, $|\overline{p_1p_2}| > |\overline{p_1p_3}| > |\overline{p_2p_3}|$, then the robot at p_3 moves toward the foot of the perpendicular drop from its current position to $\overline{p_1p_2}$ while the robots at p_1 and p_2 remain stationary. Then eventually the robots become collinear and the problem is reduced to part 1.1.

1.4. If $n = 3$ and p_1, p_2, and p_3 form an equilateral triangle, then every robot moves towards the center of the triangle. Since all robots can move up to at least a constant distance $\epsilon > 0$ in one step, if part 1.4 continues to hold then eventually either the robots meet at the center, or the triangle they form becomes no longer equilateral and the problem is reduced to part 1.2 or part 1.3.

Case 2. $n \geq 4$; C_t denotes the smallest enclosing circle of the robots at time t.

2.1. If $n \geq 4$ and there is exactly one robot r in the interior of C_t, then r moves toward the position of any one robot, say, r', on the circumference of C_t while all other robots remain stationary. Then eventually r and r' occupy the same position.

2.2. If $n \geq 4$ and there are two or more robots in the interior of C_t, then these robots move toward the center of C_t while all other robots remain stationary (so that the center of C_t remains unchanged). Then eventually at least two robots reach the center.

2.3. If $n \geq 4$ and there are no robots in the interior of C_t, then every robot moves toward the center of C_t. Since all robots can move up to at least a constant distance $\epsilon > 0$ in one step, if part 2.3 continues to hold, then eventually the radius of C_t becomes at most ϵ. Once this happens, then the next time some robot moves, say, at t', either (i) two or more robots occupy the center of C_t or (ii) there is exactly one robot r at the center of C_t, and therefore there is a robot that is not on $C_{t'}$ (and the problem is reduced to part 2.1 or part 2.2) since a cycle passing through r and a point on C_t intersects with C_t at most at two points. □

Suppose that for $1 \leq i \leq n$, robot r_i has (privately) chosen a directed line ℓ_i that passes through its initial position. Algorithm $\psi_{f-point(2)}$ uses a technique with which all robots can simultaneously "broadcast" the locations and directions of $\ell_1, \ell_2, \ldots, \ell_n$. The basic idea is that each robot r_i moves repeatedly along ℓ_i in the given direction until it observes that every r_j, $j \neq i$ has changed positions at least twice (i.e., until r_i sees r_j at three or more distinct positions). Then, as we explained in the proof of Theorem 3.3, every r_j, $j \neq i$ must have (become active and) seen r_i at two or more distinct positions along ℓ_i, and thus r_j can conclude that the ℓ_i that r_i has chosen passes through the first two distinct positions of r_i that r_j has observed and that ℓ_i is oriented in the direction from the first to the second positions of r_i that r_j has observed. Care must be taken so that r_i continues to move at least one more time (to any distinct position) after observing that every r_j has changed position at least twice, since at this moment some r_j might have observed r_i only at two distinct positions.

Another problem is that, since the robots are indistinguishable by their appearances, if $n > 2$, then r_j may not be able to determine how r_i has moved, given the robot distributions at two time instants. To cope with this, if $n > 2$, then we let each robot r_i memorize the distance $a_i > 0$ to its nearest neighbor when it becomes active for the first time and move at most distance $a_i/2^{k+1}$ in the kth move. Then each r_i will remain in the interior of the $a_i/2$-neighborhood of its initial position, and thus every robot can correctly determine which robot has moved to which position even after it has remained inactive for a long time.

4. Achievable geometric patterns. In this section we characterize the class of geometric patterns that the robots can form regardless of the activation schedule $\mathcal{A}$, starting from a fixed initial configuration. For simplicity of explanation we assume that each robot r_i is located at the origin of its coordinate system Z_i at time 0. Essentially the same result holds even without this assumption.

Whether or not a particular geometric pattern can be formed depends not only on the given initial positions of the robots but also on their local x-y coordinate systems. For example, suppose that, initially, four robots r_1, r_2, r_3, and r_4 form a square in counterclockwise order, where r_2 is at position $(1, 0)$ of Z_1, r_3 is at position $(1, 0)$

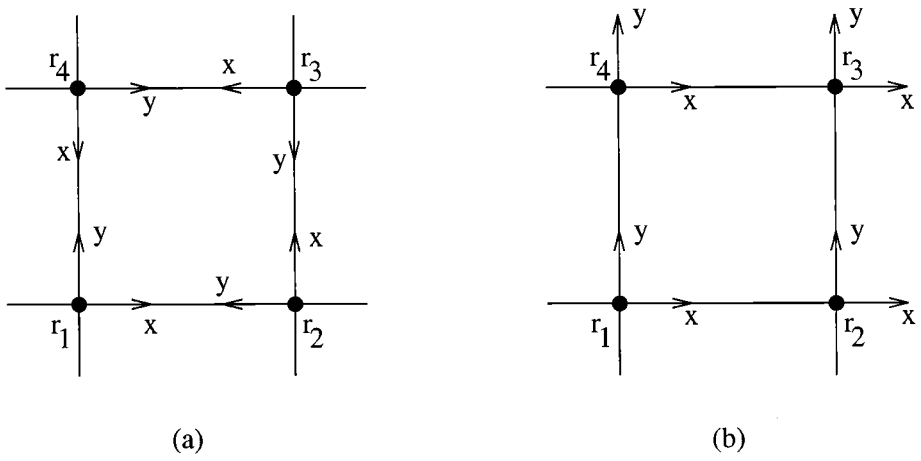


FIG. 4.1. *Two configurations of four robots that are* (a) *symmetric,* (b) *not symmetric.*

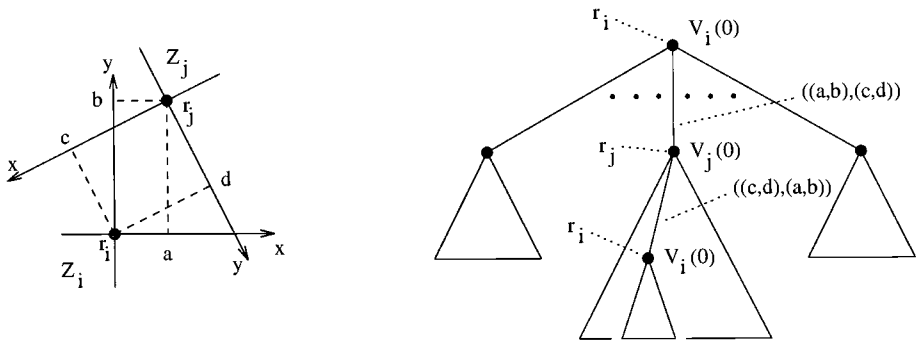


FIG. 4.2. r_i, r_j, *and views* $V_i(0)$ *and* $V_j(0)$.

of Z_2, and so on, as shown in Figure 4.1(a). Intuitively, the robots have the same "view," and thus, if they are synchronized, then they will never be able to break symmetry and form a pattern other than a square, but if the direction of the positive x-axis happens to be the same for all four robots, as shown in Figure. 4.1(b), then intuitively every robot has a unique "view," and hence the robots may be able break symmetry and form a pattern that is not a square. (In fact, the result given below shows that the robots can form *any* pattern for this case.) We now formalize this observation.

Following [19], [20], the *view* of robot r_i at time t, denoted $V_i(t)$, is defined recursively as a rooted infinite tree as follows. See Figure 4.2.

1. The root of $V_i(t)$ has $n-1$ subtrees, one for each robot r_j, $j \neq i$.
2. The edge from the root of $V_i(t)$ to the subtree corresponding to r_j is labeled $((a,b),(c,d))$, where (a,b) is the position of r_j in terms of Z_i and (c,d) is the position of r_i in terms of Z_j.
3. The subtree corresponding to r_j is the view $V_j(t)$ of r_j at time t.

Note that each vertex of $V_i(t)$ corresponds to a robot, but it is not labeled as such. Two views $V_i(t)$ and $V_j(t')$ are said to be *equivalent*, written $V_i(t) \equiv V_j(t')$, if they are isomorphic to each other, including the labels. A view is defined as an infinite tree

for convenience of discussion; the relevant information is contained in the subtree of height 2 from the root.

$V_i(0)$ is thus the view of r_i at time 0. Note that since the robots occupy distinct positions at time 0, the edges incident on the root of $V_i(0)$ have distinct labels. Since at time 0 the robots have no knowledge of other robots' local coordinate systems, at time 0 robot r_i does not know its view $V_i(0)$. Using the following algorithm, the robots can obtain sufficient information to construct their views at time 0.

ALGORITHM $\psi_{getview}$—NONOBLIVIOUS.

The robots first broadcast the x-axes of their respective local coordinate systems by moving in the respective positive x directions, return straight to their respective initial positions, broadcast the y-axes of their respective local coordinate systems by moving in the respective positive y directions, and finally return straight to their respective initial positions. Since different robots may start the second broadcast (of their local y-axes) at different time instants, every robot r_i broadcasting its y-axis must continue to move along its y-axis until it observes that every r_j, $j \neq i$ has changed positions at least twice along a line perpendicular to the first line that r_j broadcasted.

At this moment every robot r_i has discovered the initial distribution $P(0)$ (in terms of Z_i) as well as the direction of the positive x-axis of Z_j for every robot r_j. Then r_i measures the minimum distance d_i between any two robots in $P(0)$ in terms of Z_i and "announces" the value of d_i to all other robots by broadcasting the directed line through its initial position with direction $f(d_i)$ of Z_i, where for $x > 0$, $f(x) = (1 - 1/2^x) \times 360^\circ$ is a monotonically increasing function with range $(0^\circ, 360^\circ)$. Then, any robot observing the movement of r_i can determine the value of d_i (and hence the unit distance of Z_i) from its knowledge on the positive x direction of Z_i and direction $f(d_i)$ of Z_i. Finally, the robots return to their respective initial positions. □

When $\psi_{getview}$ is completed, each robot r_i can determine the positions of all other robots in terms of Z_j for any j. Using this information, r_i can construct its view $V_i(0)$.

Let m be the size of a largest subset of robots having an equivalent view at time 0. If $m = 1$, then every robot has a unique view, and thus once Algorithm $\psi_{getview}$ is executed the robots can be ordered using a suitable total ordering of the views. Then for any multiset F of n points, using a predetermined total ordering of the points in F, the ith robot in the ordering can compute the location of the ith point in F relative to some reference points (e.g., the positions of the first and second robots at time 0 if the first and second points of F are distinct) and move to that point. Therefore, if $m = 1$, the robots can form a pattern similar to F for arbitrary F.

Therefore, in the following, we consider the case $m \geq 2$. Lemmas 4.1, 4.2, 4.3, and 4.4 refer to a fixed initial configuration with $m \geq 2$.

LEMMA 4.1. *The robots can be partitioned into n/m groups of m robots each, such that two robots have an equivalent view iff they belong to the same group.*

Proof. The claim is trivial if $m = n$. Thus assume that $m < n$, and without loss of generality suppose that $V_1(0) \equiv V_2(0) \equiv \cdots \equiv V_m(0)$ but $V_1(0) \not\equiv V_{m+1}(0)$. That is, $r_1, r_2, \ldots, r_m$ have an equivalent view at time 0 but r_{m+1} does not. Let $((a, b), (c, d))$ be the label of the edge from the root of $V_1(0)$ to the vertex corresponding to r_{m+1}. Since $V_1(0) \equiv V_2(0) \equiv \cdots \equiv V_m(0)$ for each ℓ, $1 \leq \ell \leq m$, there exists an edge with label $((a, b), (c, d))$ from the root of $V_\ell(0)$ to a vertex corresponding to some robot r_{i_ℓ}, where $r_{i_1} = r_{m+1}$. Now we show that the robots $r_{i_1}, r_{i_2}, \ldots, r_{i_m}$ are all distinct. Note that by symmetry there is an edge with label $((c, d), (a, b))$ from the

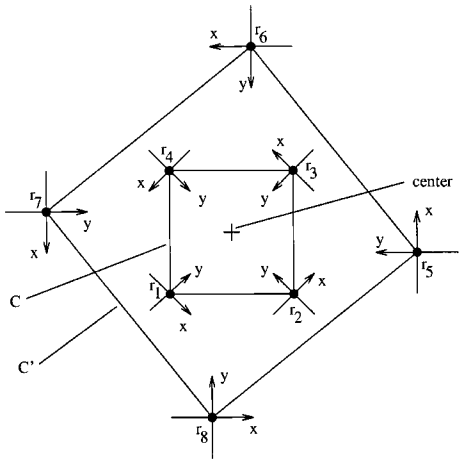


FIG. 4.3. *Illustration for the proof of Lemma* 4.2*, for the case* $m = 4$.

root of $V_{i_\ell}(0)$, leading to a vertex that corresponds to robot r_ℓ. Thus if $r_{i_1} = r_{i_2}$, for instance, then we have $r_1 = r_2$, a contradiction. Thus $r_{i_1}, r_{i_2}, \ldots, r_{i_m}$ are all distinct. Furthermore, since $V_1(0) \equiv V_2(0) \equiv \cdots \equiv V_m(0)$ and $V_{i_\ell}(0)$ is a subtree of $V_\ell(0)$ connected to the root of $V_\ell(0)$ by an edge with label $((a,b),(c,d))$ for each ℓ, we have $V_{i_1}(0) \equiv V_{i_2}(0) \equiv \cdots \equiv V_{i_m}(0)$. Thus there are at least m robots (including r_{m+1}) having a view equivalent to that of r_{m+1}. But then there must be exactly m such robots, since there cannot exist more than m such robots by the definition of m. The lemma follows from this observation. □

LEMMA 4.2. *At time* 0*, the robots in the same group form a regular* m*-gon, and the regular* m*-gons formed by all the groups have a common center.*[7] *(See Figure* 4.3*.)*

Proof. Suppose that $V_1(0) \equiv V_2(0) \equiv \cdots \equiv V_m(0)$, that is, $r_1, r_2, \ldots, r_m$ have an equivalent view at time 0. Consider the initial positions $p_1(0), p_2(0), \ldots, p_m(0)$ of these robots. Clearly, at least one of $p_1(0), p_2(0), \ldots, p_m(0)$ is a corner of the convex hull C of $\{p_1(0), p_2(0), \ldots, p_m(0)\}$. Then, since $V_1(0) \equiv V_2(0) \equiv \cdots \equiv V_m(0)$, each of $p_1(0), p_2(0), \ldots, p_m(0)$ must be a corner of C. Without loss of generality, assume that $p_1(0), p_2(0), \ldots, p_m(0)$ occur in counterclockwise order around the convex hull. (See Figure 4.3.)

Since $V_1(0) \equiv V_2(0) \equiv \cdots \equiv V_m(0)$, the internal angles of C at the corners $p_1(0), p_2(0), \ldots, p_m(0)$ must all be identical, and the lengths of the edges of the convex hull must all be identical. (If $\overline{p_1(0)p_2(0)}$ looks shorter than $\overline{p_2(0)p_3(0)}$ to r_2, then $\overline{p_2(0)p_3(0)}$ should look shorter than $\overline{p_3(0)p_4(0)}$ to r_3, and so on, leading to a conclusion that $\overline{p_1(0)p_2(0)}$ is shorter than $\overline{p_1(0)p_2(0)}$, a contradiction.) Thus $p_1(0), p_2(0), \ldots, p_m(0)$ form a regular m-gon.

Suppose that at time 0, $r_{m+1}, r_{m+2}, \ldots, r_{2m}$ also have an equivalent view and that their respective positions $p_{m+1}(0), p_{m+2}(0), \ldots, p_{2m}(0)$ appear in counterclockwise order around the regular m-gon they form. Then again, since $V_1(0) \equiv V_2(0) \equiv \cdots \equiv V_m(0)$, the position of $p_{m+1}(0)$ relative to p_1 is the same as the position of $p_{m+2}(0)$ relative to p_2, and so on. (See Figure 4.3.) So the regular m-gon formed by $p_1(0), p_2(0), \ldots, p_m(0)$ and the regular m-gon formed by $p_{m+1}(0), p_{m+2}(0), \ldots, p_{2m}(0)$ have the same center. □

[7] A regular 2-gon is simply a line segment whose center is the midpoint of the endpoints.

LEMMA 4.3. *For any algorithm ψ, if the robots are synchronized, then at any time instant t, the robots in the same group form a regular m-gon and the regular m-gons formed by all the groups have a common center.*

Proof. Suppose that $V_1(0) \equiv V_2(0) \equiv \cdots \equiv V_m(0)$, that is, $r_1, r_2, \ldots, r_m$ have an equivalent view at time 0. Now, since the initial distribution of the robots looks identical to $r_1, r_2, \ldots, r_m$, the new positions they compute using ψ in their respective $Z_1, Z_2, \ldots, Z_m$ are all identical. Also, since $V_1(0) \equiv V_2(0) \equiv \cdots \equiv V_m(0)$, the center of the regular m-gon that $r_1, r_2, \ldots, r_m$ form at time 0 has the same x-y coordinates in all of $Z_1, Z_2, \ldots, Z_m$. This means that $r_1, r_2, \ldots, r_m$ move in a symmetric manner relative to the center of the regular m-gon, and thus at time 1 they again form a regular m-gon with the same center. The same applies to all n/m groups, and since the robots are synchronized, at time 1 they together form a collection of n/m regular m-gons all having the same center. Since the robots in the same group have observed the same robot distributions, their next moves at time 1 are also symmetric relative to the center of the regular m-gon they currently form. Therefore, again, at time 2 the robots form a collection of n/m regular m-gons all having the same center. Continuing in the same manner, we can prove that at any time instant t the robots form a collection of n/m regular m-gons all having the same center. □

Since the robots may happen to be synchronized, by Lemma 4.3 there exists an algorithm ψ for forming a pattern similar to F starting from the given initial configuration only if F can be partitioned into n/m regular m-gons all having the same center. Conversely, we have the next lemma.

LEMMA 4.4. *For any multiset F of points that can be partitioned into n/m regular m-gons all having the same center, there exists an algorithm ψ for forming a pattern similar to F starting from the initial configuration. (The algorithm does not depend on the initial configuration.)*

Proof. We fix a total ordering over views and we fix an ordering of the n/m regular m-gons in F. The idea is to move the robots in the jth group in the ordering of the views to the corners of the jth regular m-gon, as in the case $m = 1$. Specifically, first the robots execute Algorithm $\psi_{getview}$ and obtain their views. The robots in the first group need not move any more, since the m-gon they form is similar to the corners of the first m-gon of F (except when the first m-gon is a point, in which case the robots must move to the center of the m-gon they form). Each robot in the second group computes the position of a corner of the second m-gon of F (relative to the location of the first m-gon of F) that is closest to its current position, breaking ties in any deterministic manner. (If the first m-gon is a point and the second m-gon is not, then the robots in the second group need not move.) The robots in other groups also compute their final positions in a similar manner. Then the robots move to their respective final positions and form a pattern similar to F. □

The following theorem summarizes the discussion given above.

THEOREM 4.5. *Let m be the size of a largest subset of robots having an equivalent view at time 0. Let F be a multiset of n points. There exists an algorithm ψ for forming a pattern similar to F, starting from the given initial configuration iff either (1) $m = 1$ or (2) $m \geq 2$ and F can be partitioned into n/m regular m-gons all having the same center.*

Proof. The theorem follows from Lemmas 4.3 and 4.4. □

We introduced in section 3 a predicate π_{point} such that $\pi_{point}(p_1, \ldots, p_n) =$ true iff $p_i = p_j$ for any $1 \leq i, j \leq n$. Consider another predicate $\pi_{regular}$, where $\pi_{regular}(p_1, \ldots, p_n) =$ true iff $p_1, \ldots, p_n$ form a regular n-gon. The following theo-

rem, which follows as a corollary to Theorem 4.5, states that these two are the only predicates for which the formation problem is solvable.

THEOREM 4.6. *There exists an algorithm for solving the formation problem for a predicate π iff either $\pi = \pi_{point}$ or $\pi = \pi_{regular}$.*

Proof. The if part for $\pi_{regular}$ follows immediately from Theorem 4.5, and that for π_{point} follows from Theorem 4.5 and the observation that, for any m that divides n, a point can be viewed as a collection of n/m degenerate regular m-gons all having the same center. The only-if part follows from the fact that if $m = n$, where m is the size of a largest subset of robots having an equivalent view at time 0, then by Theorem 4.5 an algorithm exists for the formation problem only for a single regular n-gon (which reduces to a point if the polygon is degenerate). □

5. Concluding remarks. We formally modeled the system of anonymous mobile robots and characterized the class of geometric patterns that the robots can form. In this section, we discuss other related issues.

5.1. Agreement on a common *x*-*y* coordinate system. In section 1 we briefly mentioned that the agreement problem on a common x-y coordinate system is reducible to the formation problem of certain geometric patterns. By Theorem 4.6 it is always possible for the robots to form a point and a regular n-gon, hence the robots can always agree on both the origin and unit distance (of a common x-y coordinate system). On the one hand, the agreement problem on direction is unsolvable in general, since otherwise the formation problem of a line segment would be solvable, contradicting Theorem 4.6. On the other hand, it can be shown that if the robots have a sense of direction (i.e., their local coordinate systems agree on the positive x direction), then they have distinct views at time 0 (i.e., $m = 1$ where m is as defined in section 4). As we have shown, in this case the robots can form (a pattern similar to) any geometric pattern. This means that the difficulty of forming certain geometric patterns lies in the difficulty of agreeing on direction (and break symmetry).

5.2. Issues of fault tolerance. As we mentioned in section 1, Debest [2] discussed the problem of forming a circle from the viewpoint of self-stabilizing systems. Algorithms for controlling robots must be sufficiently robust against sensor and control errors. Oblivious algorithms are, by definition, self-stabilizing in the sense that they achieve their goal even in the presence of a finite number of sensor and control errors. In contrast, nonoblivious algorithms are sensitive to errors in general, and it is a challenging open problem to enhance fault tolerance in such algorithms.

Another interesting issue in fault tolerance arises when the number of robots changes dynamically a finite number of times during the execution of an algorithm, where by this we mean that a robot becomes visible (or invisible) when it is added to (or removed from) the system. Again by definition, an oblivious algorithm correctly solves the given problem even if the number of robots changes a finite number of times. One way to make nonoblivious algorithms robust against such changes is to adopt an additional assumption that, if the number of robots changes, then it never changes again until all robots have noticed the change. Under this assumption, it can be shown that any nonoblivious algorithm works correctly when it is modified so that a robot noticing a change in the number of robots "resets its memory and restarts the algorithm" (i.e., it ignores the pairs (Q_ℓ, p_ℓ) for the observations made previously).

5.3. Time complexity. Since a robot may remain inactive for an unpredictable period of time, we cannot use the total number of steps for measuring the time complexity of a formation algorithm. An alternative measure of the complexity of

an algorithm is the total distance that a robot must move to form a given pattern. Under this measure, a robot moves over distance $O(d)$ by the method used in the proof of Lemma 4.4, where d is the diameter of the smallest enclosing circle of the initial positions of the robots. (Note that the total distance that a robot moves while executing $\psi_{getview}$ can be limited to $O(1)$.) The bound of $O(d)$ is tight for some patterns (e.g., a point), since a robot can move at most a constant distance at a time.

5.4. Other open problems. Algorithms for solving a formation problem based on the method given in the proof of Lemma 4.4 are nonoblivious. Thus Theorem 4.6 implies that a point and a regular n-gon can be formed by n robots regardless of the initial distribution $P(0)$ and the activation schedule $\mathcal{A}$, by a *nonoblivious* algorithm. An interesting question is whether these patterns can also be formed, regardless of $P(0)$ and $\mathcal{A}$, by an *oblivious* algorithm. For the case of a point we already have the answer: an oblivious algorithm for forming a point exists for the case $n \geq 3$ (Theorem 3.4), but not for the case $n = 2$ (Theorem 3.1). However, the question remains open for the formation of a regular n-gon. We are currently working on this issue and also are conducting similar investigations on (1) randomized algorithms, (2) the case in which the motion of a robot is not instantaneous, and (3) the three-dimensional case.

REFERENCES

[1] G. Beni, S. Hackwood, and X. Liu, *High-order strictly local swarms*, in Distributed Autonomous Robotic Systems, Asama, Fukuda, Arai, and Endo, eds., Springer-Verlag, New York, 1994, pp. 267–278.

[2] X. A. Debest, *Remark about self-stabilizing systems*, Communications of the ACM, 38 (1995), pp. 115–117.

[3] B. R. Donald, *Information invariants in robotics: Part* I*—State, communication, and side-effects*, in Proceedings of the IEEE International Conference on Robotics and Automation, Atlanta, GA, 1993, pp. 276–283.

[4] B. R. Donald, *Information invariants in robotics: Part* II*—Sensors and computation*, in Proceedings of the IEEE International Conference on Robotics and Automation, Atlanta, GA, 1993, pp. 284–290.

[5] K. Fujimura, *Model of reactive planning for multiple mobile agents*, in Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, CA, 1991, pp. 1503–1509.

[6] T. Fukuda and S. Nakagawa, *Approach to the dynamically reconfigurable robot systems*, Journal of Intelligent and Robotics Systems, 1 (1988), pp. 55–72.

[7] Y. Kawauchi, M. Inaba, and T. Fukuda, *A principle of decision making of cellular robotic system (CEBOT)*, in Proceedings of the IEEE International Conference on Robotics and Automation, Los Alamitos, CA, 1993, pp. 833–838.

[8] S. Kokaji, *A fractal mechanism and a decentralized control method*, in Proceedings of the USA-Japan Symposium on Flexible Automation, Minneapolis, MN, 1988, pp. 1129–1134.

[9] M. J. Mataric, *Designing emergent behaviors: From local interactions to collective intelligence*, in From Animals to Animates 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior, Meyer, Roitblat, and Wilson, eds., MIT Press, Cambridge, MA, 1993, pp. 432–441.

[10] S. Murata, H. Kurokawa and S. Kokaji, *Self-assembling machine*, in Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, CA, 1994, pp. 441–448.

[11] L. E. Parker, *Designing control laws for cooperative agent teams*, in Proceedings of the IEEE International Conference on Robotics and Automation, Los Alamitos, CA, 1993, pp. 582–587.

[12] M. Schneider, *Self–stabilization*, ACM Computing Surveys, 25 (1993), pp. 45–67.

[13] K. Sugihara and I. Suzuki, *Distributed motion coordination of multiple mobile robots*, in Proceedings of the 5th IEEE International Symposium on Intelligent Control, Philadelphia, PA, 1990, pp. 138–143.

[14] K. Sugihara and I. Suzuki, *Distributed algorithms for formation of geometric patterns with many mobile robots*, Journal of Robotic Systems, 13 (1996), pp. 127–139.

[15] I. Suzuki and M. Yamashita, *Formation and agreement problems for anonymous mobile robots*, in Proceedings of the 31st Annual Allerton Conference on Communication, Control, and Computing, University of Illinois, Urbana, IL, 1993, pp. 93–102.

[16] O. Tanaka, *Forming a Circle by Distributed Anonymous Mobile Robots*, Bachelor thesis, Department of Electrical Engineering, Hiroshima University, Hiroshima, Japan, 1992.

[17] J. Wang and G. Beni, *Cellular robotic systems: Self-organizing robots and kinetic pattern generation*, in Proceedings of the 1988 IEEE International Workshop on Intelligent Robots and Systems, Tokyo, Japan, 1988, pp. 139–144.

[18] L. L. Whitcomb, D. E. Koditschek, and J. B. D. Cabrera, *Toward the automatic control of robot assembly tasks via potential functions: The case of 2-D sphere assemblies*, in Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France, 1992, pp. 2186–2191.

[19] M. Yamashita and T. Kameda, *Computing on anonymous networks Part* I*: Characterizing the solvable cases*, IEEE Transactions on Parallel and Distributed Systems, 7 (1996), pp. 69–89.

[20] M. Yamashita and T. Kameda, *Computing on anonymous networks Part* II*: Decision and membership problems*, IEEE Transactions on Parallel and Distributed Systems, 7 (1996), pp. 90–96.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1364–1396

A PSEUDORANDOM GENERATOR FROM ANY ONE-WAY FUNCTION*

JOHAN HÅSTAD†, RUSSELL IMPAGLIAZZO‡, LEONID A. LEVIN§, AND MICHAEL LUBY¶

Abstract. Pseudorandom generators are fundamental to many theoretical and applied aspects of computing. We show how to construct a pseudorandom generator from *any* one-way function. Since it is easy to construct a one-way function from a pseudorandom generator, this result shows that there is a pseudorandom generator if and only if there is a one-way function.

Key words. one-way function, pseudorandom generator, cryptography, complexity theory

AMS subject classifications. 68P25, 68Q25, 68Q99

PII. S0097539793244708

1. Introduction. One of the basic primitives in the study of the interaction between randomness and feasible computation is a pseudorandom generator. Intuitively, a *pseudorandom generator* is a polynomial time-computable function g that stretches a short random string x into a long string $g(x)$ that "looks" random to any feasible algorithm, called an adversary. The adversary tries to distinguish the string $g(x)$ from a random string the same length as $g(x)$. The two strings "look" the same to the adversary if the acceptance probability for both strings is essentially the same. Thus, a pseudorandom generator can be used to efficiently convert a small amount of true randomness into a much larger number of effectively random bits.

The notion of randomness tests for a string evolved over time: from set-theoretic tests to enumerable [K65], recursive, and finally limited time tests. Motivated by cryptographic applications, the seminal paper [BM82] introduced the idea of a generator which produces its output in polynomial time such that its output passes a general polynomial time test. The fundamental paper [Yao82] introduced the definition of a pseudorandom generator most commonly used today and proves that this definition and the original of [BM82] are equivalent.

The robust notion of a pseudorandom generator, due to [BM82], [Yao82], should be contrasted with the classical methods of generating random looking bits as described in, e.g., [Knuth97]. In studies of classical methods, the output of the generator is considered good if it passes a particular set of standard statistical tests. The linear congruential generator is an example of a classical method for generating random looking bits that pass a variety of standard statistical tests. However, [Boyar89] and [K92] show that there is a polynomial time statistical test which the output from this

*Received by the editors February 22, 1993; accepted for publication (in revised form) August 18, 1997; published electronically April 7, 1999.
http://www.siam.org/journals/sicomp/28-4/24470.html

†Department of Numerical Analysis and Computer Science, Royal Institute of Technology, S-100 44 Stockholm 70, Sweden (Johanh@nada.kth.se). This research was supported by the Swedish National Board for Technical Development.

‡Department of Computer Science, University of California at San Diego, La Jolla, CA 92093 (Russell@cs.ucsd.edu). This research was supported by NSF grant CCR 88-13632.

§Computer Science Department, Boston University, 111 Cummington St., Boston, MA 02215 (Lnd@cs.bu.edu). This research was supported by NSF grants CCR-9015276 and CCR-9610455.

¶International Computer Science Institute, University of California at Berkeley, 1947 Center Street, Berkeley, CA 94704 (Luby@icsi.berkeley.edu). This research was supported by NSERC grant A8092 and NSF grants CCR-9016468 and CCR-9304722.

generator does not pass.

The distinction between the weaker requirement that the output pass some particular statistical tests and the stronger requirement that it pass all feasible tests is particularly important in the context of many applications. As pointed out by [BM82], in cryptographic applications the adversary must be assumed to be as malicious as possible, with the only restriction on tests being computation time. A pseudorandom generator can be directly used to design a private key cryptosystem secure against all such adversaries.

In the context of Monte Carlo simulation applications, a typical algorithm uses long random strings, and a typical analysis shows that the algorithm produces a correct answer with high probability if the string it uses is chosen uniformly. In practice, the long random string is not chosen uniformly, as this would require more random bits than it is typically reasonable to produce (and store). Instead, a short random string is stretched into a long string using a simple generator such as a linear congruential generator, and this long string is used by the simulation algorithm. In general, it is hard to directly analyze the simulation algorithm to prove that it produces the correct answer with high probability when the string it uses is produced using such a method. A pseudorandom generator provides a generic solution to this problem. For example, [Yao82] shows how pseudorandom generators can be used to reduce the number of random bits needed for any probabilistic polynomial time algorithm and thus shows how to perform a deterministic simulation of any polynomial time probabilistic algorithm in subexponential time based on a pseudorandom generator. The results on deterministic simulation were subsequently generalized in [BH89], [BFNW96].

Since the conditions are rather stringent, it is not easy to come up with a natural candidate for a pseudorandom generator. On the other hand, there seem to be a variety of natural examples of another basic primitive: the one-way function. Informally, f is *one-way* if it is easy to compute but hard on average to invert. If $P{=}NP$, then there are no one-way functions, and it is not even known if $P \neq NP$ implies there are one-way functions. However, there are many examples of functions that seem to be one-way in practice and that are conjectured to be one-way. Some examples of conjectured one-way functions are the discrete logarithm problem modulo a large randomly chosen prime (see, e.g., [DH76]), factoring a number that is the product of two large randomly chosen primes (see, e.g., [RSA78]), problems from coding theory (see, e.g., [McEl78], [GKL93]), and the subset sum problem for appropriately chosen parameters (see, e.g., [IN96]).

The paper [BM82] is the first to construct a pseudorandom generator based on a one-way function. They introduce an elegant construction that shows how to construct a pseudorandom generator based on the presumed difficulty of the discrete logarithm problem. The paper [Yao82] substantially generalizes this result by showing how to construct a pseudorandom generator from any one-way permutation. (Some of the arguments needed in the proof were missing in [Yao82] and were later completed by [Levin87]. Also, [Levin87] conjectured that a much simpler construction would work for the case of one-way permutations, and this was eventually shown in [GL89].)

There are several important works that have contributed to the expansion of the conditions on one-way functions under which a pseudorandom generator can be constructed. [GMT82] and [Yao82] show how to construct a pseudorandom generator based on the difficulty of factoring, and this was substantially simplified in [ACGS88]. When f is a one-way permutation, the task of inverting $f(x)$ is to find x. In the case when f is not a permutation, the natural extension of successful inversion is finding

any x' such that $f(x') = f(x)$. The paper [Levin87] introduces one-way functions which remain one-way after several iterations and shows them to be necessary and sufficient for the construction of a pseudorandom generator. The paper [GKL93] shows how to construct a pseudorandom generator from any one-way function with the property that each value in the range of the function has roughly the same number of preimages. This expanded the list of conjectured one-way functions from which pseudorandom generators can be constructed to a variety of nonnumber theoretic functions, including coding theory problems.

However, the general question of how to construct a pseudorandom generator from a one-way function with no structural properties was left open. This paper resolves this question. We give several successively more intricate constructions, starting with constructions for one-way functions with a lot of structure and finishing with the constructions for one-way functions with no required structural properties.

This paper is a combination of the results announced in the conference papers [ILL89] and [H90].

1.1. Concepts and tools. Previous methods, following [BM82], rely on constructing a function that has an output bit that is computationally unpredictable given the other bits of the output, but is nevertheless statistically correlated with these other bits. [GL89] provide a simple and natural input bit which is hidden from (a padded version of) any one-way function. Their result radically simplifies the previous constructions of pseudorandom generators from one-way permutations and in addition makes all previous constructions substantially more efficient. We use their result in a fundamental way.

Our overall approach is different in spirit from previous constructions of pseudorandom generators based on one-way functions with special structure. Previous methods rely on iterating the one-way function many times, and from each iteration they extract a computationally unpredictable bit. The approach is to make sure that after many iterations the function is still one-way. In contrast, as explained below in more detail, our approach concentrates on extracting and smoothing entropy in parallel from many independent copies of the one-way function. Our overall construction combines this parallel approach with a standard method for iteratively stretching the output of a pseudorandom generator.

The notion of computational indistinguishability provides one of the main conceptual tools in our paper. Following [GM84] and [Yao82], we say that two probability distributions $\mathcal{D}$ and $\mathcal{E}$ are *computationally indistinguishable* if no feasible adversary can distinguish $\mathcal{D}$ from $\mathcal{E}$. In these terms, a pseudorandom generator is intuitively the following: let g be a polynomial time computable function that maps strings of length n to longer strings of length $\ell_n > n$. Let X be a random variable that is uniformly distributed on strings of length n and let Y be a random variable that is uniformly distributed on strings of length ℓ_n. Then g is a *pseudorandom generator* if $g(X)$ and Y are computationally indistinguishable.

The Shannon entropy of a distribution is a good measure of its information content. A fundamental law of information theory is that the application of a function cannot increase entropy. For example, because X has n bits of entropy, $g(X)$ can also have at most n bits of entropy (see Proposition 2.6). The work presented in this paper focuses on a computational analogue of Shannon entropy, namely computational entropy. We say the *computational entropy* of $g(X)$ is at least the Shannon entropy of Y if $g(X)$ and Y are computationally indistinguishable. If $g(X)$ is a pseudorandom generator, the computational entropy of $g(X)$ is greater than the Shannon entropy of

its input X, and in this sense g amplifies entropy.

We introduce the following generalizations of a pseudorandom generator based on computational entropy. We say that $g(X)$ is a *pseudoentropy generator* if the computational entropy of $g(X)$ is significantly more than the Shannon entropy of X. We say that $g(X)$ is a *false-entropy generator* if the computational entropy of $g(X)$ is significantly more than the Shannon entropy of $g(X)$.

We show how to construct a false-entropy generator from any one-way function, a pseudoentropy generator from any false-entropy generator, and finally a pseudorandom generator from any pseudoentropy generator. (The presentation of these results in the paper is in reverse order.)

We use hash functions and their analysis in a fundamental way in our constructions. This approach has its roots in [GKL93]. In [GL89], it turns out that the easily computable bit that is hidden is the parity of a random subset of the input bits, i.e., the inner product of the input and a random string. This random inner product can be viewed as a hash function from many bits to one bit.

Due to its importance in such basic algorithms as primality testing, randomness has become an interesting computational resource in its own right. Recently, various studies for extracting good random bits from biased "slightly random" sources that nevertheless possess a certain amount of entropy have been made; these sources model the imperfect physical sources of randomness, such as Geiger counter noise and Zener diodes, that would have to actually be utilized in real life. (See [Blum84], [SV86], [V87], [VV85], [CG88], and [McIn87].) One of our main technical lemmas (Lemma 4.8) can be viewed as a hashing lemma which is used to manipulate entropy in various ways: it can be viewed as a method for extracting close to uniform random bits from a slightly random source using random bits as a catalyst.

1.2. Outline. An outline of the paper is as follows.

In section 2 we give notation, especially as related to probability distributions and ensembles. In section 3, we define the basic primitives used in the paper and a general notion of reduction between primitives. We spend a little more time on this than is conventional in papers on cryptography, since we want to discuss the effects of reductions on security in quantitative terms.

Section 4 introduces the basic mechanisms for finding hidden bits and manipulating entropy with hash functions. The main result of the section is a reduction from a false-entropy generator to a pseudorandom generator via a pseudoentropy generator.

In section 5, we present a construction of a pseudorandom generator from a one-way function where preimage sizes can be estimated. Although such one-way functions are very common, and so this is an important special case, the main reason for including this is to develop intuition for general one-way functions.

Section 6 presents the most technically challenging construction: that of a false-entropy generator from any one-way function. Combined with section 4, this yields the main result of the paper: the construction of a pseudorandom generator from any one-way function.

In section 7, we present a somewhat more direct and efficient construction of a pseudorandom generator from any one-way function. This section uses the ideas from sections 4, 5, and 6, but avoids some redundancy involved in combining three generic reductions. Section 8 concludes by placing our results in the context of modern cryptographic complexity.

2. Basic notation. $\mathcal{N}$ is the set of natural numbers. If S is a set, then $\sharp S$ is the number of elements in S. If S and T are sets, then $S \setminus T$ is the set consisting of

all elements in S that are not in T. If a is a number, then $|a|$ is the absolute value of a, $\lceil a \rceil$ is the smallest integer greater than or equal to a, and $\log(a)$ is the logarithm base two of a.

Let x and y be bit strings. We let $\langle x, y \rangle$ denote the sequence x followed by y, and when appropriate we also view this as the concatenation of x and y. If $x \in \{0,1\}^n$, then x_i is the ith bit of x, $x_{\{i,\ldots,j\}}$ is $\langle x_i, \ldots, x_j \rangle$, and $x \oplus y$ is $\langle x_1 \oplus y_1, \ldots, x_n \oplus y_n \rangle$.

An $m \times n$ bit matrix x is indicated by $x \in \{0,1\}^{m \times n}$. We write $x_{i,j}$ to refer to the (i,j)-entry in x. We can also view x as a sequence $x = \langle x_1, \ldots, x_m \rangle$ of m strings, each of length n, where in this case x_i is the ith row of the matrix, or we can view x as a bit string of length mn, which is the concatenation of the rows of the matrix.

The $\odot$ operation indicates matrix multiplication over GF[2]. If $x \in \{0,1\}^n$ appears to the left of $\odot$, then it is considered to be a row vector, and if it appears to the right of $\odot$, it is considered to be a column vector. Thus, if $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$, then $x \odot y = \sum_{i=1}^{n} x_i \cdot y_i \bmod 2$. More generally, if $x \in \{0,1\}^{\ell \times m}$ and $y \in \{0,1\}^{m \times n}$, then $x \odot y$ is the $\ell \times n$ bit matrix, where the (i,j)-entry is $r \odot c$, where r is the ith row of x and c is the jth column of y.

2.1. Probability notation. In general, we use capital and Greek letters to denote random variables and random events. Unless otherwise stated, all random variables are independent of all other random variables.

A distribution $\mathcal{D}$ on a finite set S assigns a probability $\mathcal{D}(x) \geq 0$ to each $x \in S$, and thus $\sum_{x \in S} \mathcal{D}(x) = 1$. We say a random variable X is distributed according to $\mathcal{D}$ on S if for all $x \in S$, $\Pr[X = x] = \mathcal{D}(x)$, and we indicate this by $X \in_{\mathcal{D}} S$. We write $\mathcal{D} : \{0,1\}^{\ell_n}$ to indicate that $\mathcal{D}$ is supported on strings of length ℓ_n. We sometimes, for convenience, blur the distinction between a random variable and its distribution. If X_1 and X_2 are random variables (that are not necessarily independent), then $(X_1 | X_2 = x_2)$ denotes the random variable that takes on value x_1 with the conditional probability $\Pr[X_1 = x_1 | X_2 = x_2] = \Pr[X_1 = x_1 \wedge X_2 = x_2] / \Pr[X_2 = x_2]$.

If f is a function mapping S to a set T, then $f(X)$ is a random variable that defines a distribution $\mathcal{E}$, where for all $y \in T$, $\mathcal{E}(y) = \sum_{x \in S, f(x) = y} \mathcal{D}(x)$. We let $f(\mathcal{D})$ indicate the distribution $\mathcal{E}$.

We let $X \in_{\mathcal{U}} S$ indicate that X is uniformly distributed in S; i.e., for all $x \in S$, $\Pr[X = x] = 1/\sharp S$. We let $\mathcal{U}_n$ indicate the uniform distribution on $\{0,1\}^n$; i.e., X is distributed according to $\mathcal{U}_n$ if $X \in_{\mathcal{U}} \{0,1\}^n$.

We sometimes want to indicate a random sample chosen from a distribution, and we do this by using the same notation as presented above for random variables except that we use lowercase letters; i.e., $x \in_{\mathcal{D}} S$ indicates that x is a fixed element of S chosen according to distribution $\mathcal{D}$.

If X is a real-valued random variable, then $\mathrm{E}[X]$ denotes the expected value X. If E is a probabilistic event, then $\Pr[E]$ denotes the probability that event E occurs.

DEFINITION 2.1 (statistical distance). *Let $\mathcal{D}$ and $\mathcal{E}$ be distributions on a set S. The* statistical distance *between $\mathcal{D}$ and $\mathcal{E}$ is*

$$\mathbf{L_1}(\mathcal{D}, \mathcal{E}) = \sum_{x \in S} \left| \Pr[\mathcal{D}(x)] - \Pr[\mathcal{E}(x)] \right| / 2.$$

PROPOSITION 2.2. *For any function f with domain S and for any pair of distributions $\mathcal{D}$ and $\mathcal{E}$ on S, $\mathbf{L_1}(f(\mathcal{D}), f(\mathcal{E})) \leq \mathbf{L_1}(\mathcal{D}, \mathcal{E})$.*

2.2. Entropy. The following definition of entropy is from [S48].

DEFINITION 2.3 (information and entropy). *Let $\mathcal{D}$ be a distribution on a set S. For each $x \in S$, define the* information *of x with respect to $\mathcal{D}$ to be $\mathbf{I}_{\mathcal{D}}(x) =$*

$-\log(\mathcal{D}(x))$. *Let* $X \in_{\mathcal{D}} S$. *The* (Shannon) entropy *of* $\mathcal{D}$ *is* $\mathbf{H}(\mathcal{D}) = \mathrm{E}[\mathbf{I}_{\mathcal{D}}(X)]$. *Let* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *be distributions on* S *that are not necessarily independent, and let* $X_1 \in_{\mathcal{D}_1} S$ *and* $X_2 \in_{\mathcal{D}_2} S$. *Then the* conditional entropy *of* $\mathcal{D}_1$ *with respect to* $\mathcal{D}_2$, $\mathbf{H}(\mathcal{D}_1|\mathcal{D}_2)$, *is* $\mathrm{E}_{x_2 \in_{\mathcal{D}_2} S}[\mathbf{H}(X_1|X_2 = x_2)]$.

We sometimes refer to the entropy $\mathbf{H}(X)$ of random variable X, which is equal to $\mathbf{H}(\mathcal{D})$. We sometimes refer to the conditional entropy $\mathbf{H}(X_1|X_2)$ of X_1 conditioned on X_2, which is equal to $\mathbf{H}(\mathcal{D}_1|\mathcal{D}_2)$.

The following variant definition of entropy is due to [Renyi70].

DEFINITION 2.4 (Renyi entropy). *Let* $\mathcal{D}$ *be a distribution on a set* S. *The* Renyi entropy *of* $\mathcal{D}$ *is* $\mathbf{H}_{\mathbf{Ren}}(\mathcal{D}) = -\log(\Pr[X = Y])$, *where* $X \in_{\mathcal{D}} S$ *and* $Y \in_{\mathcal{D}} S$ *are independent.*

There are distributions that have arbitrarily large entropy but have only a couple of bits of Renyi entropy.

PROPOSITION 2.5. *For any distribution* $\mathcal{D}$, $\mathbf{H}_{\mathbf{Ren}}(\mathcal{D}) \leq \mathbf{H}(\mathcal{D})$.

We sometimes use the following proposition implicitly. This proposition shows that a function cannot increase entropy in a statistical sense.

PROPOSITION 2.6. *Let* f *be a function and let* $\mathcal{D}$ *be a distribution on the domain of* f. *Then* $\mathbf{H}(f(\mathcal{D})) \leq \mathbf{H}(\mathcal{D})$.

The following definition characterizes how much entropy is lost by the application of a function f to the uniform distribution.

DEFINITION 2.7 (degeneracy of f). *Let* $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ *and let* $X \in_{\mathcal{U}} \{0,1\}^n$. *The* degeneracy *of* f *is* $\mathbf{D}_n(f) = \mathbf{H}(X|f(X)) = \mathbf{H}(X) - \mathbf{H}(f(X))$.

2.3. Ensembles. We present all of our definitions and results in asymptotic form. Ensembles are used to make the asymptotic definitions, e.g., to define primitives such as one-way functions and pseudorandom generators, and to define the adversaries that try to break the primitives. In all cases, we use $n \in \mathcal{N}$ as the index of the ensemble and, implicitly, the definition and/or result holds for all values of $n \in \mathcal{N}$.

In our definitions of ensembles, the input and output lengths are all polynomially related. To specify this, we use the following.

DEFINITION 2.8 (polynomial parameter). *We say parameter* k_n *is a* polynomial parameter *if there is a constant* $c > 0$ *such that for all* $n \in \mathcal{N}$,

$$\frac{1}{cn^c} \leq k_n \leq cn^c.$$

We say k_n *is a* $\mathbf{P}$-time polynomial parameter *if in addition there is a constant* $c' > 0$ *such that, for all* n, k_n *is computable in time at most* $c'n^{c'}$.

In many uses of a polynomial parameter k_n, k_n is integer valued, but it is sometimes the case that k_n is real valued.

DEFINITION 2.9 (function ensemble). *We let* $f : \{0,1\}^{t_n} \to \{0,1\}^{\ell_n}$ *denote a* function ensemble, *where* t_n *and* ℓ_n *are integer-valued* $\mathbf{P}$*-time polynomial parameters and where* f *with respect to* n *is a function mapping* $\{0,1\}^{t_n}$ *to* $\{0,1\}^{\ell_n}$. *If* f *is injective, then it is a* one-to-one function ensemble. *If* f *is injective and* $\ell_n = t_n$, *then it is a* permutation ensemble. *We let* $f : \{0,1\}^{t_n} \times \{0,1\}^{\ell_n} \to \{0,1\}^{m_n}$ *denote a function ensemble with two inputs. In this case, we sometimes consider* f *as being a function of the second input for a fixed value of the first input, in which case we write* $f_x(y)$ *in place of* $f(x,y)$.

DEFINITION 2.10 ($\mathbf{P}$-time function ensemble). *We say* $f : \{0,1\}^{t_n} \times \{0,1\}^{\ell_n} \to \{0,1\}^{m_n}$ *is a* T_n-time function ensemble *if* f *is a function ensemble such that, for all* $x \in \{0,1\}^{t_n}$, *for all* $y \in \{0,1\}^{\ell_n}$, $f(x,y)$ *is computable in time* T_n. *We say* f *is a* $\mathbf{P}$-

time function ensemble *if there is a constant c such that, for all n, $T_n \leq cn^c$. We say f is a* mildly nonuniform **P**-time function ensemble *if it is a* **P***-time function ensemble except that it has an additional input* $\mathbf{a}_n$ *called the* advice*, that is, an integer-valued polynomial parameter that is not necessarily* **P***-time computable.*

These definitions generalize in a natural way to functions with more than two inputs. Sometimes we describe functions that have variable length inputs or outputs; in these cases we implicitly assume that the string is padded out with a special blank symbol to the appropriate length.

In some of our intermediate reductions, we use certain statistical quantities in order to construct our new primitive. For example, we might use an approximation of the entropy of a distribution in our construction of a pseudoentropy generator. Although in many cases these quantities are not easy to approximate, the number of different approximation values they can take on is small. This is the reason for the definition of a mildly nonuniform **P**-time function ensemble in the above definition. In all the definitions we give below, e.g., of one-way functions, false-entropy generators, pseudoentropy generators, and pseudorandom generators, there is also an analogous mildly nonuniform version. In Proposition 4.17, we show how to remove mild nonuniformity in the final construction of a pseudorandom generator.

DEFINITION 2.11 (range and preimages of a function). *Let $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ be a function ensemble. With respect to n, define*

$$\text{range}_f = \{f(x) : x \in \{0,1\}^n\}.$$

For each $y \in \text{range}_f$, define

$$\text{pre}_f(y) = \{x \in \{0,1\}^n : f(x) = y\}.$$

DEFINITION 2.12 (regular function ensemble). *We say function ensemble $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ is* σ_n-regular *if $\sharp\text{pre}_f(y) = \sigma_n$ for all $y \in \text{range}_f$.*

DEFINITION 2.13 ($\tilde{\mathbf{D}}_f$). *Let $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ be a* **P***-time function ensemble. For $z \in \text{range}_f$, define the* approximate degeneracy of z *as*

$$\tilde{\mathbf{D}}_f(z) = \left\lceil \log(\sharp\text{pre}_f(z)) \right\rceil.$$

Notice that $\tilde{\mathbf{D}}_f(z)$ is an approximation to within an additive factor of 1 of the quantity $n - \mathbf{I}_{f(X)}(z)$. Furthermore, $\mathrm{E}[\tilde{\mathbf{D}}_f(f(X))]$ is within an additive factor of 1 of the degeneracy of f. If f is a σ_n-regular function then, for each $z \in \text{range}_f$, $\tilde{\mathbf{D}}_f(z)$ is within an additive factor of 1 of $\log(\sigma_n)$, which is the degeneracy of f.

DEFINITION 2.14 (probability ensemble). *We let $\mathcal{D} : \{0,1\}^{\ell_n}$ denote a* probability ensemble*, where ℓ_n is an integer-valued* **P***-time polynomial parameter and where $\mathcal{D}$ with respect to n is a probability distribution on $\{0,1\}^{\ell_n}$.*

DEFINITION 2.15 (**P**-samplable probability ensemble). *We let $\mathcal{D} : \{0,1\}^{\ell_n}$ denote a probability ensemble that, with respect to n, is a distribution on $\{0,1\}^{\ell_n}$ that can be generated from a random string of length r_n for some r_n; i.e., there is a function ensemble $f : \{0,1\}^{r_n} \to \{0,1\}^{\ell_n}$ such that if $X \in_{\mathcal{U}} \{0,1\}^{r_n}$ then $f(X)$ has the distribution $\mathcal{D}$. We say $\mathcal{D}$ is a* T_n-samplable probability ensemble *if, for all $x \in \{0,1\}^{r_n}$, $f(x)$ is computable in time T_n. We say $\mathcal{D}$ is* **P**-samplable *if f is a* **P***-time function ensemble, and $\mathcal{D}$ is* mildly nonuniformly **P**-samplable *if f is a mildly nonuniform* **P***-time function ensemble.*

DEFINITION 2.16 (copies of functions and ensembles). *Let k_n be an integer-valued* $\mathbf{P}$*-time polynomial parameter. If $\mathcal{D} : \{0,1\}^{\ell_n}$ is a probability ensemble, then $\mathcal{D}^{k_n} : \{0,1\}^{\ell_n k_n}$ is the probability ensemble where, with respect to parameter n, $\mathcal{D}^{k_n}$ consists of the concatenation of k_n independent copies of $\mathcal{D}$. Similarly, if $f : \{0,1\}^{m_n} \to \{0,1\}^{\ell_n}$ is a function ensemble, then $f^{k_n} : \{0,1\}^{m_n k_n} \to \{0,1\}^{\ell_n k_n}$ is the function ensemble where, for $y \in \{0,1\}^{k_n \times m_n}$,*

$$f^{k_n}(y) = \langle f(y_1), \ldots, f(y_{k_n}) \rangle.$$

3. Definitions of primitives and reductions. Primitives described in this paper include one-way functions and pseudorandom generators. The primitives we describe can be used in cryptographic applications but are also useful as described in the introduction in other applications. In the definition of the primitives, we need to describe what it means for the primitive to be secure against an attack by an adversary. We first introduce adversaries and security and then describe the basic primitives that we use thereafter.

3.1. Adversaries and security. An adversary is, for example, trying to invert a one-way function or trying to distinguish the output of a pseudorandom generator from a truly random string. The time–success ratio of a particular adversary is a measure of its ability to break the cryptographic primitive. (Hereafter, we use "primitive" in place of the more cumbersome and sometimes misleading phrase "cryptographic primitive.") The security of a primitive is a lower bound on the time–success ratio of *any* adversary to break the primitive.

In the constructions of some primitives, we allow both private and public inputs. A *public input* is part of the output of the primitive and is known to the adversary at the time it tries to break the primitive. When we construct one primitive based on another, the constructed primitive often has public inputs. At first glance it could seem that these public inputs are not useful because an adversary knows them at the time it tries to break the constructed primitive. On the contrary, public inputs turn out to be quite useful. Intuitively, this is because their value is randomly chosen, and the adversary cannot a priori build into its breaking strategy a strategy for all possible values.

The *private input* to a primitive is not directly accessible to the adversary. The security parameter of a primitive is the length of its private input. This is because the private input to the primitive is what is kept secret from the adversary, and thus it makes sense to measure the success of the adversary in terms of this.

DEFINITION 3.1 (breaking adversary and security). *An* adversary *A is a function ensemble. The* time–success ratio *of A for an instance f of a primitive is defined as $\mathbf{R}_{t_n} = T_n/sp_n(A)$, where t_n is the length of the private input to f, T_n is the worst-case expected running time of A over all instances parameterized by n, and $sp_n(A)$ is the success probability of A for breaking f. In this case, we say A is an* $\mathbf{R}$-breaking *adversary for f. We say f is* $\mathbf{R}$*-secure if there is no* $\mathbf{R}$*-breaking adversary for f.*

A mildly nonuniform adversary *for a mildly nonuniform* $\mathbf{P}$*-time function ensemble f that has advice $\mathbf{a}_n$ is a function ensemble A which is given $\mathbf{a}_n$ as an additional input. The success probability and time–success ratio for a mildly nonuniform adversary is the same as for uniform adversaries.*

The definition of the success probability $sp_n(A)$ for f depends on the primitive in question; i.e., this probability is defined when the primitive is defined. Intuitively, the smaller the time–success ratio of an adversary for a primitive, the better the adversary is able to break the primitive, i.e., it uses less time and/or has a larger

success probability.

The above definitions are a refinement of definitions that appear in the literature. Previously, an adversary was considered to be breaking if it ran in polynomial time and had inverse polynomial success probability. The advantage of the definition introduced here is that it is a more precise characterization of the security of a primitive. This is important because different applications require different levels of security. For some applications polynomial security is enough (e.g., $\mathbf{R}_{t_n} = {t_n}^{10}$) and for other applications better security is crucial (e.g., $\mathbf{R}_{t_n} = 2^{\log^2(t_n)}$, or even better $\mathbf{R}_{t_n} = 2^{\sqrt{t_n}}$).

3.2. One-way function.

DEFINITION 3.2 (one-way function). *Let* $f : \{0,1\}^{t_n} \to \{0,1\}^{\ell_n}$ *be a* $\mathbf{P}$*-time function ensemble and let* $X \in_{\mathcal{U}} \{0,1\}^{t_n}$. *The* success probability *of adversary* A *for inverting* f *is*

$$sp_n(A) = \Pr[f(A(f(X))) = f(X)].$$

Then f *is an* $\mathbf{R}$*-secure one-way function if there is no* $\mathbf{R}$*-breaking adversary for* f.

A function cannot be considered to be "one-way" in any reasonable sense in case the time to invert it is smaller than the time to evaluate it in the forward direction. Thus, for example, if there is an $\mathcal{O}(t_n)$-breaking adversary for f, then it is not secure at all. On the other hand, an exhaustive adversary that tries all possible inputs to find an inverse is $t_n^{\mathcal{O}(1)} \cdot 2^{t_n}$-breaking. Thus, the range of securities that can be hoped for falls between these two extremes.

3.3. Pseudorandom generator. The following definition can be thought of as the computationally restricted adversary definition of statistical distance. The original idea is from [GM84] and [Yao82].

DEFINITION 3.3 (computationally indistinguishable). *Let* $\mathcal{D} : \{0,1\}^{\ell_n}$ *and* $\mathcal{E} : \{0,1\}^{\ell_n}$ *be probability ensembles. The* success probability *of adversary* A *for distinguishing* $\mathcal{D}$ *and* $\mathcal{E}$ *is*

$$sp_n(A) = |\Pr[A(X) = 1] - \Pr[A(Y) = 1]|,$$

where X *has distribution* $\mathcal{D}$ *and* Y *has distribution* $\mathcal{E}$. $\mathcal{D}$ *and* $\mathcal{E}$ *are* $\mathbf{R}$*-secure* computationally indistinguishable *if there is no* $\mathbf{R}$*-breaking adversary for distinguishing* $\mathcal{D}$ *and* $\mathcal{E}$.

The following alternative definition of computationally indistinguishable more accurately reflects the trade-off between the running time of the adversary and its success probability. In the alternative definition, success probability is defined as $sp'_n(A) = (sp_n(A))^2$. This is because it takes $1/sp'_n(A)$ trials in order to approximate $sp_n(A)$ to within a constant factor.

DEFINITION 3.4 (computationally indistinguishable (alternative)). *This is exactly the same as the original definition, except the* success probability *of adversary* A *is* $sp'_n(A) = (sp_n(A))^2$.

In all cases except where noted, the strength of the reduction is the same under either definition of computationally indistinguishable, and we find it easier to work with the first definition. However, there are a few places where we explicitly use the alternative definition to be able to claim the reduction is linear-preserving.

Strictly speaking, there are no private inputs in the above definition, and thus by default we use n as the security parameter. However, in a typical use of this

definition, $\mathcal{D}$ is the distribution defined by the output of a **P**-time function ensemble (and thus $\mathcal{D}$ is **P**-samplable), in which case the length of the private input to this function ensemble is the security parameter. In some circumstances, it is important that both $\mathcal{D}$ and $\mathcal{E}$ are **P**-samplable; e.g., this is the case for Proposition 4.12.

The paper [Yao82] originally gave the definition of a pseudorandom generator as below, except that we parameterize security more precisely.

DEFINITION 3.5 (pseudorandom generator). *Let $g : \{0,1\}^{t_n} \to \{0,1\}^{\ell_n}$ be a* **P***-time function ensemble where $\ell_n > t_n$. Then g is an* **R***-secure pseudorandom generator if the probability ensembles $g(\mathcal{U}_{t_n})$ and $\mathcal{U}_{\ell_n}$ are* **R***-secure computationally indistinguishable.*

The definition of a pseudorandom generator only requires the generator to stretch the input by at least one bit. The following proposition provides a general way to produce a pseudorandom generator that stretches by many bits from a pseudorandom generator that stretches by at least one bit. This proposition appears in [BH89] and is due to O. Goldreich and S. Micali.

PROPOSITION 3.6. *Suppose $g : \{0,1\}^n \to \{0,1\}^{n+1}$ is a pseudorandom generator that stretches by one bit. Define $g^{(1)}(x) = g(x)$, and inductively, for all $i \geq 1$,*

$$g^{(i+1)}(x) = \langle g(g^{(i)}(x)_{\{1,\ldots,n\}}), g^{(i)}(x)_{\{n+1,\ldots,n+i\}} \rangle.$$

Let k_n be an integer-valued **P***-time polynomial parameter. Then $g^{(k_n)}$ is a pseudorandom generator. The reduction is linear-preserving.*

In section 3.6 we give a formal definition of reduction and what it means to be linear-preserving, but intuitively it means that $g^{(k_n)}$ as a pseudorandom generator is almost as secure as pseudorandom generator g.

3.4. Pseudoentropy and false-entropy generators. The definitions in this subsection introduce new notions (interesting in their own right) which we use as intermediate steps in our constructions.

The difference between a pseudorandom generator and a pseudoentropy generator is that the output of a pseudoentropy generator doesn't have to be computationally indistinguishable from the uniform distribution; instead it must be computationally indistinguishable from some probability ensemble $\mathcal{D}$ that has more entropy than the input to the generator. Thus, a pseudoentropy generator still amplifies randomness so that the output randomness is more computationally than the input randomness, but the output randomness is no longer necessarily uniform.

DEFINITION 3.7 (computational entropy). *Let $f : \{0,1\}^{t_n} \to \{0,1\}^{\ell_n}$ be a* **P***-time function ensemble and let s_n be a polynomial parameter. Then f has* **R***-secure* computational entropy *s_n if there is a* **P***-time function ensemble $f' : \{0,1\}^{m_n} \to \{0,1\}^{\ell_n}$ such that $f(\mathcal{U}_{t_n})$ and $f'(\mathcal{U}_{m_n})$ are* **R***-secure computationally indistinguishable and $\mathbf{H}(f'(\mathcal{U}_{m_n})) \geq s_n$.*

DEFINITION 3.8 (pseudoentropy generator). *Let $f : \{0,1\}^{t_n} \to \{0,1\}^{\ell_n}$ be a* **P***-time function ensemble and let s_n be a polynomial parameter. Then f is an* **R***-secure* pseudoentropy generator *with pseudoentropy s_n if $f(\mathcal{U}_{t_n})$ has* **R***-secure computational entropy $t_n + s_n$.*

If f is a pseudorandom generator, then it is easy to see that it is also a pseudoentropy generator. This is because $f(\mathcal{U}_{t_n})$ and $\mathcal{U}_{\ell_n}$ are computationally indistinguishable and by definition of a pseudorandom generator, $\ell_n > t_n$. Consequently, $\mathbf{H}(\mathcal{U}_{\ell_n}) = \ell_n \geq t_n + 1$; i.e., f is a pseudoentropy generator with pseudoentropy at least 1.

A false-entropy generator is a further generalization of pseudoentropy generator.

A false-entropy generator doesn't necessarily amplify the input randomness; it just has the property that the output randomness is more computationally than it is statistically.

DEFINITION 3.9 (false-entropy generator). *Let $f : \{0,1\}^{t_n} \to \{0,1\}^{\ell_n}$ be a* **P***-time function ensemble and let s_n be a polynomial parameter. Then f is an* **R***-secure* false-entropy generator *with false entropy s_n if $f(\mathcal{U}_{t_n})$ has* **R***-secure computational entropy $\mathbf{H}(f(\mathcal{U}_{t_n})) + s_n$.*

Note that, in the definition of computational entropy, the function ensemble f' that is computationally indistinguishable from f is required to be **P**-time computable. This is consistent with the definition of a pseudorandom generator, where the distribution from which the pseudorandom generator is indistinguishable is the uniform distribution. There is also a nonuniform version of computational entropy, where f' is not necessarily **P**-time computable, and corresponding nonuniform versions of a pseudoentropy generator and false-entropy generator. It turns out to be easier to construct a false-entropy generator f, where f' is not necessarily **P**-time computable from a one-way function, than it is to construct a false-entropy generator f, where f' is **P**-time samplable. Using this approach and a nonuniform version of Proposition 4.12, [ILL89] describes a nonuniform reduction from a one-way function to a pseudorandom generator. However, a uniform reduction using Proposition 4.12 requires that f' be **P**-time computable. Thus, one of the main difficulties in our constructions below is to build a false-entropy generator f, where f' is **P**-time computable.

3.5. Hidden bits. In the construction of a pseudorandom generator from a one-way function, one of the key ideas is to construct from the one-way function another function which has an output bit that is computationally unpredictable from the other output bits (it is "hidden") and yet statistically somewhat predictable from the other output bits. This idea is used in the original construction of a pseudorandom generator from the discrete logarithm problem [BM82] and has been central to all such constructions since that time.

DEFINITION 3.10 (hidden bit). *Let $f : \{0,1\}^{t_n} \to \{0,1\}^{\ell_n}$ and $b : \{0,1\}^{t_n} \to \{0,1\}$ be* **P***-time function ensembles. Let $\mathcal{D} : \{0,1\}^{t_n}$ be a* **P***-samplable probability ensemble, let $X \in_{\mathcal{D}} \{0,1\}^{t_n}$, and let $\beta \in_{\mathcal{U}} \{0,1\}$. Then $b(X)$ is* **R***-secure* hidden *given $f(X)$ if $\langle f(X), b(X)\rangle$ and $\langle f(X), \beta\rangle$ are* **R***-secure computationally indistinguishable.*

3.6. Reductions. All the results presented in this paper involve a reduction from one type of primitive to another.

We make the following definitions to quantify the strength of reductions. The particular parameterization of security and the different quantitative measures of the security-preserving properties of a reduction are derived from [Luby96], [HL92].

Intuitively, a reduction constructs from a first primitive f on inputs of length t_n a second primitive $g^{(f)}$ on inputs of length t'_n. The reduction also specifies an oracle TM $M^{(\cdot)}$ such that if there is an adversary A for breaking $g^{(f)}$, then $M^{(A)}$ is an adversary for breaking f. How much security is preserved by the reduction is parameterized by $\mathcal{S}$.

DEFINITION 3.11 (reduction). *Let t_n and t'_n be polynomial parameters and let $\mathcal{S} : \mathcal{N} \times \Re^+ \to \Re^+$. An* $\mathcal{S}$-reduction *from primitive* 1 *to primitive* 2 *is a pair of oracles $g^{(\cdot)}$ and $M^{(\cdot)}$ so that the following hold:*

- *For each* **P***-time function ensemble $f : \{0,1\}^{t_n} \to \{0,1\}^{\ell_n}$ that instantiates primitive* 1*, $g^{(f)} : \{0,1\}^{t'_n} \to \{0,1\}^{\ell'_n}$ instantiates primitive* 2*.*
- *$g^{(f)}$ is a* **P***-time function ensemble, and on inputs of length t'_n, it only makes calls to f on inputs of length t_n.*
- *Suppose A is an adversary with time–success ratio $\mathbf{R}'_{t'_n}$ for $g^{(f)}$ on inputs*

of length t'_n. Define $\mathbf{R}_{t_n} = \mathcal{S}(n, \mathbf{R}'_{t'_n})$. Then $M^{(A)}$ is an adversary with time–success ratio $\mathbf{R}_{t_n}$ for f on inputs of length t_n.

To discuss the security-preserving properties of the reduction, we compare how well A breaks $g^{(f)}$ with how well $M^{(A)}$ breaks f on inputs of similar size. We say the reduction is

- linear-preserving *if* $\mathbf{R}_N = N^{\mathcal{O}(1)} \cdot \mathcal{O}(\mathbf{R}'_N)$,
- poly-preserving *if* $\mathbf{R}_N = N^{\mathcal{O}(1)} \cdot \mathbf{R}'^{\mathcal{O}(1)}_{\mathcal{O}(N)}$,
- weak-preserving *if* $\mathbf{R}_N = N^{\mathcal{O}(1)} \cdot \mathbf{R}'^{\mathcal{O}(1)}_{N^{\mathcal{O}(1)}}$.

A mildly nonuniform reduction *has the same properties except that $g^{(\cdot)}$ and $M^{(\cdot)}$ are both allowed access to an integer-valued polynomial parameter $\mathbf{a}_n$ that depends on f. The same notions of security preservation apply to mildly nonuniform reductions.*

f can always be broken in time exponential in t_n. Therefore, if $\mathbf{R}'_{t'_n} \geq 2^{t_n}$, or even $\mathbf{R}'_{t'_n} \geq 2^{t_n^{\Omega(1)}} = 2^{n^{\Omega(1)}}$ in the case of a weak-preserving reduction, $M^{(A)}$ can ignore the oracle and break f by brute force. Therefore, we can assume without loss of generality that $\mathbf{R}'_{t'_n} \leq 2^{t_n}$.

Obvious from the definition of reduction are the following propositions that say that security is preserved by reductions and that reductions can be composed.

PROPOSITION 3.12. *If $(g^{(\cdot)}, M^{(\cdot)})$ is a (mildly nonuniform) $\mathcal{S}$-reduction from primitive 1 to primitive 2 and f is a (mildly nonuniform) $\mathbf{P}$-time function ensemble that instantiates primitive 1 with security $\mathbf{R}_{t_n}$, then $g^{(f)}$ is a (mildly nonuniform) $\mathbf{P}$-time function ensemble that instantiates primitive 2 with security $\mathbf{R}'_{t'_n}$.*

PROPOSITION 3.13. *If $(g_1^{(\cdot)}, M_1^{(\cdot)})$ is a (mildly nonuniform) $\mathcal{S}_1$-reduction from primitive 1 to primitive 2, and if $(g_2^{(\cdot)}, M_2^{(\cdot)})$ is a (mildly nonuniform) $\mathcal{S}_2$-reduction from primitive 2 to primitive 3, then $(g_1^{(g_2^{(\cdot)})}, M_1^{(M_2^{(\cdot)})})$ is a (mildly nonuniform) $\mathcal{S}$-reduction from primitive 1 to primitive 3, where $\mathcal{S}(N, R) = \mathcal{S}_2(N, \mathcal{S}_1(N, R))$.*

Although we phrase our definitions in terms of asymptotic complexity, one can easily interpret them for fixed length inputs in the context of an actual implementation, just as one does for algorithm analysis.

Clearly, in standard situations, $t'_n \geq t_n$ and $\mathbf{R}_{t_n} \geq \mathbf{R}'_{t'_n}$, and the closer these two inequalities are to equalities the more the security of f is transferred to g. We now describe how the slack in these inequalities affects the security-preserving properties of the reduction.

The number of calls $M^{(A)}$ makes to A is invariably either a constant or depends polynomially on the time–success ratio of A, and thus $\mathbf{R}_{t_n}$ is at most polynomial in $\mathbf{R}'_{t'_n}$. The slackness in this inequality turns out not to be the major reason for a loss in security in the reduction; instead the loss primarily depends on how much larger t'_n is than t_n. If t'_n is much larger than t_n, then $\mathbf{R}_{t_n}$ is much larger as a function of t_n than $\mathbf{R}'_{t'_n}$ is as a function of t'_n. We can formalize this as follows.

PROPOSITION 3.14.

- *If $t'_n = t_n$, $M^{(A)}$ runs in time polynomial in n (not counting the running time of A), and $sp_n(M^{(A)}) = sp_n(A)/n^{\mathcal{O}(1)}$, then the reduction is linear-preserving.*
- *If $t'_n = \mathcal{O}(t_n)$, $M^{(A)}$ runs in time polynomial in $\mathbf{R}'_{t'_n}$, and $sp_n(M^{(A)}) = sp_n^{\mathcal{O}(1)}(A)/n^{\mathcal{O}(1)}$, then the reduction is poly-preserving.*
- *If $t'_n = t_n^{\mathcal{O}(1)}$, $M^{(A)}$ runs in time polynomial in $\mathbf{R}'_{t'_n}$, and $sp_n(M^{(A)}) = sp_n^{\mathcal{O}(1)}(A)/n^{\mathcal{O}(1)}$, then the reduction is weak-preserving.*

It is important to design the strongest reduction possible. The techniques described in this paper can be directly used to yield poly-preserving reductions from regular or nearly regular (with polynomial time computable degree of regularity) one-way functions to pseudorandom generators [Luby96], and this covers almost all the conjectured one-way functions. However, the reduction for general one-way functions is only weak-preserving.

4. Hidden bits, hash functions, and computational entropy.

4.1. Constructing a hidden bit. How do we go about constructing a function such that one of its output bits is computationally unpredictable yet statistically correlated with its other output bits? The following fundamental proposition of [GL89] (strengthened in [Levin93]) provides the answer.

PROPOSITION 4.1. *Let $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ be a one-way function. Then $X \odot R$ is hidden given $\langle f(X), R\rangle$, where $X, R \in_{\mathcal{U}} \{0,1\}^n$. The reduction is linear-preserving with respect to the alternative definition of computationally indistinguishable.*

Proposition 4.1 presents an elegant, simple, and general method of obtaining a hidden bit from a one-way function. We need the following stronger proposition of [GL89] (see also [Levin93]) in some of our proofs.

PROPOSITION 4.2. *There is an oracle TM M with the following properties: Let A be any adversary that accepts as input n bits and outputs a single bit. Then $M^{(A)}$ on input parameter $\delta_n > 0$ outputs a list $\mathcal{L}$ of n-bit strings with the following property: For any fixed $x \in \{0,1\}^n$, if it is the case that*

$$|\Pr[A(R) = x \odot R] - \Pr[A(R) \neq x \odot R]| \geq \delta_n,$$

where $R \in_{\mathcal{U}} \{0,1\}^n$, then, with probability at least $1/2$, it is the case that $x \in \mathcal{L}$. (The probability here depends only on the values of the random bits used by $M^{(A)}$.) The running time of $M^{(A)}$ is polynomial in n, $1/\delta_n$, and the running time of A. Also, the number of n-bit strings in $\mathcal{L}$ is bounded by $\mathcal{O}(1/\delta_n^2)$.

The following proposition is an immediate consequence of Proposition 4.1 and Definition 3.10.

PROPOSITION 4.3. *Let $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ be a one-way function. Then $\langle f(X), R, X \odot R\rangle$ and $\langle f(X), R, \beta\rangle$ are computationally indistinguishable, where $X, R \in_{\mathcal{U}} \{0,1\}^n$ and $\beta \in_{\mathcal{U}} \{0,1\}$. The reduction is linear-preserving with respect to the alternative definition of computationally indistinguishable.*

4.2. One-way permutation to a pseudorandom generator. We describe a way to construct a pseudorandom generator from any one-way permutation which is substantially simpler (and has stronger security-preserving properties) than the original construction of [Yao82]. The construction and proof described here is due to [GL89].

PROPOSITION 4.4. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation. Let $x, r \in \{0,1\}^n$ and define $\mathbf{P}$-time function ensemble $g(x,r) = \langle f(x), r, x \odot r\rangle$. Then g is a pseudorandom generator. The reduction is linear-preserving with respect to the alternative definition of computationally indistinguishable.*

Proof. Let $X, R \in_{\mathcal{U}} \{0,1\}^n$, and $\beta \in_{\mathcal{U}} \{0,1\}$. Because f is a permutation, $\langle f(X), R, \beta\rangle$ is the uniform distribution on $\{0,1\}^{2n+1}$. By Proposition 4.3, $g(X,R)$ and $\langle f(X), R, \beta\rangle$ are computationally indistinguishable, where the reduction is linear-preserving with respect to the alternative definition of computationally indistinguishable. ∎

Proposition 4.4 works when f is a permutation because

(1) $f(X)$ is uniformly distributed and hence already looks random;
(2) for any $x \in \{0,1\}^n$, $f(x)$ uniquely determines x. So no entropy is lost by the application of f.

For a general one-way function neither (1) nor (2) necessarily holds. Intuitively, the rest of the paper constructs a one-way function with properties (1) and (2) from a general one-way function. This is done by using hash functions to smooth the entropy of $f(X)$ to make it more uniform and to recapture the entropy of X lost by the application of $f(X)$.

Proposition 4.4 produces a pseudorandom generator that only stretches the input by one bit. To construct a pseudorandom generator that stretches by many bits, combine this with the construction described previously in Proposition 3.6.

4.3. One-to-one one-way function to a pseudoentropy generator. We now describe a construction of a pseudoentropy generator from any one-to-one one-way function. This construction, together with Theorem 4.14, yields a pseudorandom generator from any one-to-one one-way function. The overall construction is different in spirit than the original construction of [GKL93]: it illustrates how to construct a pseudoentropy generator in a particularly simple way using [GL89]. Although the assumptions and the consequences are somewhat different, the construction is the same as described in Proposition 4.4.

PROPOSITION 4.5. *Let $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ be a one-to-one one-way function. Let $x, r \in \{0,1\}^n$ and define $\mathbf{P}$-time function ensemble $g(x,r) = \langle f(x), r, x \odot r\rangle$. Then g is a pseudoentropy generator$\tilde{}$with pseudoentropy 1. The reduction is linear-preserving with respect to the alternative definition of computationally indistinguishable.*

Proof. Let $X, R \in_{\mathcal{U}} \{0,1\}^n$ and $\beta \in_{\mathcal{U}} \{0,1\}$. Proposition 4.3 shows that $g(X,R)$ and $\langle f(X), R, \beta\rangle$ are computationally indistinguishable, where the reduction is linear-preserving with respect to the alternative definition of computationally indistinguishable. Because f is a one-to-one function and β is a random bit, $\mathbf{H}(f(X), R, \beta) = 2n+1$, and thus $g(X,R)$ has pseudoentropy 1. □

Note that it is not possible to argue that g is a pseudorandom generator. For example, let $f(x) = \langle 0, f'(x)\rangle$, where f' is a one-way permutation. Then f is a one-to-one one-way function and yet $g(X,R) = \langle f(X), R, X \odot R\rangle$ is not a pseudorandom generator, because the first output bit of g is zero independent of its inputs, and thus its output can easily be distinguished from a uniformly chosen random string.

4.4. Universal hash functions. The concept of a universal hash function, introduced in [CW79], has proved to have a far-reaching and broad spectrum of applications in the theory of computation.

DEFINITION 4.6 (universal hash functions). *Let $h : \{0,1\}^{\ell_n} \times \{0,1\}^n \to \{0,1\}^{m_n}$ be a $\mathbf{P}$-time function ensemble. Recall from Definition 2.9 that for fixed $y \in \{0,1\}^{\ell_n}$, we view y as describing a function $h_y(\cdot)$ that maps n bits to m_n bits. Then h is a (pairwise independent)* universal hash function *if, for all $x \in \{0,1\}^n$, $x' \in \{0,1\}^n \setminus \{x\}$, and for all $a, a' \in \{0,1\}^{m_n}$,*

$$\Pr[(h_Y(x) = a) \text{ and } (h_Y(x') = a')] = 1/2^{2m_n},$$

where $Y \in_{\mathcal{U}} \{0,1\}^{\ell_n}$.

Intuitively, a universal hash function has the property that every distinct pair x and x' are mapped randomly and independently with respect to Y.

In all of our constructions of function ensembles using universal hash functions, the description of the hash function y is viewed as a public input to the function

ensemble and thus is also part of the output. The following construction of a universal hash function is due to [CW79].

DEFINITION 4.7 (matrix construction). *Let*

$$h : \{0,1\}^{(n+1)m_n} \times \{0,1\}^n \to \{0,1\}^{m_n}$$

be the following **P***-time function ensemble: For* $x \in \{0,1\}^n$ *and* $y \in \{0,1\}^{(n+1)\times m_n}$, $h_y(x) = \langle x, 1\rangle \odot y$.

We concatenate a 1 to x in the above definition to cover the case when $x = 0^n$. Hereafter, whenever we refer to universal hash functions, one can think of the construction given above. However, any universal hash function that satisfies the required properties may be used. We note that there are more efficient hash functions in terms of the number of bits used in specification. One such example is using Toeplitz matrices (see for example [GL89] or [Levin93]). A Toeplitz matrix is a matrix that is constant on any diagonal, and thus to specify an $n \times m$ Toeplitz matrix we can specify values for the $m + n - 1$ diagonals. This is the simplest bit-efficient construction of a universal hash function, so we adopt it as the default for the remainder of the paper.

4.5. Smoothing distributions with hashing. The following lemma is a key component in most of the subsequent reductions we describe.

LEMMA 4.8. *Let* $\mathcal{D} : \{0,1\}^n$ *be a probability ensemble that has Renyi entropy at least* m_n. *Let* e_n *be a positive-integer-valued parameter. Let* $h : \{0,1\}^{\ell_n} \times \{0,1\}^n \to \{0,1\}^{m_n - 2e_n}$ *be a universal hash function. Let* $X \in_{\mathcal{D}} \{0,1\}^n$, $Y \in_{\mathcal{U}} \{0,1\}^{\ell_n}$, *and* $Z \in_{\mathcal{U}} \{0,1\}^{m_n - 2e_n}$. *Then*

$$\mathbf{L_1}(\langle h_Y(X), Y\rangle, \langle Z, Y\rangle) \le 2^{-(e_n+1)}.$$

This lemma is a generalization of a lemma that appears in [S83]. There, $\mathcal{D}$ is the uniform distribution on a set $S \subseteq \{0,1\}^n$ with $\sharp S = 2^{m_n}$. The papers [McIn87] and [BBR88] also proved similar lemmas. For the special case of linear hash functions, this lemma can be derived from [GL89] by considering unlimited adversaries. A generalization to a broader class of hash functions appears in [IZ89].

The lemma can be interpreted as follows: the universal hash function smooths out the Renyi entropy of X to the almost uniform distribution on bit strings of length almost m_n. The integer parameter e_n controls the trade-off between the uniformity of the output bits of the universal hash function and the amount of entropy lost in the smoothing process. Thus, we have managed to convert almost all the Renyi entropy of X into uniform random bits while maintaining our original supply of random bits Y.

Proof. Let $\ell = \ell_n$, $e = e_n$, $m = m_n$, and $s = m - 2e$. For all $y \in \{0,1\}^\ell$, $a \in \{0,1\}^s$, and $x \in \{0,1\}^n$, define $\mathcal{X}(h_y(x) = a) = 1$ if $h_y(x) = a$ and 0 otherwise. We want to show that

$$\mathrm{E}_Y\left[\sum_{a\in\{0,1\}^s} |\mathrm{E}_X[\mathcal{X}(h_Y(X) = a)] - 2^{-s}|\right]/2 \le 2^{-(e+1)}.$$

We show below that for all $a \in \{0,1\}^s$,

$$\mathrm{E}_Y\left[\left|\mathrm{E}_X[\mathcal{X}(h_Y(X) = a)] - 2^{-s}\right|\right] \le 2^{-(s+e)},$$

and from this the proof follows.

For any random variable Z, $\mathrm{E}[|Z|^2] \geq \mathrm{E}[|Z|]^2$ by Jensen's inequality. Letting $Z = \mathrm{E}[\mathcal{X}(h_Y(X) = a)] - 2^{-s}$, we see that it is sufficient to show for all $a \in \{0,1\}^s$,

$$\mathrm{E}_Y[(\mathrm{E}_X[\mathcal{X}(h_Y(X) = a)] - 2^{-s})^2] \leq 2^{-2(s+e)}.$$

Let $X' \in_{\mathcal{D}} \{0,1\}^n$. Using some elementary expansion of terms and rearrangements of summation, we can write the above as

$$\mathrm{E}_{X,X'}[\mathrm{E}_Y[(\mathcal{X}(h_Y(X) = a) - 2^{-s})(\mathcal{X}(h_Y(X') = a) - 2^{-s})]].$$

For each fixed value of X to x and X' to x', where $x \neq x'$, the expectation with respect to Y is zero because of the pairwise independence property of universal hash functions. For each fixed value of X to x and X' to x', where $x = x'$,

$$\mathrm{E}[(\mathcal{X}(h_Y(x) = a) - 2^{-s})^2] = 2^{-s}(1 - 2^{-s}) \leq 2^{-s}.$$

Because the Renyi entropy of $\mathcal{D}$ is at least m, it follows that $\Pr[X = X'] \leq 2^{-m}$. Thus, the entire sum is at most $2^{-(m+s)}$, which is equal to $2^{-2(s+e)}$ by the definition of s. □

In this lemma, $\mathcal{D}$ is required to have Renyi entropy at least m_n. In many of our applications, the distribution in question has at least Renyi entropy m_n, and thus the lemma applies because of Proposition 2.5. For other applications, we need to work with Shannon entropy. The following technical result due to [S48] allows us to convert Shannon entropy to Renyi entropy by looking at product distributions.

PROPOSITION 4.9. *Let k_n be an integer-valued polynomial parameter.*

- *Let $\mathcal{D} : \{0,1\}^n$ be a probability ensemble. There is a probability ensemble $\mathcal{E} : \{0,1\}^{nk_n}$ satisfying*
  - $\mathbf{H_{Ren}}(\mathcal{E}) \geq k_n \mathbf{H}(\mathcal{D}) - nk_n^{2/3}$,
  - $\mathbf{L_1}(\mathcal{E}, \mathcal{D}^{k_n}) \leq 2^{-k_n^{1/3}}$.
- *Let $\mathcal{D}_1 : \{0,1\}^n$ and $\mathcal{D}_2 : \{0,1\}^n$ be not necessarily independent probability ensembles; let $\mathcal{D} = \langle \mathcal{D}_1, \mathcal{D}_2 \rangle$. There is a probability ensemble $\mathcal{E} : \{0,1\}^{2nk_n}$, with $\mathcal{E} = \langle \mathcal{E}_1, \mathcal{E}_2 \rangle$, satisfying the following:*
  - *For every value $E_1 \in \{0,1\}^{nk_n}$ such that $\Pr_{\mathcal{E}_1}[E_1] > 0$, $\mathbf{H_{Ren}}(\mathcal{E}_2 | \mathcal{E}_1 = E_1) \geq k_n \mathbf{H}(\mathcal{D}_2|\mathcal{D}_1) - nk_n^{2/3}$.*
  - $\mathbf{L_1}(\mathcal{E}, \mathcal{D}^{k_n}) \leq 2^{-k_n^{1/3}}$.

COROLLARY 4.10. *Let k_n be an integer-valued $\mathbf{P}$-time polynomial parameter.*

- *Let $\mathcal{D} : \{0,1\}^n$ be a probability ensemble, let $m_n = k_n \mathbf{H}(\mathcal{D}) - 2nk_n^{2/3}$, and let $h : \{0,1\}^{p_n} \times \{0,1\}^{nk_n} \to \{0,1\}^{m_n}$ be a universal hash function. Let $X' \in_{\mathcal{D}^{k_n}} \{0,1\}^{k_n \times n}$ and let $Y \in_{\mathcal{U}} \{0,1\}^{p_n}$. Then*

$$\mathbf{L_1}(\langle h_Y(X'), Y \rangle, \mathcal{U}_{m_n + p_n}) \leq 2^{1 - k_n^{1/3}}.$$

- *Let $\mathcal{D}_1 : \{0,1\}^n$ and $\mathcal{D}_2 : \{0,1\}^n$ be not necessarily independent probability ensembles, and let $\mathcal{D} = \langle \mathcal{D}_1, \mathcal{D}_2 \rangle$. Let $m_n = k_n \mathbf{H}(\mathcal{D}_2|\mathcal{D}_1) - 2nk_n^{2/3}$. Let $h : \{0,1\}^{p_n} \times \{0,1\}^{nk_n} \to \{0,1\}^{m_n}$ be a universal hash function. Let $\langle X_1', X_2' \rangle \in_{\mathcal{D}^{k_n}} \{0,1\}^{k_n \times 2n}$ and let $Y \in_{\mathcal{U}} \{0,1\}^{p_n}$. Then*

$$\mathbf{L_1}(\langle h_Y(X_2'), Y, X_1' \rangle, \langle \mathcal{U}_{m_n + p_n}, X_1' \rangle) \leq 2^{1 - k_n^{1/3}}.$$

Proof. For the proof, combine Proposition 4.9, Lemma 4.8, Proposition 2.2, and Proposition 2.5. □

4.6. Pseudoentropy generator to a pseudorandom generator. Let $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ be a pseudoentropy generator with pseudoentropy s_n. In this subsection, we construct a pseudorandom generator based on f. We first start with two preliminary propositions. The following proposition is the computational analogue of Proposition 2.2.

PROPOSITION 4.11. *Let $\mathcal{D} : \{0,1\}^n$ and $\mathcal{E} : \{0,1\}^n$ be two probability ensembles and let $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ be a $\mathbf{P}$-time function ensemble. Let $\mathcal{D}$ and $\mathcal{E}$ be computationally indistinguishable. Then $f(\mathcal{D})$ and $f(\mathcal{E})$ are computationally indistinguishable. The reduction is linear-preserving.*

The following proposition first appeared in [GM84].

PROPOSITION 4.12. *Let k_n be an integer-valued $\mathbf{P}$-time polynomial parameter. Let $\mathcal{D} : \{0,1\}^{\ell_n}$ and $\mathcal{E} : \{0,1\}^{\ell_n}$ be $\mathbf{P}$-samplable probability ensembles. Let $\mathcal{D}$ and $\mathcal{E}$ be computationally indistinguishable. Then $\mathcal{D}^{k_n}$ and $\mathcal{E}^{k_n}$ are computationally indistinguishable. The reduction is weak-preserving.*

More precisely, there is a probabilistic oracle TM M with the following properties: If A is an $\mathbf{R}'_{nk_n}$-breaking adversary for distinguishing $\mathcal{D}^{k_n}$ and $\mathcal{E}^{k_n}$, then $M^{(A)}$ is an $\mathbf{R}_n$-breaking adversary for distinguishing $\mathcal{D}$ and $\mathcal{E}$, where $\mathbf{R}_n$ is essentially equal to $k_n\mathbf{R}'_{nk_n}$. It is crucial that $\mathcal{D}$ and $\mathcal{E}$ are $\mathbf{P}$-samplable because the sampling algorithms are used by M. The reduction is only weak-preserving because distinguishing $\mathcal{D}^{k_n}$ and $\mathcal{E}^{k_n}$ with respect to private inputs of length nk_n only translates into distinguishing $\mathcal{D}$ and $\mathcal{E}$ on private inputs of length n.

We now give the construction of a pseudorandom generator g from a pseudoentropy generator f.

CONSTRUCTION 4.13. *Let $f : \{0,1\}^n \to \{0,1\}^{m_n}$ be a $\mathbf{P}$-time function ensemble and let s_n be a $\mathbf{P}$-time polynomial parameter. Let $k_n = (\lceil (2m_n+1)/s_n \rceil)^3$ and $j_n = \lfloor k_n(n+s_n) - 2m_n k_n^{2/3} \rfloor$. Let $h : \{0,1\}^{p_n} \times \{0,1\}^{k_n m_n} \to \{0,1\}^{j_n}$ be a universal hash function. Let $u \in \{0,1\}^{k_n \times n}$, $y \in \{0,1\}^{p_n}$, and define $\mathbf{P}$-time function ensemble $g(u,y) = \langle h_y(f^{k_n}(u)), y\rangle$.*

THEOREM 4.14. *Let f and g be as described in Construction 4.13. Let f be a pseudoentropy generator with pseudoentropy s_n. Then g is a pseudorandom generator. The reduction is weak-preserving.*

Proof. Let $f' : \{0,1\}^{n'_n} \to \{0,1\}^{m_n}$ be the $\mathbf{P}$-time function ensemble that witnesses the pseudoentropy generator of f as guaranteed in Definition 3.7 of computational entropy; i.e., $f'(X')$ and $f(X)$ are $\mathbf{R}$-secure computationally indistinguishable and $\mathbf{H}(f'(X')) \geq n + s_n$, where $X \in_{\mathcal{U}} \{0,1\}^n$ and $X' \in_{\mathcal{U}} \{0,1\}^{n'_n}$. Let $U \in_{\mathcal{U}} \{0,1\}^{k_n \times n}$, $W \in_{\mathcal{U}} \{0,1\}^{k_n \times n'_n}$, and $Y \in_{\mathcal{U}} \{0,1\}^{p_n}$. By Proposition 4.12, $f^{k_n}(U)$ and $f'^{k_n}(W)$ are computationally indistinguishable. From Proposition 4.11, it follows that $g(U,Y) = \langle h_Y(f^{k_n}(U)), Y\rangle$ and $\langle h_Y(f'^{k_n}(W)), Y\rangle$ are computationally indistinguishable. Because $\mathbf{H}(f'(X')) \geq n + s_n$, by choice of k_n and j_n, using Corollary 4.10, it follows that $\mathbf{L}_{\mathbf{1}}(\langle h_Y(f'^{k_n}(W)), Y\rangle, \mathcal{U}_{j_n+p_n}) \leq 2^{-k_n^{1/3}}$. Thus, it follows that $g(U,Y)$ and $\mathcal{U}_{j_n+p_n}$ are computationally indistinguishable. Note that by choice of k_n, the output length $j_n + p_n$ of g is longer than its input length $nk_n + p_n$. ∎

4.7. False entropy generator to a pseudoentropy generator. Let $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ be a false-entropy generator with false entropy s_n. In this subsection, we construct a mildly nonuniform pseudoentropy generator based on f. An idea is to extract $\tilde{\mathbf{D}}_f(f(X))$ bits of entropy out of X without compromising the false entropy. (See section 2.3 for the definition of $\tilde{\mathbf{D}}_f$.) Let $X \in_{\mathcal{U}} \{0,1\}^n$. The major obstacles are that $\tilde{\mathbf{D}}_f$ is not necessarily a $\mathbf{P}$-time function ensemble and that f could

be a very nonregular function, and thus the variance of $\tilde{\mathbf{D}}_f(f(X))$ could be quite high as a function of X, and we cannot guess its value consistently with accuracy.

Let k_n be an integer-valued $\mathbf{P}$-time polynomial parameter and let $U \in_{\mathcal{U}} \{0,1\}^{k_n \times n}$. The intuition behind the following construction is that the false entropy of f^{k_n} is k_n times that of f and that the degeneracy of f^{k_n} is k_n times that of f. Furthermore, if k_n is large enough then, with high probability with respect to U, $\tilde{\mathbf{D}}_{f^{k_n}}(f^{k_n}(U))$ is close to the degeneracy of f^{k_n}. Thus we use a universal hash function h to extract roughly the degeneracy of $f^{k_n}(U)$ bits of entropy out of U without compromising the false entropy of $f^{k_n}(U)$.

CONSTRUCTION 4.15. *Let $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ be a $\mathbf{P}$-time function ensemble. Let s_n be a $\mathbf{P}$-time polynomial parameter and assume for simplicity that $s_n \leq 1$. Let $\mathbf{e}_n$ be an approximation of $\mathbf{H}(f(X))$ to within an additive factor of $s_n/8$, where $X \in_{\mathcal{U}} \{0,1\}^n$. Fix $k_n = \lceil (4n/s_n)^3 \rceil$ and $j_n = \lceil k_n(n - \mathbf{e}_n) - 2nk_n^{2/3} \rceil$. Let $h : \{0,1\}^{p_n} \times \{0,1\}^{nk_n} \to \{0,1\}^{j_n}$ be a universal hash function. For $u \in \{0,1\}^{k_n \times n}$ and $r \in \{0,1\}^{p_n}$, define $\mathbf{P}$-time function ensemble*

$$g(\mathbf{e}_n, u, r) = \langle f^{k_n}(u), h_r(u), r \rangle.$$

LEMMA 4.16. *Let f and g be as described in Construction 4.15. Let f be a false-entropy generator with false entropy s_n. Then g is a mildly nonuniform pseudoentropy generator with pseudoentropy 1. The reduction is weak-preserving.*

Proof. Let $Z \in_{\mathcal{U}} \{0,1\}^{j_n}$. First, note that $\mathbf{H}(X|f(X)) = n - \mathbf{H}(f(X)) = n - \mathbf{e}_n$. From this and Corollary 4.10 (letting $X_1 = f(X)$, $X_2 = X$ in the corollary), it follows that $\mathbf{L_1}(g(\mathbf{e}_n, U, R,), \langle f^{k_n}(U), Z, R \rangle) \leq 2^{-k_n^{1/3}}$.

We now prove that $g(\mathbf{e}_n, U, R)$ has computational entropy at least $p_n + nk_n + 1$. Let $\mathcal{D} : \{0,1\}^{\ell_n}$ be the $\mathbf{P}$-samplable probability ensemble such that $\mathcal{D}$ and $f(X)$ are computationally indistinguishable and such that

$$\mathbf{H}(\mathcal{D}) \geq \mathbf{H}(f(X)) + s_n.$$

Since $\mathcal{D}$ and $f(X)$ are computationally indistinguishable, $\langle f^{k_n}(U), Z, R \rangle$ and $\langle \mathcal{D}^{k_n}, Z, R \rangle$ are computationally indistinguishable by Proposition 4.12, which together with the first claim implies that $g(\mathbf{e}_n, U, R)$ and $\langle \mathcal{D}^{k_n}, Z, R \rangle$ are computationally indistinguishable. Now,

$$\mathbf{H}(\mathcal{D}^{k_n}, Z, R) \geq k_n \cdot (\mathbf{H}(f(X)) + s_n) + j_n + p_n \geq k_n(\mathbf{e}_n + 7s_n/8) + j_n + p_n,$$

and because by choice of k_n and j_n this is at least $p_n + nk_n + 1$. Thus g has computational entropy at least $p_n + nk_n + 1$, and the lemma follows. □

4.8. Mildly nonuniform to a uniform pseudorandom generator.

PROPOSITION 4.17. *Let $\mathbf{a}_n$ be any value in $\{0, \ldots, k_n\}$, where k_n is an integer-valued $\mathbf{P}$-time polynomial parameter. Let $g : \{0,1\}^{\lceil \log(k_n) \rceil} \times \{0,1\}^n \to \{0,1\}^{\ell_n}$ be a $\mathbf{P}$-time function ensemble, where $\ell_n > nk_n$. Let $x' \in \{0,1\}^{k_n \times n}$ and define $\mathbf{P}$-time function ensemble $g'(x') = \oplus_{i=1}^{k_n} g(i, x'_i)$. Let g be a mildly nonuniform pseudorandom generator when the first input is set to $\mathbf{a}_n$. Then g' is a pseudorandom generator. The reduction is weak-preserving.*

Proof. Let $X \in_{\mathcal{U}} \{0,1\}^n$, $X' \in_{\mathcal{U}} \{0,1\}^{k_n \times n}$, and $Z \in_{\mathcal{U}} \{0,1\}^{\ell_n}$. Suppose there is an adversary A that has distinguishing probability

$$sp_n(A) = |\Pr[A(g'(X')) = 1] - \Pr[A(Z) = 1]|.$$

We describe an oracle TM M such that, for all $i = 1, \ldots, k_n$, $M^{(A)}(i)$ has $sp_n(A)$ distinguishing probability for $g(i, X)$ and Z. For all i, the running time for $M^{(A)}(i)$ is the running time for A plus the time to compute the output of g on $k_n - 1$ inputs. This works with respect to all i, in particular when $i = \mathbf{a}_n$, from which the result follows.

For each $i = 1, \ldots, k_n$, $M^{(A)}(i)$ works as follows. On input u (and i), $M^{(A)}(i)$ randomly generates $x'_1, \ldots, x'_{i-1}, x'_{i+1}, \ldots, x'_{k_n} \in_{\mathcal{U}} \{0,1\}^n$ and computes $v = \oplus_{j \neq i} g(j, x'_j) \oplus u$. Then $M^{(A)}(i)$ runs A on input v and outputs $A(v)$. By the nature of $\oplus$, if u is chosen randomly according to Z, then $M^{(A)}(i) = 1$ with probability $\Pr[A(Z) = 1]$, whereas if u is chosen randomly according to $g(i, X)$, then $M^{(A)}(i) = 1$ with probability $\Pr[A(g'(X')) = 1]$. Thus, for each value of i, $M^{(A)}(i)$ has distinguishing probability $sp_n(A)$ for $g(i, X)$ and Z. □

Note that it may be the case that, for most fixed values of $i \in \{1, \ldots, k_n\}$, $g(i, X)$ is completely predictable. On the other hand, even if there is a value $\mathbf{a}_n$ for each n such that $g(\mathbf{a}_n, X)$ is pseudorandom, the value of $\mathbf{a}_n$ may not be **P**-time computable. This is exactly the case when the lemma is useful; i.e., it is useful to transform the mildly nonuniform pseudorandom generator g into a pseudorandom generator g'.

Note that in the given construction the length of the output of g' on inputs of length nk_n is $\ell_n > nk_n$, and thus g' stretches the input to a string of strictly greater length.

This reduction is only weak-preserving, and the reason is the usual one; i.e., the breaking adversary for $g'(X')$ on inputs of length nk_n is transferred into a breaking adversary for $g(i, X)$ on inputs of length only n.

If g in Proposition 4.17 does not satisfy the property that $\ell_n > nk_n$, then for each fixed i we can use Proposition 3.6 to stretch the output of $g(i, x)$ (viewed as a function of x) into a string of length longer than nk_n and then exclusive-or together the stretched outputs.

4.9. Summary. Putting together the results in this section, we have

- a reduction from a one-way permutation to a pseudorandom generator (from subsection 4.2);
- a reduction from a one-to-one one-way function to a pseudorandom generator(combining subsections 4.3 and 4.6);
- a reduction from a pseudoentropy generator to a pseudorandom generator (from subsection 4.6);
- a reduction from a false-entropy generator to a pseudorandom generator (combining subsections 4.7, 4.6, and 4.8).

5. Extracting entropy from one-way functions. In this section we show how to construct a pseudoentropy generator from any one-way function f with the additional property that the number of inverses of f can be computed in polynomial time; i.e., the function $\tilde{\mathbf{D}}_f$ is a **P**-time function ensemble. Combined with the results summarized in subsection 4.9, this gives a construction of a pseudorandom generator from a one-way function with this property.

One of the reasons for giving the first construction is because it illustrates some of the additional ideas needed for our construction of a false-entropy generator from any one-way function. A general one-way function f does not necessarily have the property that $\tilde{\mathbf{D}}_f$ is a **P**-time function ensemble, and considerably more effort is needed to construct a pseudorandom generator from it. In the next section, we describe how to construct a false-entropy generator from any one-way function. Combined with the

results summarized in subsection 4.9, this gives a construction of a pseudorandom generator from any one-way function.

5.1. One-way function with approximable preimage sizes to a pseudoentropy generator. To see where we get into trouble with the construction given in Proposition 4.5, suppose $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ is a $2^{n/4}$-regular one-way function, and let $X, R \in_{\mathcal{U}} \{0,1\}^n$ and $\beta \in_{\mathcal{U}} \{0,1\}$. Then, although $\langle f(X), R, X \odot R\rangle$ and $\langle f(X), R, \beta\rangle$ are computationally indistinguishable, $\mathbf{H}(f(X), R, X \odot R)$ is only about $7n/4 + 1$, and thus we have lost about $n/4$ bits of the input entropy through the application of f. Similarly, although $X \odot R$ is hidden given $\langle f(X), R\rangle$, it is also almost completely statistically uncorrelated.

The way to overcome these problems is to create a new function which is the original one-way function concatenated with the degeneracy of the function number of bits hashed out of its input to regain the lost entropy. Then Proposition 4.5 can be applied to the new function to obtain a pseudoentropy generator. We first show how to construct a pseudoentropy generator in the case when $\tilde{\mathbf{D}}_f$ is a $\mathbf{P}$-time function ensemble.

CONSTRUCTION 5.1. *Let $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ be a $\mathbf{P}$-time function ensemble and suppose that $\tilde{\mathbf{D}}_f$ is a $\mathbf{P}$-time function ensemble. Let $h : \{0,1\}^{p_n} \times \{0,1\}^n \to \{0,1\}^{n+2}$ be a universal hash function. For $x \in \{0,1\}^n$ and $y \in \{0,1\}^{p_n}$, define $\mathbf{P}$-time function ensemble*

$$f'(x,y) = \langle f(x), h_y(x)_{\{1,\ldots,\tilde{\mathbf{D}}_f(f(x))+2\}}, y\rangle.$$

LEMMA 5.2. *Let f and f' be as described in Construction* 5.1.

(1) *Let f be a one-way function. Then f' is a one-way function. The reduction is poly-preserving.*

(2) *Let $X \in_{\mathcal{U}} \{0,1\}^n$ and $Y \in_{\mathcal{U}} \{0,1\}^{p_n}$. Then $\mathbf{H}(f'(X,Y)) \geq n + p_n - 1/2$.*

Proof of Lemma 5.2(1). Suppose adversary A inverts $f'(X,Y)$ with probability δ_n in time T_n. We prove that the following oracle TM M using A on input $z = f(x)$ finds $x' \in \mathrm{pre}_f(z)$ with probability at least $\delta_n^3/128$ when $x \in_{\mathcal{U}} \{0,1\}^n$.

Description of $M^{(A)}(z)$.
Compute $\tilde{\mathbf{D}}_f(z)$.
Choose $\alpha \in_{\mathcal{U}} \{0,1\}^{\tilde{\mathbf{D}}_f(z)+2}$.
Choose $y \in_{\mathcal{U}} \{0,1\}^{p_n}$.
If $A(z,\alpha,y)$ outputs x' with $f(x') = z$ then output x'.

Let $j_n = 2\lceil \log(2/\delta_n)\rceil$. For all $z \in \mathrm{range}_f$, for all $y \in \{0,1\}^{p_n}$, define random variable

$$\beta_{z,y} = h_y(W)_{\{1,\ldots,\tilde{\mathbf{D}}_f(z)-j_n\}},$$

where $W \in_{\mathcal{U}} \mathrm{pre}_f(z)$. Then the probability that there is a $\gamma \in \{0,1\}^{2+j_n}$ such that A inverts f on input $\langle f(X), \langle \beta_{f(X),Y}, \gamma\rangle, Y\rangle$ is at least δ_n.

Fix $z \in \mathrm{range}_f$, and let $\beta'_z \in_{\mathcal{U}} \{0,1\}^{\tilde{\mathbf{D}}_f(z)-j_n}$. By Lemma 4.8,

$$\mathbf{L_1}(\langle \beta_{z,Y}, Y\rangle, \langle \beta'_z, Y\rangle) \leq \delta_n/2.$$

Therefore, the probability that there is a $\gamma \in \{0,1\}^{2+j_n}$ such that A inverts f on input $\langle f(X), \langle \beta'_{f(X)}, \gamma\rangle, Y\rangle$ is at least $\delta_n/2$. If we choose $\gamma \in_{\mathcal{U}} \{0,1\}^{2+j_n}$, we have

that the probability that A inverts $f(X)$ on input $\langle f(X), \langle \beta'_{f(X)}, \gamma\rangle, Y\rangle$ is at least

$$2^{-(2+j_n)} \cdot \frac{\delta_n}{2} \geq \frac{\delta_n^3}{128}.$$

Note that this is the input distribution in the call to A within $M^{(A)}$. Note also that the run time of $M^{(A)}$ is dominated by the time to run A. Thus, the time–success ratio of $M^{(A)}$ for inverting f' is about $128T_n/\delta_n^3$.

Proof of Lemma 5.2(2). Fix $z \in \text{range}_f$ and let $x, x' \in \text{pre}_f(z)$ such that $x \neq x'$. From the properties of a universal hash function,

$$\Pr[h_Y(x)_{\{1,\ldots,\tilde{\mathbf{D}}_f(z)+2\}} = h_Y(x')_{\{1,\ldots,\tilde{\mathbf{D}}_f(z)+2\}}] = 2^{-(\tilde{\mathbf{D}}_f(z)+2)} \leq \frac{1}{4 \cdot \sharp\text{pre}_f(z)}.$$

By calculating the Renyi entropy it follows that

$$\mathbf{H}(f'(X,Y)) \geq -\log\left(\frac{5}{4} \cdot 2^{-n+p_n}\right) = n + p_n + 2 - \log(5).$$

The result follows since $\log(5) \leq 5/2$. □

COROLLARY 5.3. *Let f, h, and f' be as described in Construction* 5.1. *Let $r \in \{0,1\}^n$ and define $\mathbf{P}$-time function ensemble $g(x,y,r) = \langle f'(x,y), r, x \odot r\rangle$. Let f be a one-way function. Then g is a pseudoentropy generator with pseudoentropy* $1/2$. *The reduction is poly-preserving.*

Proof. The proof is the same as the proof of Proposition 4.5. Let $X, R \in_{\mathcal{U}} \{0,1\}^n$, $Y \in_{\mathcal{U}} \{0,1\}^{p_n}$, and $\beta \in_{\mathcal{U}} \{0,1\}$. From Lemma 5.2 (1) and Proposition 4.3 it follows that $g(X,Y,R)$ and $\langle f'(X,Y), R, \beta\rangle$ are computationally indistinguishable, where the reduction is poly-preserving. From Lemma 5.2 (2) it follows that $\mathbf{H}(f'(X,Y), R, \beta) \geq 2n + p_n + 1/2$. On the other hand, the input entropy to $g(X,Y,R)$ is $2n + p_n$, and thus it follows that g has pseudoentropy $1/2$. □

THEOREM 5.4. *A pseudorandom generator can be constructed from a one-way function f where $\tilde{\mathbf{D}}_f$ is a $\mathbf{P}$-time function ensemble. The reduction is weak-preserving.*

Proof. For the proof, combine Construction 5.1 with Construction 4.13, and use Corollary 5.3 and Theorem 4.14. □

The following theorem, an easy corollary of Theorem 5.4, was previously obtained by [GKL93] using a different construction and proof techniques.

THEOREM 5.5. *Let $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$ be a σ_n-regular one-way function, where σ_n is a $\mathbf{P}$-time polynomial parameter. Then a pseudorandom generator can be constructed from f. The reduction is weak-preserving.*

Proof. Note that in this case $\tilde{\mathbf{D}}_f(f(x)) = \lceil \log(\sigma_n) \rceil$ for all $x \in \{0,1\}^n$. Furthermore, $\lceil \log(\sigma_n) \rceil \in \{0, \ldots, n\}$. Using this, combining Construction 5.1 with Construction 4.13, and using Corollary 5.3 and Theorem 4.14 yield a mildly nonuniform pseudorandom generator. Then Proposition 4.17 shows how to construct a pseudorandom generator from this. □

Based on the ideas presented above, [Luby96, Theorems 10.1 and 9.3] gives versions of Theorems 5.4 and 5.5 where the reduction is poly-preserving when the security parameter is $\mathbf{P}$-time computable.

6. Any one-way function to a false-entropy generator.

6.1. Finding determined hidden bits. The final step in the general construction of a pseudorandom generator from a one-way function is to construct a false-entropy generator from any one-way function. This is the technically most difficult part of this paper. This construction uses some of the ideas from Construction 5.1. Let

$$f : \{0,1\}^n \to \{0,1\}^{\ell_n} \tag{6.1}$$

be a one-way function and let

$$h : \{0,1\}^{p_n} \times \{0,1\}^n \to \{0,1\}^{n+\lceil \log(2n) \rceil} \tag{6.2}$$

be a universal hash function. Similar to Construction 5.1, for $x \in \{0,1\}^n$, $i \in \{0,\dots,n-1\}$, and $r \in \{0,1\}^{p_n}$, define **P**-time function ensemble

$$f'(x,i,r) = \langle f(x), h_r(x)_{\{1,\dots,i+\lceil \log(2n) \rceil\}}, i, r \rangle. \tag{6.3}$$

Note that from Lemma 5.2, the restricted function $f'(x, \tilde{\mathbf{D}}_f(f(x)), r)$ is an almost one-to-one one-way function, except that this is not necessarily a **P**-time function ensemble since $\tilde{\mathbf{D}}_f$ may not be a **P**-time function ensemble, and this is the main difficulty we must overcome.

Let $X \in_{\mathcal{U}} \{0,1\}^n$, $R \in_{\mathcal{U}} \{0,1\}^{p(n)}$, $Y \in_{\mathcal{U}} \{0,1\}^n$, and $\beta \in_{\mathcal{U}} \{0,1\}$. From the proof of Lemma 5.2 and Corollary 5.3, we claim and formalize below that if $i \le \tilde{\mathbf{D}}_f(f(X))$, then a time limited adversary cannot distinguish $X \odot Y$ from β given Y and $f'(X,i,R)$.

Let

$$\mathcal{T} = \{\langle x,i\rangle | x \in \{0,1\}^n, i \in \{0,\dots,\tilde{\mathbf{D}}_f(f(x))\}\},$$
$$\overline{\mathcal{T}} = \{\langle x,i\rangle | x \in \{0,1\}^n, i \in \{\tilde{\mathbf{D}}_f(f(x))+1,\dots,n-1\}\}.$$

LEMMA 6.1. *Let $W = \langle \tilde{X}, \tilde{I} \rangle \in_{\mathcal{U}} \mathcal{T}$, $R \in_{\mathcal{U}} \{0,1\}^{p(n)}$, $Y \in_{\mathcal{U}} \{0,1\}^n$, and $\beta \in_{\mathcal{U}} \{0,1\}$. Let f be a one-way function. Then*

$$\langle f'(W,R), \tilde{X} \odot Y, Y \rangle \text{ and } \langle f'(W,R), \beta, Y \rangle$$

are computationally indistinguishable. The reduction is poly-preserving.

Proof. Let A be an $\mathbf{R}'$-breaking adversary for distinguishing the two distributions. Then A is also an $\mathbf{R}'$-breaking adversary for hidden bit $\tilde{X} \odot Y$ given $\langle f'(W,R), Y \rangle$. Then, from Proposition 4.2, there is an oracle TM M' such that $M'^{(A)}$ is an $\mathbf{R}''$-breaking adversary for inverting $f'(W,R)$, where $\mathbf{R}''(n) = n^{\mathcal{O}(1)} \cdot \mathbf{R}'(n)^{\mathcal{O}(1)}$. Finally, we use the same idea as in Lemma 5.2; i.e., the success probability of $M'^{(A)}$ on input $\langle f(x), \alpha, i, R \rangle$ for $\alpha \in_{\mathcal{U}} \{0,1\}^{i+\lceil \log(2n) \rceil}$ is at least inverse polynomial in the success probability of $M'^{(A)}$ on input $f'(x,i,R)$ for each fixed $\langle x,i \rangle \in \mathcal{T}$. Consider the following oracle TM N: N^A on input $f(x)$ chooses $i \in_{\mathcal{U}} \{0,\dots,n-1\}$, $\alpha \in_{\mathcal{U}} \{0,1\}^{i+\lceil \log(2n) \rceil}$, and $r \in_{\mathcal{U}} \{0,1\}^{p_n}$ and runs $M'^{(A)}$ on input $\langle f(x), \alpha, i, r \rangle$. Since $\Pr[\langle x,i \rangle \in \mathcal{T}] \ge 1/n$ when $i \in_{\mathcal{U}} \{0,\dots,n-1\}$, it follows that $N^A(f(X))$ produces an inverse with probability at least $1/n$ times the probability $M'^{(A)}(f'(W,R))$ produces an inverse. ∎

If $i \ge \tilde{\mathbf{D}}_f(f(X))$, then X is almost completely determined by $f'(X,i,R)$, and thus $X \odot Y$ is almost completely determined by $f'(X,i,R)$ and Y.

The interesting case is when $i = \tilde{\mathbf{D}}_f(f(X))$, in which case, from Lemma 6.1, the adversary is unable to distinguish $X \odot Y$ from β given Y and $f'(X, i, R)$, and yet from Lemma 5.2(2), $X \odot Y$ is almost completely determined by $f'(X, i, R)$ and Y. It is from this case that we can extract a little bit of false entropy.

6.2. Construction and main theorem. We now describe the construction of a false-entropy generator g based on f'. Let

$$k_n \geq 125n^3. \tag{6.4}$$

Part of the construction is to independently and randomly choose k_n sets of inputs to f' and concatenate the outputs. In particular, let $X' \in_{\mathcal{U}} \{0,1\}^{k_n \times n}$, $I' \in_{\mathcal{U}} \{0,1\}^{k_n \times \lceil \log(n) \rceil}$, $R' \in_{\mathcal{U}} \{0,1\}^{k_n \times p_n}$. Part of the construction is then $f'^{k_n}(X', I', R')$.

Let $I \in_{\mathcal{U}} \{0, \ldots, n-1\}$, let

$$\mathbf{p}_n = \Pr[I \leq \tilde{\mathbf{D}}_f(f(X))], \tag{6.5}$$

and let

$$m_n = k_n \mathbf{p}_n - 2k_n^{2/3}. \tag{6.6}$$

We show later that it is sufficient to have an approximation of $\mathbf{p}_n$ to within an additive factor of $1/n$ for the entire construction to work. We need this to be able to claim that g described below is mildly nonuniform. For now we assume we have the exact value of $\mathbf{p}_n$. Let $Y' \in_{\mathcal{U}} \{0,1\}^{k_n \times n}$. The value of k_n is chosen to be large enough so that with high probability it is the case that $I'_j \leq \tilde{\mathbf{D}}_f(f(X'_j))$ for at least m_n of the k_n possible values of j, and in this case, from Lemma 6.1, $X'_j \odot Y'_j$ looks like a random bit to a time limited adversary given Y'_j and $f'(X'_j, I'_j, R'_j)$.

The problem is that we don't know for which set of m_n values of j the bit $X'_j \odot Y'_j$ looks random to a time limited adversary. Instead, the idea is to hash m_n bits out of all k_n such bits and release the hashed bits. The intuition is that these m_n hashed bits will look random to a time limited adversary, even though there are really at most $(\mathbf{p}_n - 1/n)k_n$ bits of randomness left in these k_n bits after seeing Y' and $f'^{k_n}(X', I', R')$, and thus there are approximately $m_n - (\mathbf{p}_n - 1/n)k_n \approx n^2$ bits of false entropy. Let

$$h' : \{0,1\}^{p'_n} \times \{0,1\}^{k_n} \to \{0,1\}^{m_n} \tag{6.7}$$

be a universal hash function, let $U \in_{\mathcal{U}} \{0,1\}^{p'_n}$, and define $\mathbf{P}$-time function ensemble

$$\begin{aligned} &g(\mathbf{p}_n, X', Y', I', R', U) \\ &= \langle h'_U(\langle X'_1 \odot Y'_1, \ldots, X'_{k_n} \odot Y'_{k_n} \rangle), f'^{k_n}(X', I', R'), U, Y' \rangle. \end{aligned} \tag{6.8}$$

THEOREM 6.2. *Let f be a one-way function and g be as described above in* (6.1)–(6.8). *Then g is a mildly nonuniform false-entropy generator with false entropy $10n^2$. The reduction is weak-preserving.*

Proof. Let $Z \in_{\mathcal{U}} \{0,1\}^{m_n}$, and let

$$\begin{aligned} \mathcal{D} &= \langle h'_U(\langle X'_1 \odot Y'_1, \ldots, X'_{k_n} \odot Y'_{k_n} \rangle), f'^{k_n}(X', I', R'), U, Y' \rangle, \\ \mathcal{E} &= \langle Z, f'^{k_n}(X', I', R'), U, Y' \rangle. \end{aligned}$$

Note that $\mathcal{D}$ is the distribution of the output of g, and $\mathcal{E}$ is the same except that the m_n output bits of h' have been replaced by random bits. Lemma 6.4 shows that $\mathbf{H}(\mathcal{E}) \geq \mathbf{H}(\mathcal{D}) + 10n^2$. Corollary 6.6 shows that if f is a one-way function, then we have that $\mathcal{D}$ and $\mathcal{E}$ are computationally indistinguishable, where the overall reduction is weak-preserving. □

What remain to prove Theorem 6.2 are the proofs of Lemmas 6.4 and 6.5 of the next subsection. (Corollary 6.6 follows immediately from Lemmas 6.5 and 6.1.) Before we turn to this, we state the main result of this paper based on Theorem 6.2.

THEOREM 6.3. *There are one-way functions iff there are pseudorandom generators.*

Proof. That pseudorandom generators imply one-way functions follows from [Levin87]. The converse now follows from Theorem 6.2 and the results are summarized in subsection 4.9. □

6.3. The main lemmas.

LEMMA 6.4. $\mathbf{H}(\mathcal{E}) \geq \mathbf{H}(\mathcal{D}) + 10n^2$.

Proof. The entropy of $\mathcal{D}$ and $\mathcal{E}$ excluding the first m_n bits is exactly the same. The additional entropy in the first m_n bits of $\mathcal{E}$ is equal to m_n. An upper bound on the additional entropy in the first m_n bits of $\mathcal{D}$ is the additional entropy in $\langle X'_1 \odot Y'_1, \ldots, X'_{k_n} \odot Y'_{k_n} \rangle$. For each $j \in \{1, \ldots, k_n\}$ where $I'_j < \tilde{\mathbf{D}}_f(f(X'_j))$, the amount of entropy added by $X'_j \odot Y'_j$ is at most 1. On the other hand, under the condition that $I'_j \geq \tilde{\mathbf{D}}_f(f(X'_j))$, $X'_j \odot Y'_j$ is determined by $\langle f'(X'_j, I'_j, R'_j), Y'_j \rangle$ with probability at least $1 - 1/2n$, and thus the additional entropy under this condition is at most $1/2n$. Since $I'_j < \tilde{\mathbf{D}}_f(f(X'_j))$ with probability $\mathbf{p}_n - 1/n$, it follows that the additional entropy added by $X'_j \odot Y'_j$ is at most $\mathbf{p}_n - 1/2n$. Therefore, the additional entropy in the first m_n bits of $\mathcal{D}$ is at most $k_n(\mathbf{p}_n - 1/2n) = m_n + 2k_n^{2/3} - k_n/2n < m_n - 10n^2$ by choice of k_n. □

LEMMA 6.5. *Let A be an adversary with distinguishing probability*

$$\delta_n = \Pr[A(\mathcal{D}) = 1] - \Pr[A(\mathcal{E}) = 1]$$

for $\mathcal{D}$ and $\mathcal{E}$. (We assume without loss of generality that $\Pr[A(\mathcal{D}) = 1] > \Pr[A(\mathcal{E}) = 1]$.) Let $W = \langle \tilde{X}, \tilde{I} \rangle \in_{\mathcal{U}} \mathcal{T}$, $R \in_{\mathcal{U}} \{0,1\}^{p(n)}$, $Y \in_{\mathcal{U}} \{0,1\}^n$, and $\beta \in_{\mathcal{U}} \{0,1\}$. There is an oracle TM M such that $M^{(A)}$ distinguishes between

$$\langle f'(W, R), \tilde{X} \odot Y, Y \rangle \text{ and } \langle f'(W, R), \beta, Y \rangle$$

with probability at least $\delta_n/(16k_n)$. The running time of $M^{(A)}$ is polynomial in the running time of A.

The proof of Lemma 6.5 is the most technically involved in this paper. Before proving this lemma, we give the main corollary to this lemma, and then we give some motivation for the proof of the lemma.

COROLLARY 6.6. *Let f be a one-way function. Then $\mathcal{D}$ and $\mathcal{E}$ are computationally indistinguishable. The reduction is weak-preserving.*

Proof. For the proof, combine Lemma 6.1 with Lemma 6.5. □

We now give some intuition to the proof of Lemma 6.5. The oracle TM $M^{(A)}$ will use a nonstraightforward hybrid of distributions argument to be able to distinguish the two distributions in the statement of the lemma. To give some intuition about this nonstraightforward hybrid, we first describe a related straightforward hybrid argument that we do not know how to implement efficiently.

Consider the following distribution. For $j \in \{1, \ldots, k_n\}$, let $C_j = 1$ with probability $\mathbf{p}_n$ and $C_j = 0$ with probability $1 - \mathbf{p}_n$. For all j, if $C_j = 1$ then let $\langle \tilde{X}'_j, \tilde{I}'_j \rangle \in_{\mathcal{U}} \mathcal{T}$, and if $C_j = 0$ then let $\langle \tilde{X}'_j, \tilde{I}'_j \rangle \in_{\mathcal{U}} \overline{\mathcal{T}}$. Let R', Y', and U be as defined previously. If these random variables are used to define a distribution using the same construction as used to define $\mathcal{D}$, with $\langle \tilde{X}'_j, \tilde{I}'_j \rangle$ replacing $\langle X'_j, I'_j \rangle$, then the distribution is $\mathcal{D}$, except that it is described in a slightly different way. Now, suppose this distribution is altered as follows: if $C_j = 1$, then change the jth input bit of h_U from $\tilde{X}'_j \odot Y'_j$ to $B_j \in_{\mathcal{U}} \{0, 1\}$. Call this distribution $\mathcal{D}'$.

From Lemma 6.1 intuitively it should be the case that a time limited adversary should not be able to distinguish $\mathcal{D}$ from $\mathcal{D}'$. On the other hand, it is not hard to see using Lemma 4.8 that the statistical distance between $\mathcal{D}'$ and $\mathcal{E}$ is exponentially small in n. Thus, if adversary A can distinguish between $\mathcal{D}$ and $\mathcal{E}$, we should be able to use this to distinguish $\langle f'(W, R), \tilde{X} \odot Y, Y \rangle$ and $\langle f'(W, R), \beta, Y \rangle$ as in the statement of Lemma 6.5.

The question is whether we can really prove that $\mathcal{D}'$ is computationally indistinguishable from $\mathcal{D}$. Toward resolving this question, consider the following family of hybrid distributions. For all $j \in \{0, \ldots, k_n\}$, let $\mathcal{F}^{(j)}$ be the hybrid distribution between $\mathcal{D}$ and $\mathcal{E}$, which is the same as $\mathcal{D}'$ up to position j and the same as $\mathcal{D}$ thereafter; i.e., it is the same as $\mathcal{D}$ except that for all $i \leq j$, if $C_i = 1$ then change the ith input bit of h_U from $\tilde{X}'_i \odot Y'_i$ to $B_i \in_{\mathcal{U}} \{0, 1\}$. Then $\mathcal{F}^{(0)} = \mathcal{D}$ and $\mathcal{F}^{(k_n)} \approx \mathcal{E}$. Let $J \in_{\mathcal{U}} \{1, \ldots, k_n\}$. Then $\mathrm{E}_J[A(\mathcal{F}^{(J-1)}) - A(\mathcal{F}^{(J)})] = \delta_n / k_n$.

An inefficient oracle TM could work as follows on input $\langle f'(w, r), b, y \rangle$: the first phase chooses $j \in_{\mathcal{U}} \{1, \ldots, k_n\}$ and chooses a sample from $\mathcal{F}^{(j)}$. If $c_j = 0$, then the oracle TM produces a random bit and stops. In the more interesting case, where $c_j = 1$, it replaces the inputs corresponding to the jth position in the sample according to $f'(w, r)$ and y, and the jth input bit of h_u is set to $b \in_{\mathcal{U}} \{0, 1\}$. Then the second phase runs the adversary A on this input and outputs the bit produced by A. The distinguishing probability for this oracle TM is δ_n / k_n. The problem is that this is not an efficient oracle TM, because it may not be possible to efficiently uniformly sample from $\mathcal{T}$ and $\overline{\mathcal{T}}$ as required. However, it is possible to sample uniformly from $\{0, 1\}^n \times \{0, \ldots, n-1\}$: a $\mathbf{p}_n$ fraction of the samples will be randomly distributed in $\mathcal{T}$ and a $1 - \mathbf{p}_n$ fraction of the samples will be randomly distributed in $\overline{\mathcal{T}}$. This simple idea is used to construct the efficient adversary described below.

The efficient adversary $M^{(A)}$ described in detail in the proof of Lemma 6.5 proceeds in two phases similar to the inefficient oracle TM described above. The first phase of $M^{(A)}$ consists of k_n stages, where stage j produces a coupled pair of distributions, $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$, both of which are polynomially samplable. Each stage consists of using adversary A and sampling from the distributions produced in the previous stage to produce the pair of output distributions for the current stage. Initially, $\mathcal{D}^{(0)} = \mathcal{D}$ and $\mathcal{E}^{(0)} = \mathcal{E}$, and it will turn out that $\mathcal{D}^{(k_n)} \approx \mathcal{E}^{(k_n)}$.

The first $j - 1$ positions in both $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$ are already fixed in essentially the same way in $\mathcal{D}^{(j-1)}$ and $\mathcal{E}^{(j-1)}$, and these positions will be fixed the same way in $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$. To fill in position j in $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$, many samples of $\langle x_j, i_j \rangle$ are drawn uniformly from $\{0, 1\}^n \times \{0, \ldots, n-1\}$, and then with high probability many of them will be in $\mathcal{T}$ and many will be in $\overline{\mathcal{T}}$. We cannot directly tell for each sample whether it is in $\mathcal{T}$ or $\overline{\mathcal{T}}$. Thus, we must use another criterion to decide which of the samples to keep to fill in position j. The criterion employed is to use the sample for which the distinguishing probability of A between $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$ is highest when the jth position is fixed according to the sample.

Let $\delta^{(j)} = \Pr[A(\mathcal{D}^{(j)}) = 1] - \Pr[A(\mathcal{E}^{(j)}) = 1]$. Because $\delta^{(0)} = \delta_n$ and $\delta^{(k_n)} \approx 0$, it follows that

$$\mathrm{E}_{j \in_{\mathcal{U}} \{1,\ldots,k_n\}}[\delta^{(j-1)} - \delta^{(j)}] \geq \delta_n / k_n.$$

It is because of this discrepancy between the value of $\delta^{(j)}$ and $\delta^{(j-1)}$ that f' can be inverted in the second phase.

Intuitively, stage j of the first phase works as follows. A bit c_j is chosen randomly to be one with probability $\mathbf{p}_n$ and to be zero with probability $1 - \mathbf{p}_n$. In the distribution $\mathcal{D}^{(j)}$, the jth input $\langle x'_j, i'_j, r'_j \rangle$ to ${f'}^{k_n}$ is chosen randomly, y'_j is chosen randomly, u is chosen randomly, and then the jth input bit of ${h'}_u$ is set to a random bit b_j if $c_j = 1$ and to the correct inner product bit if $c_j = 0$. In the distribution $\mathcal{E}^{(j)}$, the jth input of ${f'}^{k_n}$ is set the same way it is set in $\mathcal{D}^{(j)}$, and thus the two distributions $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$ are correlated. The choice of the jth inputs is done several times (c_j is chosen only once at the beginning, i.e., it is not rechosen for each of the times) and each time the distinguishing probability of A for $\mathcal{D}^{(j)}$ and the corresponding $\mathcal{E}^{(j)}$ is approximated, and the choice that maximizes the difference between these accepting probabilities determines how $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$ are finally set.

The second phase of $M^{(A)}$ chooses $j \in_{\mathcal{U}} \{1, \ldots, k_n\}$ and then uses the pair of distributions $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$ produced in the first stage. The idea is to choose a random sample from both $\mathcal{D}^{(j)}$ and $\mathcal{E}^{(j)}$, modify portions of the $\mathcal{D}^{(j)}$ part according to the input to $M^{(A)}$, run A on both the modified $\mathcal{D}^{(j)}$ sample and the $\mathcal{E}^{(j)}$ sample, and, based on the outputs, produce a one bit output. The intuition is that the distinguishing probability will be $\delta^{(j)}$, which on average over all j is at least δ_n / k_n.

We now turn to the formal proof of Lemma 6.5.

Proof of Lemma 6.5. The oracle TM $M^{(A)}$ works as follows on input $\langle f'(w, r), b, y \rangle$.

Phase 1. Define $\mathcal{D}^{(0)} = \mathcal{D}$ and $\mathcal{E}^{(0)} = \mathcal{E}$. Let $B \in_{\mathcal{U}} \{0,1\}^{k_n}$. Let $\rho = \delta_n / (16 k_n)$ and $\tau = 64 n^2 / \rho$. Stage $j = 1, \ldots, k_n$ works as follows: randomly choose $c_j \in \{0,1\}$ so that $c_j = 1$ with probability $\mathbf{p}_n$. Choose $\hat{x}_1, \ldots, \hat{x}_\tau \in_{\mathcal{U}} \{0,1\}^n$ and $\hat{i}_1, \ldots, \hat{i}_\tau \in_{\mathcal{U}} \{0, \ldots, n-1\}$. For each $m \in \{1, \ldots, \tau\}$, define $w_m = \langle \hat{x}_m, \hat{i}_m \rangle$ and let $\mathcal{D}_{c_j}^{(j-1)}(w_m)$ be the same as $\mathcal{D}^{(j-1)}$ except that $\langle X'_j, I'_j \rangle$ is fixed to w_m and the jth input bit of h' is set to

$$\begin{cases} \hat{x}_m \odot Y'_j & \text{if } c_j = 0, \\ B_j & \text{if } c_j = 1. \end{cases}$$

Similarly, define $\mathcal{E}^{(j-1)}(w_m)$ to be the same as $\mathcal{E}^{(j-1)}$ except that $\langle X'_j, I'_j \rangle$ is fixed to w_m. Let

$$\delta_{c_j}^{(j-1)}(w_m) = \Pr[A(\mathcal{D}_{c_j}^{(j-1)}(w_m)) = 1] - \Pr[A(\mathcal{E}^{(j-1)}(w_m)) = 1].$$

Using A and sampling $\mathcal{O}(n/\rho^2)$ times from $\mathcal{D}_{c_j}^{(j-1)}(w_m)$ and $\mathcal{E}^{(j-1)}(w_m)$, produce an estimate $\Delta_{c_j}^{(j-1)}(w_m)$ so that

$$\Pr[|\Delta_{c_j}^{(j-1)}(w_m) - \delta_{c_j}^{(j-1)}(w_m)| > \rho] \leq 2^{-n}.$$

Let $m_0 \in \{1, \ldots, \tau\}$ be the index for which $\Delta_{c_j}^{(j-1)}(w_{m_0})$ is maximized. Set $\langle x'_j, i'_j \rangle = w_{m_0}$, $\mathcal{D}^{(j)} = \mathcal{D}_{c_j}^{(j-1)}(w_{m_0})$, $\mathcal{E}^{(j)} = \mathcal{E}^{(j-1)}(w_{m_0})$ and go to the next stage.

Phase 2. Pick $j \in_{\mathcal{U}} \{0, \ldots, k_n - 1\}$. Let $\mathcal{D}^{(j)}(w, r, b, y)$ be the distribution $\mathcal{D}^{(j)}$ except that $f'(X'_{j+1}, I'_{j+1}, R'_{j+1})$ is set to $f'(w, r)$ and the $j + 1^r$st input bit of h' is set to b and Y'_{j+1} is set to y. Let $\mathcal{E}^{(j)}(w, r, y)$ be the same as $\mathcal{E}^{(j)}$ except that $f'(X'_{j+1}, I'_{j+1}, R'_{j+1})$ is set to $f'(w, r)$ and Y'_{j+1} is set to y. Let $\beta \in_{\mathcal{U}} \{0, 1\}$, let D be a sample of $\mathcal{D}^{(j)}(w, r, b, y)$, and let E be a sample of $\mathcal{E}^{(j)}(w, r, y)$. If $A(D) = A(E)$, then output β else output $A(D)$.

We now prove that the oracle adversary $M^{(A)}$ as just described distinguishes as claimed in the lemma. Let $w = \langle x, i \rangle$, $d^{(j)}(w, r, b, y) = \mathrm{E}[A(\mathcal{D}^{(j)}(w, r, b, y))]$ and $e^{(j)}(w, r, y) = \mathrm{E}[A(\mathcal{E}^{(j)}(w, r, y))]$. Then

$$\Pr[M^{(A)}(f'(w, r), b, y) = 1] = 1/2 + (d^{(j)}(w, r, b, y) - e^{(j)}(w, r, y))/2.$$

Also, it follows directly from the definitions that

$$\mathrm{E}[d^{(j)}(w, R, x \odot Y, Y) - e^{(j)}(w, R, Y)] = \delta_0^{(j)}(w)$$

and

$$\mathrm{E}[d^{(j)}(w, R, \beta, Y) - e^{(j)}(w, R, Y)] = \delta_1^{(j)}(w).$$

Let $\epsilon^{(j)} = \mathrm{E}[\delta_0^{(j)}(W) - \delta_1^{(j)}(W)]$. Thus, the distinguishing probability of $M^{(A)}$ is

$$\begin{aligned} &\mathrm{E}[M^{(A)}(f'(W, R), \tilde{X} \odot Y, Y)] - \mathrm{E}[M^{(A)}(f'(W, R), \beta, Y)] \\ &= \mathrm{E}_j[\delta_0^{(j)}(W) - \delta_1^{(j)}(W)]/2 = \mathrm{E}_j[\epsilon^{(j)}]/2, \end{aligned}$$

where $j \in_{\mathcal{U}} \{0, \ldots, k_n - 1\}$ in the last two expectations. To prove the lemma, it is sufficient to show that $\mathrm{E}_j[\epsilon^{(j)}]/2 \geq \rho$ or, equivalently,

$$\mathrm{E}\left[\sum_{j \in \{0, \ldots, k_n - 1\}} \epsilon^{(j)}\right] \geq 2\rho k_n = \delta_n/8. \tag{6.9}$$

The expectation here is over the random choices of $M^{(A)}$ in the first phase. Let $\delta^{(j)} = \Pr[A(\mathcal{D}^{(j)}) = 1] - \Pr[A(\mathcal{E}^{(j)}) = 1]$. We prove (6.9) by showing the following below:

(a) $\mathrm{E}[\delta^{(k_n)}] \leq 2^{-n}$. The expectation is over the random choices of $M^{(A)}$ in the first phase.

(b) $\mathrm{E}[\delta^{(j)} - \delta^{(j+1)}] \leq \epsilon^{(j)} + 4\rho$. The expectation is over random choices in the $j + 1^r$st stage of Phase 1 conditional on any set of choices in the previous stages.

From (a) and (b), and because $\delta^{(0)} = \delta_n$, it follows that

$$\begin{aligned} \delta_n/2 &< \delta_n - \mathrm{E}[\delta^{(k_n)}] \\ &= \sum_{j \in \{0, \ldots, k_n - 1\}} \mathrm{E}[\delta^{(j)} - \delta^{(j+1)}] \\ &\leq 4k_n\rho + \mathrm{E}\left[\sum_{j \in \{0, \ldots, k_n - 1\}} \epsilon^{(j)}\right] \end{aligned}$$

$$= \delta_n/4 + \mathrm{E}\left[\sum_{j\in\{0,\ldots,k_n-1\}} \epsilon^{(j)}\right],$$

and this proves the bound in (6.9). Thus, it suffices to prove (a) and (b) above.

Proof of (a). Since $\Pr[c_j = 1] = \mathbf{p}_n$, applying Chernoff bounds (e.g., see [MR95]), we get that, with probability at least $1 - 2^{-n}$,

$$\sum_{j\in\{0,\ldots,k_n-1\}} c_j \geq k_n\mathbf{p}_n - k_n^{2/3} = m_n + {k_n}^{2/3}.$$

The entropy of the input to h' conditional on the rest of the bits of $\mathcal{D}^{(k_n)}$ is at least $\sum_{j\in\{0,\ldots,k_n-1\}} c_j$. So, if this sum is at least $m_n + {k_n}^{2/3}$, applying Lemma 4.8, $\mathbf{L_1}(\mathcal{D}^{(k_n)}, \mathcal{E}^{(k_n)}) \leq 2^{-n}$. Thus, $\delta^{(k_n)} = \mathrm{E}[A(\mathcal{D}^{(k_n)})] - \mathrm{E}[A(\mathcal{E}^{(k_n)})] \leq 2^{-n}$.

Proof of (b). Let $\bar{W} \in_{\mathcal{U}} \overline{\mathcal{T}}$, and recall that $W \in_{\mathcal{U}} \mathcal{T}$. Then, since the $j+1$st input of h' is always $X'_{j+1} \odot Y'_{j+1}$ in $\mathcal{D}^{(j)}$,

$$\begin{aligned}
\delta^{(j)} &= \mathbf{p}_n\mathrm{E}[\delta_0^{(j)}(W)] + (1-\mathbf{p}_n)\mathrm{E}[\delta_0^{(j)}(\bar{W})] \\
&= \mathbf{p}_n\mathrm{E}[\delta_1^{(j)}(W)] + \mathbf{p}_n(\mathrm{E}[\delta_0^{(j)}(W)] - \mathrm{E}[\delta_1^{(j)}(W)]) + (1-\mathbf{p}_n)(\mathrm{E}[\delta_0^{(j)}(\bar{W})]) \\
&= \mathbf{p}_n\mathrm{E}[\delta_1^{(j)}(W)] + \mathbf{p}_n\epsilon^{(j)} + (1-\mathbf{p}_n)(\mathrm{E}[\delta_0^{(j)}(\bar{W})]) \\
&< \epsilon^{(j)} + \mathbf{p}_n\mathrm{E}[\delta_1^{(j)}(W)] + (1-\mathbf{p}_n)(\mathrm{E}[\delta_0^{(j)}(\bar{W})]).
\end{aligned}$$

We now show that $\mathrm{E}[\delta^{(j+1)}] \geq \mathbf{p}_n\mathrm{E}[\delta_1^{(j)}(W)] + (1-\mathbf{p}_n)(\mathrm{E}[\delta_0^{(j)}(\bar{W})]) - 4\rho$, and this concludes the proof. Let $c \in \{0,1\}$ and consider stage j in Phase 1. From our choice of τ and the fact that $1/n \leq \mathbf{p}_n \leq 1 - 1/n$, it follows that, with probability at least $1 - 2^{-n}$, at least n/ρ of the w_m's are in $\mathcal{T}$, and at least n/ρ of the w_m's are in $\overline{\mathcal{T}}$. It then follows using Chernoff bounds that

$$\Pr[\max_{1\leq m\leq\tau}\{\delta_c^{(j)}(w_m)\} \geq \max\{\mathrm{E}[\delta_c^{(j)}(W)], \mathrm{E}[\delta_c^{(j)}(\bar{W})]\} - \rho]$$

is at least $1 - 2^{-n}$. Also, with probability at least $1 - 2^{-n}$, $\Delta_c^{(j)}(w_m)$ is within ρ of the corresponding $\delta_c^{(j)}(w_m)$, and thus (recalling how w_{m_0} is chosen above in stage j)

$$\begin{aligned}
\delta_c^{(j)}(w_{m_0}) &\geq \Delta_c^{(j)}(w_{m_0}) - \rho \\
&= \max_{m\in\{1,\ldots,\tau\}}\{\Delta_c^{(j)}(w_m)\} - \rho \\
&\geq \max_{m\in\{1,\ldots,\tau\}}\{\delta_c^{(j)}(w_m)\} - 2\rho \\
&\geq \max\{\mathrm{E}[\delta_c^{(j)}(W)], \mathrm{E}[\delta_c^{(j)}(\bar{W})]\} - 3\rho
\end{aligned}$$

with probability at least $1 - 3\cdot 2^{-n}$. Let $\delta_c^{(j+1)}$ be the value of $\delta^{(j+1)}$ conditional on $c_{j+1} = c$. From this we can conclude that

$$\mathrm{E}[\delta_c^{(j+1)}] \geq \max\{\mathrm{E}[\delta_c^{(j)}(W)], \mathrm{E}[\delta_c^{(j)}(\bar{W})]\} - 4\rho.$$

Since $c_{j+1} = 1$ with probability $\mathbf{p}_n$,

$$\begin{aligned}
\mathrm{E}[\delta^{(j+1)}] &= \mathbf{p}_n\mathrm{E}[\delta_1^{(j+1)}] + (1-\mathbf{p}_n)\mathrm{E}[\delta_0^{(j+1)}] \\
&\geq \mathbf{p}_n\mathrm{E}[\delta_1^{(j)}(W)] + (1-\mathbf{p}_n)(\mathrm{E}[\delta_0^{(j)}(\bar{W})]) - 4\rho. \qquad \square
\end{aligned}$$

Before we continue let us just check that a good approximation of $\mathbf{p}_n$ is sufficient. Suppose that $\mathbf{p}_n \le \tilde{\mathbf{p}}_n \le \mathbf{p}_n + \frac{1}{n}$ and do the entire construction with $\tilde{\mathbf{p}}_n$ replacing $\mathbf{p}_n$. Enlarge $\mathcal{T}$ to density $\tilde{\mathbf{p}}_n$ by making it contain some elements $\langle x, i\rangle$ with $i = \tilde{\mathbf{D}}_f(f(x))+1$. Lemma 6.1 is easily seen to remain valid, and Lemma 6.4 just becomes more true in that the entropy of $\mathcal{D}$ decreases. This implies that it is sufficient to try $\mathcal{O}(n)$ different values of $\mathbf{p}_n$.

7. A direct construction. We have shown how to construct a false-entropy generator from an arbitrary one-way function, a pseudoentropy generator from a false-entropy generator, and finally a pseudorandom generator from a pseudoentropy generator. The combinations of these constructions give a pseudorandom generator from an arbitrary one-way function as stated in Theorem 6.3. By literally composing the reductions given in the preceding parts of this paper, we construct a pseudorandom generator with inputs of length n^{34} from a one-way function with inputs of length n. This is obviously not a suitable reduction for practical applications. In this subsection, we use the concepts developed in the rest of this paper, but we provide a more direct and efficient construction. However, this construction still produces a pseudorandom generator with inputs of length n^{10}, which is clearly still not suitable for practical applications. (A sharper analysis can reduce this to n^8, which is the best we could find using the ideas developed in this paper.) The result could only be considered practical if the pseudorandom generator had inputs of length n^2, or perhaps even close to n. (However, in many special cases of one-way functions, the ideas from this paper are practical; see, e.g., [Luby96].)

The improvement in the direct construction given here comes from the observation that more than one of the reductions involves a product distribution, whereas only one product distribution is needed for the overall proof.

We start with a one-way function $f : \{0,1\}^n \to \{0,1\}^{\ell_n}$. We construct f' as in (6.3), and let $\mathbf{p}_n$ be the probability that $I \le \tilde{\mathbf{D}}_f(f(X))$ as in the previous section. Let $\mathcal{X} = \langle X, I, R\rangle$ represent the input distribution to f', and let c_n be the length of $\mathcal{X}$ and c'_n the length of $f'(\mathcal{X})$. Let $\mathbf{e}_n = \mathbf{H}(f'(\mathcal{X}))$. Let $b(\mathcal{X}, y) = x \odot y$. Set $k_n = 2000n^6$.

Intuitively, we generate pseudorandom bits as follows: let $\mathcal{X}' = \mathcal{X}^{k_n}$ and $Y' = Y^{k_n}$. We first compute $f'^{k_n}(\mathcal{X}')$ and $b^{k_n}(\mathcal{X}', Y')$. Intuitively, we are entitled to recapture

$$k_n c_n - \mathbf{H}\langle f'^{k_n}(\mathcal{X}'), b^{k_n}(\mathcal{X}', Y')\rangle$$

bits from $\mathcal{X}'$, because this is the conditional entropy left after we have computed f'^{k_n} and b^{k_n}. We are entitled to recapture $k_n\mathbf{p}_n$ bits from the $b^{k_n}(\mathcal{X}', Y')$ (since we get a hidden bit out of each copy whenever $I \le \tilde{\mathbf{D}}_f(f(X))$). Finally, we should be able to extract $\mathbf{e}_n k_n$ bits from $f'^{k_n}(\mathcal{X}')$, since $\mathbf{e}_n$ is the entropy of $f'(\mathcal{X})$. Since $b(n)$ is almost totally predictable for almost all inputs where $I \ge \tilde{\mathbf{D}}_f(f(X))$,

$$\mathbf{H}\langle f'(\mathcal{X}), b(\mathcal{X}, Y)\rangle \le \mathbf{e}_n + \mathbf{p}_n - 1/n + 1/(2n).$$

(See the proof of Lemma 6.4.) Thus, if we add up all the output bits, we are entitled to $k_n(c_n + 1/(2n))$, or $k_n/(2n)$ more bits than the input to f'^{k_n}. However, our methods of extracting entropy are not perfect, so we need to sacrifice some bits at each stage; to use Corollary 4.10, we need to sacrifice $2nk_n^{2/3}$ at each stage, so we chose k_n to satisfy $k_n/(2n) > 6nk_n{}^{2/3}$.

Formally, let $m_n = k_n(c_n - \mathbf{e}_n - \mathbf{p}_n + 1/(2n)) - 2nk_n{}^{2/3}$, $m'_n = k_n\mathbf{p}_n - 2nk_n{}^{2/3}$, and $m''_n = k_n\mathbf{e}_n - 2nk_n{}^{2/3}$. Let R_1, R_2, and R_3 be indices of hash functions so that

h_{R_1} maps $k_n c_n$ bits to m_n bits, h_{R_2} maps k_n bits to m'_n bits, and h_{R_3} maps $k_n c'_n$ bits to m''_n bits. Our construction is as follows.

CONSTRUCTION 7.1.

$$g(\mathcal{X}', Y', R_1, R_2, R_3) = \langle h_{R_1}(\mathcal{X}'), h_{R_2}(b^{k_n}(\mathcal{X}', Y')), h_{R_3}(f'^{k_n}(\mathcal{X}')), Y', R_1, R_2, R_3\rangle.$$

THEOREM 7.2. *If f is a one-way function and g is as in Construction* 7.1*, then g is a mildly nonuniform pseudorandom generator. The reduction is weak-preserving.*

Proof. It is easy to check that g outputs more bits than it inputs.

As noted above, the conditional entropy of $\mathcal{X}$ given $f'(\mathcal{X})$ and $b(\mathcal{X}, Y)$ is at least $c_n - \mathbf{e}_n - \mathbf{p}_n + (1/2n)$. Thus, from Corollary 4.10, we have that $\langle h_{R_1}(\mathcal{X}'), R_1\rangle$ is statistically indistinguishable from random bits given $\langle f'^{k_n}(\mathcal{X}'), b^{k_n}(\mathcal{X}', Y'), Y'\rangle$. Hence, $g(\mathcal{X}', Y', R_1, R_2, R_3)$ is statistically indistinguishable from

$$\langle Z_1, h_{R_2}(b^{k_n}(\mathcal{X}', Y')), h_{R_3}(f'^{k_n}(\mathcal{X}')), Y', R_1, R_2, R_3\rangle,$$

where $Z_1 \in_{\mathcal{U}} \{0,1\}^{m_n}$. Now, from Lemmas 6.5 and 6.1, it follows that $h_{R_2}(b^{k_n}(\mathcal{X}', Y'))$ is computationally indistinguishable from random bits given $\langle f'^{k_n}(\mathcal{X}'), R_2, Y'\rangle$. Thus, $g(\mathcal{X}', Y', R_1, R_2, R_3)$ is computationally indistinguishable from

$$\langle Z_1, Z_2, h_{R_3}(f'^{k_n}(\mathcal{X}')), Y', R_1, R_2, R_3\rangle,$$

where $Z_2 \in_{\mathcal{U}} \{0,1\}^{m'_n}$. Finally, from Corollary 4.10, $\langle h_{R_3}(f'^{k_n}(\mathcal{X}')), R_3\rangle$ is statistically indistinguishable from $\langle Z_3, R_3\rangle$, where $Z_3 \in_{\mathcal{U}} \{0,1\}^{m''_n}$. Thus, the output of g is computationally indistinguishable from a truly random output of the same length. □

If we use hash functions constructed as Toeplitz matrices, then $\mathcal{O}(m)$ bits are sufficient to construct a hash function on m bits and the inputs needed for the hash function is just a constant fraction of all inputs. Then the input length to g is $\mathcal{O}(nk_n) = \mathcal{O}(n^7)$.

We still need to use Proposition 4.17 to get rid of the mild nonuniformity. From the arguments above, it is clear that an approximation of both $\mathbf{e}_n$ and $\mathbf{p}_n$ that is within $1/(8n)$ of their true values is sufficient. Since $0 \le \mathbf{e}_n \le n$, and $0 \le \mathbf{p}_n < 1$, there are at most $\mathcal{O}(n^3)$ cases of pairs to consider. This means that we get a total of $\mathcal{O}(n^3)$ generators, each needing an input of length $\mathcal{O}(n^7)$. Thus the total input size to the pseudorandom generator is $\mathcal{O}(n^{10})$, as claimed.

8. Conclusions. A general problem is to characterize the conditions under which cryptographic applications are possible. By conditions we mean complexity theoretic conditions, e.g., $P \neq NP$, the existence of one-way functions, etc. Examples of cryptographic applications are private key cryptography, identification/authentication, digital signatures, bit commitment, exchanging secrets, coin flipping over the telephone, etc.

For a variety of cryptographic applications it is known that a secure protocol can be constructed from a pseudorandom generator, e.g., the work of [GGM86], [LR88], [GMR89], [Naor88], [GMW91], shows that applications ranging from private key encryption to zero-knowledge proofs can be based on a pseudorandom generator. The results presented in this paper show that these same protocols can be based on any one-way function. The paper [NY89] gives a signature scheme that can be based on any one-way permutation, and [R90] substantially improves this by basing such a scheme on any one-way function.

Using the notion of a false-entropy generator, [G89] shows that the existence of pseudorandom generators is equivalent to the existence of a pair of **P**-samplable distributions which are computationally indistinguishable but statistically very different.

The paper [IL89] provides complementary results; a one-way function can be constructed from a secure protocol for any one of a variety of cryptographic applications, including private key encryption, identification/authentication, bit commitment, and coin flipping by telephone. The paper [OW93] shows that a one-way function can be constructed from any nontrivial zero-knowledge proof protocol. Thus, secure protocols for any of these applications are equivalent to the existence of one-way functions.

The results described in this paper and the previous three paragraphs show that the existence of a one-way function is *central* to modern complexity-based cryptography.

Some applications seem unlikely to be shown possible based on any one-way function; e.g., [IR89] gives strong evidence that exchanging secrets over a public channel is an application of this type.

A fundamental issue is that of efficiency, both in size and time; the general construction we give for a pseudorandom generator based on any one-way function increases the size of the input by a large polynomial amount and thus is only weak-preserving. This is not good news for practical applications; it would be nice to have a general poly-preserving or a linear-preserving reduction.

Acknowledgments. This research evolved over a long period of time and was greatly influenced by many people. We thank Amos Fiat, Moni Naor, Ronitt Rubinfeld, Manuel Blum, Steven Rudich, Noam Nisan, Lance Fortnow, Umesh Vazirani, Charlie Rackoff, Oded Goldreich, Hugo Krawczyk, and Silvio Micali for their insights and contributions to this work. We in particular thank Charlie, Umesh, and Manuel for their advice and enthusiasm, and Oded and Hugo for exposing the fourth author to their wealth of insights on this problem. Finally, Oded's insightful comments on every aspect of earlier versions of this paper have improved the presentation tremendously.

REFERENCES

[ACGS88] W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr, *RSA and Rabin functions: Certain parts are as hard as the whole*, SIAM J. Comput., 17 (1988), pp. 194–209.

[BFNW96] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson, *BPP has subexponential time simulations unless EXPTIME has publishable proofs*, Comput. Complexity, 3 (1993), pp. 307–318.

[BBR88] C. H. Bennett, G. Brassard, and J.-M. Robert, *Privacy amplification by public discussion*, SIAM J. Comput., 17 (1988), pp. 210–229.

[Blum84] M. Blum, *Independent unbiased coin flips from a correlated biased source—a finite state Markov chain*, Combinatoria, 6 (1986), pp. 97–108.

[BM82] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.

[BH89] R. Boppana and R. Hirschfeld, *Pseudo-random generators and complexity classes*, in Advances in Comp. Research 5, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 1–26.

[Boyar89] J. Boyar, *Inferring sequences produced by pseudo-random number generators*, J. Assoc. Comput. Mach., 36 (1989), pp. 129–141.

[CW79] L. Carter and M. Wegman, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[CG88] B. Chor and O. Goldreich, *Unbiased bits from sources of weak randomness and probabilistic communication complex*, SIAM J. Comput., 17 (1988), pp. 230–261.

[DH76] D. Diffie and M. Hellman, *New directions in cryptography*, IEEE Trans. Inform. Theory, 22 (1976), pp. 644–654.

[G89] O. Goldreich, *A note on computational indistinguishability*, Inform. Process. Lett., 34 (1990), pp. 277–281.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali, *How to construct random functions*, J. Assoc. Comput. Mach., 33 (1986), pp. 792–807.

[GKL93] O. Goldreich, H. Krawczyk, and M. Luby, *On the existence of pseudorandom generators*, SIAM J. Comput., 22 (1993), pp. 1163–1175.

[GL89] O. Goldreich and L. A. Levin, *A hard-core predicate for any one-way function*, in Proc. 21st ACM Sympos. on Theory of Computing, ACM, New York, 1989, pp. 25–32.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity, or all languages in NP have zero-knowledge proofs*, J. Assoc. Comput. Mach., 38 (1991), pp. 691–729.

[GM84] S. Goldwasser and S. Micali, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems*, SIAM J. Comput., 18 (1989), pp. 186–208.

[GMT82] S. Goldwasser, S. Micali, and P. Tong, *Why and how to establish a private code on a public network*, in Proc. 23rd IEEE Sympos. on Found. of Comput. Sci., IEEE, New York, 1982, pp. 134–144.

[H90] J. Håstad, *Pseudo-random generators under uniform assumptions*, in Proc. 22nd ACM Sympos. on Theory of Computing, ACM, New York, 1990, pp. 395–404.

[HL92] A. Herzberg and M. Luby, *Public randomness in cryptography*, in Advances in Cryptology, Proc. 12th Annual Cryptology Conf. (CRYPTO '92), Santa Barbara, CA, 1992, Lecture Notes in Comput. Sci. 740, Springer-Verlag, Berlin, 1993, pp. 421–432.

[IL89] R. Impagliazzo and M. Luby, *One-way functions are essential for information based cryptography*, in Proc. 30th IEEE Sympos. on Found. of Comput. Sci., IEEE, New York, 1989, pp. 230–235.

[ILL89] R. Impagliazzo, L. Levin, and M. Luby, *Pseudo-random number generation from one-way functions*, in Proc. 21st ACM Sympos. on Theory of Computing, ACM, New York, 1989, pp. 12–24.

[IN96] R. Impagliazzo and M. Naor, *Efficient cryptographic schemes provably as secure as subset sum*, J. Cryptology, 9 (1996), pp. 192–216.

[IR89] R. Impagliazzo and S. Rudich, *Limits on the provable consequences of one-way functions*, in 21st ACM Sympos. on Theory of Computing, ACM, New York, 1989, pp. 44–56.

[IZ89] R. Impagliazzo and D. Zuckerman, *How to recycle random bits*, in Proc. 30th IEEE Sympos. on Found. of Comput. Sci., IEEE, New York, 1989, pp. 248–253.

[Knuth97] D. E. Knuth, *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms,* 3rd ed., Addison-Wesley, Bonn, 1998.

[K65] A. N. Kolmogorov, *Three approaches to the concept of the amount of information*, Problems Inform. Transmission, 1 (1965), pp. 1–7.

[K92] H. Krawczyk, *How to predict congruential generators*, J. Algorithms, 13 (1992), pp. 527–545.

[Levin87] L. A. Levin, *One-way function and pseudorandom generators*, Combinatorica, 7 (1987), pp. 357–363.

[Levin93] L. A. Levin, *Randomness and non-determinism*, J. Symbolic Logic, 58 (1993), pp.1102–1103.

[Luby96] M. Luby, *Pseudorandomness and Cryptographic Applications*, Princeton Computer Science Notes, Princeton University Press, Princeton, NJ, 1996.

[LR88] M. Luby and C. Rackoff, *How to construct pseudorandom permutations from pseudorandom functions*, SIAM J. Comput., 17 (1988), pp. 373–386.

[McEl78] R. J. McEliece, *A Public Key Cryptosystem Based on Algebraic Coding Theory*, DSN Progress report, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 1978.

[McIn87] J. McInnes, *Cryptography Using Weak Sources of Randomness*, Tech. report 194/87, University of Toronto, 1987.

[MR95] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, 1995.

[Naor88] M. Naor, *Bit commitment using pseudorandom generators*, J. Cryptology, 4 (1991), pp. 151–158.

[NY89] M. Naor and M. Yung, *Universal one-way hash functions and their applications*, in Proc. 21st ACM Sympos. on Theory of Computing, ACM, New York, 1989, pp. 33–43.

[OW93] R. Ostrovsky and A. Wigderson, *One-way functions are essential for non-trivial zero-knowledge*, in Proc. 2nd Israel Sympos. on the Theory of Computing and Systems, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 3–17.

[Renyi70] A. Renyi, *Probability Theory*, North–Holland, Amsterdam, 1970.

[RSA78] R. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM, 21 (1978), pp. 120–126.

[R90] J. Rompel, *One-way functions are necessary and sufficient for secure signatures*, in Proc. 22nd ACM Sympos. on Theory of Computing, ACM, New York, 1990, pp. 387–394.

[SV86] M. Santha and U. Vazirani, *Generating quasi-random sequences from slightly-random sources*, J. Comput. System Sci., 33 (1986), pp. 75–87.

[S48] C. Shannon, *A mathematical theory of communication*, Bell System Tech. J., 27 (1948), pp. 379–423; 623–656.

[S83] M. Sipser, *A complexity theoretic approach to randomness*, in Proc. 15th ACM Sympos. on Theory of Computing, ACM New York, 1983, pp. 330–335.

[V87] U. Vazirani, *Towards a strong communication complexity theory or generating quasi-random sequences from two communicating slightly-random sources*, Combinatorica, 7 (1987), pp. 375–392.

[VV85] U. Vazirani and V. Vazirani, *Random polynomial time is equal to slightly-random polynomial time*, in Proc. 26th IEEE Sympos. on Found. of Comput. Sci., IEEE, New York, 1985, pp. 417–428.

[Yao82] A. C. Yao, *Theory and applications of trapdoor functions*, in Proc. 23rd IEEE Sympos. on Found. of Comput. Sci., IEEE, New York, 1982, pp. 80–91.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1397–1413

LOCAL LABELING AND RESOURCE ALLOCATION USING PREPROCESSING*

HAGIT ATTIYA†, HADAS SHACHNAI†, AND TAMI TAMIR†

Abstract. This paper studies the power of nonrestricted preprocessing on a communication graph G, in a synchronous, reliable system. In our scenario, arbitrary preprocessing can be performed on G, after which a sequence of labeling problems has to be solved on different subgraphs of G. We suggest a preprocessing that produces an orientation of G. The goal is to exploit this preprocessing for minimizing the radius of the neighborhood around each vertex from which data has to be collected in order to determine a label. We define a set of labeling problems for which this can be done. The time complexity of labeling a subgraph depends on the topology of the graph G and is always less than $\min\{\chi(G), O((\log n)^2)\}$. On the other hand, we show the existence of a graph for which even unbounded preprocessing does not allow fast solution of a simple labeling problem. Specifically, it is shown that a processor needs to know its $\Omega(\log n/\log\log n)$-neighborhood in order to pick a label.

Finally, we derive some results for the resource allocation problem. In particular, we show that $\Omega(\log n/\log\log n)$ communication rounds are needed if resources are to be fully utilized. In this context, we define the *compact coloring* problem, for which the orientation preprocessing provides fast distributed labeling algorithm. This algorithm suggests efficient solution for the resource allocation problem.

Key words. locality, preprocessing, orientation, labeling, resource allocation, response time

AMS subject classifications. 68P05, 68Q10, 68Q20, 68Q22, 68R10

PII. S0097539795285643

1. Introduction.

The time required to perform certain computations in message-passing systems depends, in many cases, on the *locality* of information, i.e., the distance to which information should be forwarded. Clearly, within t communication rounds, a processor can get information only from processors located within distance t. The study of problems that are local, i.e., in which the value of a processor depends only on its local neighborhood, has attracted much attention, e.g., [13, 16, 12, 18, 11]. This study assumed that processors have no knowledge about the network topology. In many common scenarios, this is not the situation: If the same problem has to be solved many times on different subnetworks of a fixed network G, then it might be worthwhile to conduct some preliminary preprocessing on G.

We study *labeling problems*, in which each processor has to pick a label, subject to some restrictions on the labeling of the whole network. We allow arbitrary preprocessing on G. Afterward, several instances of the same labeling problem need to be solved on different subnetworks G' of G. All processors of G can participate in the algorithm when a particular subnetwork G' is labeling itself, but only the processors of G' have to pick labels. It is assumed that the system is synchronous and operates

*Received by the editors May 4, 1995; accepted for publication (in revised form) October 9, 1997; published electronically April 7, 1999. A preliminary version of this paper appeared in *Distributed Algorithms: Proceedings of the 8th International Workshop,* Terschelling, The Netherlands, September/October 1994, Lecture Notes in Comput. Sci. 857, G. Tel and P. Vitanyi, eds., Springer-Verlag, Berlin, 1994, pp. 194–208. This work was supported by grant 92-0233 from the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel, by the fund for the promotion of research in the Technion, and by Technion VPR funds.

http://www.siam.org/journals/sicomp/28-4/28564.html

†Department of Computer Science, The Technion, Haifa 32000, Israel (hagit@cs.technion.ac.il, hadas@cs.technion.ac.il, tami@cs.technion.ac.il). Part of the work of the third author was done while at IBM T.J. Watson Research Center, Yorktown Heights, NY.

in rounds; there is no bound on message length, and local computation is unlimited. Furthermore, we assume the system is completely reliable. The preprocessing attempts to increase the locality of the problem, that is, decrease the radius of the neighborhood a processor v needs to know in order to pick a label.

The preprocessing we present produces an orientation that assigns priorities to the processors. Later, when a processor has to compute its label in some subgraph G', it considers only processors with higher priorities. We define a parameter that quantifies the quality of these orientations, denoted by $t(G)$. $t(G)$ depends on the topology of G and it is always less than $\min\{\chi(G), O((\log n)^2)\}$.

We define *extendible labeling problems*, in which a labeled graph can be extended by an independent set of vertices to a larger labeled graph without invalidating the original labels. The maximal independent set problem and the $(\Delta+1)$-coloring problem are extendible. We suggest an efficient preprocessing on G which allows us to solve these problems within $t(G)$ rounds on any subgraph of G. We also discuss a distributed randomized preprocessing on G that takes $O((\log n)^2)$ rounds and enables us to solve these problems on any subgraph of G within $O((\log n)^2)$ rounds. This gives a distributed randomized algorithm for *compact coloring*. Bar-Noy et al. [6] have shown that this algorithm provides efficient solutions to the resource allocation problem for a large class of graphs.

We introduce a problem in which processors have to communicate with processors at a nonconstant distance, even after unbounded preprocessing. The problem is *k-dense coloring*, which is a restricted coloring problem. A coloring is k-dense if every vertex with color $c > k$ has a neighbor with color c', $c - k \leq c' \leq c - 1$. Note that validating that a coloring is k-dense requires only checking with the neighbors (i.e., processors that are at distance 1). We prove that there exists a network on which processors must know their $\Omega(\log n/\log\log n)$-neighborhood in order to pick a color. That is, for some networks, even unbounded preprocessing does not allow us to solve the problem locally.

The locality of distributed computations was first studied by Cole and Vishkin, who showed in [9] that a 3-coloring of a ring requires only the knowledge of an $O(\log^* n)$-neighborhood; this bound was shown to be tight by Linial [12]. The more general problem of computing labels locally was studied by Naor and Stockmeyer [16] in the case where no preprocessing is allowed. They present local algorithms for some labeling problems whose validity can be checked locally, and they also show that randomization does not help in making a labeling problem local. In follow-up work, Mayer, Naor, and Stockmeyer [15] consider the amount of initial symmetry-breaking needed in order to solve certain labeling problems.

Other, less related, works studied coloring and the maximal independent set problem in graphs (e.g., Goldberg, Plotkin, and Shannon [11], Szegendy and Vishwanathan [18], and Panconesi and Srinivasan [17]). Another use of graph-theoretic techniques for local algorithms appears in works on sparse partitions [2, 14]. In these works, preprocessing is applied in order to partition a graph into graphs with small diameters. Given such a partition, it is possible to solve the problem locally for each subgraph and then compose the resulting labels. See also the survey by Linial [13], which describes other works on locality in distributed computation.

Preprocessing is very helpful in the context of ongoing problems, such as *resource allocation* [7], where jobs with conflicting resource requirements have to be scheduled efficiently. An instance of the problem is a *communication graph* G. The vertices represent processors, and there is an edge between a pair of processors if they may

compete on a resource. The resource requirements of a processor may vary, and current requirements are represented by a dynamic *conflict graph* C, where the vertices are processors waiting to execute their jobs, and there is an edge between two processors that currently compete on some resource. (Note that $C \subseteq G$.)

We consider a restricted version of the resource allocation problem: A schedule is *k-compact* if, for every waiting processor p_i in every k rounds, either p_i runs or there exists some conflicting neighbor of p_i which runs. This guarantees that p_i is delayed only because one of its conflicting neighbors is running.

The lower bound for the k-dense coloring problem implies that no preprocessing enables a distributed k-compact schedule within less than $\Omega(\log n / \log\log n)$ rounds. We present a distributed algorithm which is μ-compact, where μ is a known upper bound on the execution time of a job; the algorithm uses preprocessing that produces a t-orientation. The response time of our algorithm is $\delta_i \mu + t(G)$, with δ_i the degree of p_i in C.

The resource allocation problem was introduced by Chandy and Misra [7]. In their definition, known as the *dining philosophers* problem, the resource requirements of the processors are static. We consider the dynamic version of the problem, known as the *drinking philosophers* problem. Several algorithms for the drinking philosophers problem are known. Without preprocessing, the best algorithm to date [5] achieves $O(\delta_i \mu + \delta \log n)$ response time, where δ_i is the degree of p_i in C and δ is the maximal degree in C. In contrast, by using preprocessing, our algorithm achieves a response time of $\delta_i \mu + t(G)$. An algorithm that relies on preprocessing (which colors the communication graph to induce priorities between processors) and achieves a response time of $O(\delta^2 \mu)$ was presented in [8].

The usage of a preprocessing that induces an orientation of the conflict graph was first considered in [7]; Barbosa and Gafni [4] present theoretical results concerning the maximal concurrence which may be achieved using orientation. Like our algorithms, in these papers the orientation is used to induce priorities between processors to decrease the waiting time of processors. However, in this work the quality of a graph orientation is measured as the maximal *directed* length in the graph, which corresponds to the maximal *waiting chain* for a particular processor. In contrast, our measure for the quality of an orientation is the maximal *undirected* distance between two processors that are connected by a directed path. This allows us to combine the orientation preprocessing with a local distributed labeling algorithm, such that the resulting waiting time for each processor is bounded by a small constant, although the length of the maximal directed path may be equal to the size of the graph.

The rest of this paper is organized as follows. In section 2 we give some basic definitions. In section 3 we study labeling problems: we derive a lower bound for a labeling problem that also holds for the case where *unbounded* preprocessing is allowed, we introduce the t-orientation preprocessing, and we prove that this preprocessing provides efficient labeling algorithms for certain problems. Section 4 deals with the resource allocation problem: we present the lower bound for k-compact resource allocation as well as a distributed algorithm for μ-compact resource allocation using t-orientation. We conclude in section 5 with some problems, which are left open by our work.

2. Preliminaries.

Model of computation. We consider a distributed message-passing system with n processors $p_1, \ldots, p_n$. The network connecting the processors is modeled as a graph where vertices correspond to processors and there is a bidirectional communication

link between every pair of adjacent processors.

We assume that the system is synchronous and operates in rounds. That is, at the beginning of round $k+1$, each processor receives all the messages sent to it by its neighbors at the end of round k; after some local computation, the processor may send a message to (some or all of) its neighbors. There is no bound on message length, and local computation is unlimited.

Graph-theoretic notions. Consider a directed/undirected graph $G = (V, E)$. For any two vertices $v, u \in V$, let $d(u, v)$ be the undirected distance between v and u in G; note that even if G is directed, the distance is measured on the shortest undirected path in G between v and u. The diameter of the G, $diam(G)$, is $\max_{v,u \in V} d(v, u)$. Given a vertex v, the *r-neighborhood* of v for some integer $r \geq 0$ is the subgraph of G induced by all vertices u such that $d(v, u) \leq r$. The *girth* of G, $g(G)$, is the length of the shortest cycle in G.

A set of vertices $V' \subseteq V$ is an *independent* if no two vertices in V' are adjacent. An independent set is *maximal* if it is not contained in a strictly larger independent set. A *c-coloring* of G is a partition of V into c independent sets. Equivalently, a c-coloring is a mapping $\Psi : V \to \{1, \ldots, c\}$ specifying for each vertex its *color*, such that two adjacent vertices do not have the same color. The *chromatic number* of G, $\chi(G)$, indicates the smallest number c for which G has a c-coloring.

Given a graph G, denote by $\delta(v)$ the degree of the vertex v, i.e., the number of vertices adjacent to it; let Δ be the maximal degree of a vertex in G.

If G is directed, then a vertex v is a *source* in G if it has no incoming edges.

Labeling problems. A *labeling* of a graph $G = (V, E)$ with some alphabet Σ is a mapping $\lambda : V \to \Sigma$. A *labeling problem* $\mathcal{L}$ is a set of labelings. Intuitively, this is the set of labelings that satisfy certain requirements. For example, c-coloring is a labeling problem with $\Sigma = \{1, \ldots, c\}$ and the requirement that for every edge $\langle v, u \rangle$, $\lambda(v) \neq \lambda(u)$.

A distributed algorithm solves a labeling problem $\mathcal{L}$ if, after performing some rounds of communication, each processor picks a label such that the labeling of the graph is in $\mathcal{L}$.

3. Labeling problems.

3.1. A lower bound. We present a labeling problem and prove that every distributed algorithm for solving this problem requires at least $\Omega(\log n / \log \log n)$ rounds, even with unbounded preprocessing. The problem is a restricted coloring problem, where adjacent vertices should have different labels, and, in addition, the labels have to be close to each other. Formally, we have the following definition.

DEFINITION 3.1. *A coloring is* k-dense *for a fixed $k \geq 1$ if every vertex with color $c > k$ has a neighbor with color $c' \in [c - k, c - 1]$.*

Intuitively, in a k-dense coloring of a graph, every vertex with color $c > k$ has at least one neighbor with a smaller color which is relatively close to c; k captures the maximal gap between the colors. Given a labeling of a graph, every vertex v with color c can validate its label by examining its 1-neighborhood: the label is legal if v has no neighbor with label c, and if $c > k$, then v has a neighbor with color c', $c - k \leq c' \leq c - 1$. (This means that k-dense coloring is 1-checkable, in the terminology of [16].)

We now present our lower bound result. The proof shows a graph G and a vertex $v \in G$, such that v must pick different labels in two different subgraphs G_1 and G_2 of G, but v has the same $\frac{1}{2k}(\log n / \log \log n)$-neighborhoods in G_1 and G_2.

The proof uses graphs which have both a large chromatic number and a large girth; the existence of these graphs is guaranteed by the following theorem.

THEOREM 3.2 (Erdös [10]). *For any $n \geq 1$ and ℓ, $4 < \ell < n$, there exists a graph G with n vertices such that $\chi(G) > \frac{1}{2}(\log n)$ and $g(G) > \frac{1}{2}(\log n/\log \ell)$.*

The following is immediate when taking $\ell = \lfloor \log n \rfloor$ in the Theorem 3.2.

COROLLARY 3.3. *For any $n \geq 1$ and $2 \leq k < \frac{1}{2}\log n(1 - 1/\log\log n)$, there exists a graph G with n vertices such that $\chi(G) > \frac{1}{2}(\log n/\log\log n) + k$ and $g(G) > \frac{1}{k}(\log n/\log\log n)$.*

The next lemma shows that the maximal color in a k-dense coloring of a tree is a lower bound on the tree's depth.

LEMMA 3.4. *In every k-dense coloring of a tree T, if there is a vertex v with color c, then there is a vertex at distance at least $\frac{c}{k} - 1$ from v.*

Proof. Since the coloring is k-dense, v must have a neighbor v_1, such that $c(v_1) \geq c - k$. Similarly, v_1 must have a neighbor v_2, such that $c(v_2) \geq c - 2k$, and in general v_{i-1} must have a neighbor v_i with color at least $c - ik$. The path $v, v_1, v_2, \ldots, v_{i-1}$ can be extended to v_i as long as $i \leq (c-1)/k$. Therefore, the length of the path is at least $\frac{c}{k} - 1$. Clearly, this is a simple path. Since T is a tree, there is no other simple path from v to $v_{\lceil \frac{c}{k} \rceil - 1}$. Therefore, $d(v, v_{\lceil \frac{c}{k} \rceil - 1}) \geq \frac{c}{k} - 1$, which proves the lemma. □

We can now prove the main theorem of this section.

THEOREM 3.5. *For every $k > 1$ and $n \geq 1$ such that $k < \frac{1}{2}\log n(1 - 1/\log\log n)$, there is a communication graph G of size n and a subgraph G' of G such that every distributed algorithm that finds a k-dense coloring of G' requires at least $\frac{1}{2k}(\log n/\log\log n)$ rounds.*

Proof. Assume, by way of contradiction, that there exists an algorithm A which finds a k-dense coloring within R rounds such that $R < \frac{1}{2k}(\log n/\log\log n)$. Clearly, within R rounds a vertex knows only about its R-neighborhood, as we see in the following proposition.

PROPOSITION 3.6. *Let G_1 and G_2 be two subgraphs of G, and let v be a vertex of G. If the R-neighborhood of v in G_1 is identical to the R-neighborhood of v in G_2, then v picks the same label when executing A on G_1 and on G_2.*

By Corollary 3.3, there exists a graph G of size n such that $\chi(G) > kR + k$ and $g(G) > 2R$. By this assumption, A finds a k-dense coloring of any subgraph G' of G within R rounds. In particular, A finds a k-dense coloring of G itself. Since $\chi(G) > kR + k$ there exists a vertex v with color $c > kR + k$. Let G' be the R-neighborhood of v in G. Since $g(G) > 2R$ and G' includes only vertices at distance R from v, it follows that G' is a tree. Clearly, v has the same R-neighborhood in G and G'. Therefore, by Proposition 3.6, v is colored c also in G'.

Since G' is a tree, Lemma 3.4 implies that there is vertex at distance $\lceil \frac{c}{k} \rceil - 1$ from v. Hence, $R \geq \frac{c}{k} - 1$. On the other hand, since $c > kR + k$, it follows that $R < \frac{c}{k} - 1$, which is a contradiction. □

Remark. For stating the lower bound in Theorem 3.5 we assume that $k > 1$. For $k = 1$, the existence of a graph G_1 with n vertices, $\chi(G_1) > \frac{1}{2}(\log n/\log\log n) + 1$, and $g(G_1) > \frac{1}{2}(\log n/\log\log n)$, implies, in a similar way, that every distributed algorithm that finds a 1-dense coloring requires at least $\frac{1}{4}(\log n/\log\log n)$ rounds.

3.2. Efficient labeling using *t*-orientation. In this section we define a class of labeling problems, and show a specific preprocessing which allows us to solve them efficiently.

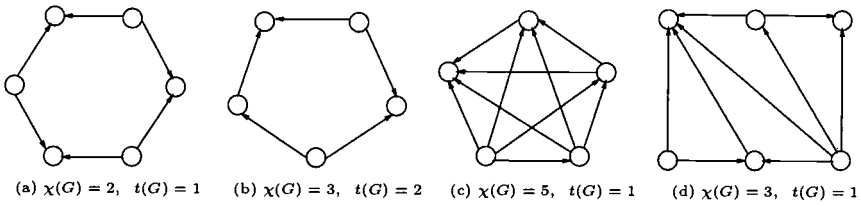


FIG. 3.1. *Optimal t-orientations of some graphs.*

Let $G' \subseteq G$ be a graph that has to be labeled. Clearly, within $diam(G')+1$ rounds, each processor $v \in G'$ can learn G', and therefore can pick a label.[1] Intuitively, the preprocessing presented in this section orients the edges between neighboring processors, thereby assigning priorities in such a way that a processor is close to vertices it is oriented to (i.e., with higher priority). We show that for some problems (including coloring and maximal independent set) there exists a labeling algorithm in which a processor's label depends only on the vertices with higher priority. This allows the processor to communicate only with these vertices, which by assumption are relatively close.

3.2.1. *t*-orientation of graphs. We require an acyclic orientation in which every vertex is close to vertices that have a directed path to it.

DEFINITION 3.7. *A* t-orientation *of a graph G is an acyclic orientation (that is, without any directed cycles) of G, such that for every two vertices v and u, if there is a directed path from v to u in the directed graph, then $d(u, v) \leq t$. The* orientation number *of a graph G, denoted by $t(G)$, is the smallest t such that G has a t-orientation.*

Note that for every graph G, topological sorting implies an acyclic orientation, and therefore, we have the following proposition.

PROPOSITION 3.8. *For every graph G, $t(G) \leq diam(G)$.*

However, in most cases we can do much better. For example, any c-coloring of G implies a $(c-1)$-orientation by directing each edge (v, u) from v to u if and only if $color(v) < color(u)$. This is a $(c-1)$-orientation since all directed paths have length at most c. This implies the following proposition.

PROPOSITION 3.9. *For every graph G, $t(G) < \chi(G)$.*

For example, the orientation number of a ring is 1 if the ring is of even length and 2 if the ring is of odd length (using a 2-coloring or 3-coloring, respectively). Figure 3.1 includes examples of optimal t-orientations for several graphs.

Recall that our definition of t-orientation requires only that the *undirected* distance between any two vertices u and v that are connected by a directed path is bounded by t. We note that Proposition 3.9 holds also for a stronger definition that requires the *directed* distance between u and v to be bounded by t. Therefore, we

[1] Note that if G' is not connected, then $diam(G') = \infty$. For some labeling problems, such as coloring, it is sufficient for a processor to know its connected component in G' in order to pick a label. For these problems, the number of rounds needed in order to label G' is $diam(G'_m)+1$, where G'_m is the connected component with maximal diameter in G'. For other labeling problems, such as finding the number of processors in G', the whole graph G' should be known. For this kind of problem, $diam(G)$ rounds are needed in order to label G'.

expect that the upper bound of $\chi(G)$, as given in Proposition 3.9, can be tightened.[2]

A simple way to construct an optimal t-orientation is by a preprocessing that collects the complete graph topology to some node and then locally finds the best orientation. (This relies on the fact that local computation power is unbounded.) It requires $O(diam(G))$ communication rounds. In the following we show that while a moderate computational effort may not yield an optimal orientation, it allows us to find orientations that are good, in the sense that t is always bounded by a small polylogarithm of n.

THEOREM 3.10. *For every graph G of size n, it is possible to find an $O((\log n)^2)$-orientation of G by a randomized distributed algorithm within $O((\log n)^2)$ rounds.*

Proof. Every graph can be partitioned into $O(\log n)$ subgraphs $V_1, V_2, \ldots$, such that the diameter of every connected component in these subgraphs is at most $O(\log n)$. This is done by the randomized distributed algorithm of Linial and Saks [14] within $O((\log n)^2)$ rounds. At the end of the algorithm, every vertex knows the identity, i, of the subgraph V_i to which it belongs, and the ids of the vertices that belong to its connected component in V_i.

This partition can be used to construct an $O((\log n)^2)$-orientation of G within $O(\log n)$ (additional) rounds as follows. Every connected component of every subgraph is oriented acyclically (e.g., by centralized topological sorting) within $O(\log n)$ rounds. Edges whose endpoints are at different subgraphs are oriented according to the ids of the subgraphs; that is, an edge $\langle v, u\rangle$, with $v \in V_i$ and $u \in V_j$, $i < j$, is oriented $v \to u$.

Clearly, this orientation is acyclic. Furthermore, assume that there is a directed path from v to u. That path visits the subgraphs defined for G in a strictly increasing order; therefore, it visits each subgraph at most once. Since the diameter of every connected component in each subgraph is at most $O(\log n)$, we have $d(v, u) = O((\log n)^2)$. □

3.2.2. Extendible labeling problems. We now define a class of labeling problems for which the t-orientation preprocessing is helpful. These are problems for which the labeling can be constructed by extending the part of the graph which is already labeled.

DEFINITION 3.11. *Let $G = (V, E)$ be a graph. An* extension *of G is a graph $G' = (V \cup V', E \cup E')$, where $V \cap V' = \emptyset$ and $E' \subseteq V \times V'$. Note that V' is an independent set in G'.*

DEFINITION 3.12. *Let $\mathcal{L}$ be a labeling problem. $\mathcal{A}$ is* an extension labeling algorithm for $\mathcal{L}$ *if, for every graph G with a labeling in $\mathcal{L}$ and every extension to $G' = (V \cup V', E \cup E')$, $\mathcal{A}$ gives a label for each $v \in V'$ such that*

- *the labeling of G' is in $\mathcal{L}$;*
- *for each $v \in V'$, the label of v depends only on the connected components of G to which v is connected—that is, the labeling of v is independent of the labeling of other vertices in V' and of the other components of G.*

DEFINITION 3.13. *A labeling problem is* extendible *if it has a deterministic extension labeling algorithm.*

We now argue that some important labeling problems are extendible. Consider the following extension algorithm for a labeling φ, denoted by A_m:

For every $v \in V'$, $\varphi(v) = 0$ if and only if v has a neighbor $u \in V$ with $\varphi(u) = 1$.

[2] The possible gap between $t(G)$ and $\chi(G)$ is well demonstrated in a clique G of n vertices, where $\chi(G) = n$ and $t(G) = 1$.

PROPOSITION 3.14. *Finding a maximal independent set is an extendible labeling problem.*

Proof. Let $G = (V, E)$ be a graph which is legally labeled; i.e., every vertex $v \in V$ has a label $\varphi(v) \in \{0, 1\}$ such that the vertices with $\varphi(v) = 1$ form a maximal independent set of G. Let $G' = (V \cup V', E \cup E')$ be an extension of G. A_m is clearly an extension algorithm for the maximal independent set problem. ∎

PROPOSITION 3.15. *$(\Delta + 1)$-coloring is an extendible labeling problem.*

Proof. Let $G = (V, E)$ be a graph which is legally colored; i.e., every vertex $v \in V$ has a label $\psi(v) \in \{1, \ldots, \Delta + 1\}$, and for every $c \in \{1, \ldots, \Delta + 1\}$ all vertices with $\psi(v) = c$ form an independent set. Let $G' = (V \cup V', E \cup E')$ be an extension of G. The following is clearly an extension algorithm for this problem:

For every $v \in V$, define $\psi(v)$ to be the smallest $c \in \{1, \ldots, \delta(v) + 1\}$ such that no neighbor u of v has $\psi(u) = c$. Such c exists because v has $\delta(v)$ neighbors, and therefore at most $\delta(v)$ colors are used by v's neighbors. For each $v \in V'$, $\delta(v) \leq \Delta$; thus G' is $(\Delta + 1)$-colored. ∎

An extension algorithm that labels a vertex with the smallest color not used by its neighbors is suitable for the k-dense coloring problem. Therefore, we have the following proposition.

PROPOSITION 3.16. *For every $k \geq 1$, the k-dense coloring problem is extendible.*

3.2.3. An algorithm for extendible labeling problems. Here we show the following theorem.

THEOREM 3.17. *Given a t-orientation of a graph G, for any extendible labeling problem $\mathcal{L}$ there is a distributed algorithm that solves $\mathcal{L}$ within t rounds on every subgraph of G.*

Proof. Let $\mathcal{L}$ be an extendible labeling problem, and let $\mathcal{A}$ be a deterministic extension algorithm for $\mathcal{L}$. We describe a distributed algorithm that solves $\mathcal{L}$ on any subgraph of G within t rounds.

Let G be a graph with an acyclic orientation, and let G' be a subgraph of G. Note that the t-orientation of G induces an acyclic orientation of G'. Consider a partition of G' into layers $L_0(G')$, $L_1(G')$, ..., $L_{max}(G')$, where max is the length of the longest directed path in G'. For any $v \in G'$, $v \in L_i(G')$ if and only if the longest directed path to v in G' is of length i. Note that this partition is well defined since G is finite and the orientation is acyclic.

CLAIM 3.1. *Each layer $L_i(G')$ forms an independent set.*

Proof. Let v and u be neighbors in G', such that $v \to u$. Every directed path to v can be extended to u, and in particular the longest path to v can be extended to u. Thus, u belongs to a layer higher than v's layer. ∎

For every vertex $v \in G'$, let $G'_{in}(v)$ be the subgraph of G' induced by v and all the vertices in G' that have a directed path to v. For each $v \in G'$, we partition $G'_{in}(v)$ into the layers $L_0(G'_{in}(v))$, $L_1(G'_{in}(v))$, ..., $L_k(G'_{in}(v))$, where k is the length of the longest directed path in $G'_{in}(v)$. This partition has the following properties:

- If $u \in G'_{in}(v)$, then every directed path to u in G' is in $G'_{in}(v)$; that is, $G'_{in}(u) \subseteq G'_{in}(v)$.
- In particular, if $u \in G'_{in}(v)$, then the longest directed path to u is in $G'_{in}(v)$; therefore, for every i and v, $L_i(G'_{in}(v)) \subseteq L_i(G')$.
- Consequently, if $u \in G'_{in}(v)$, then for every i, $L_i(G'_{in}(u)) \subseteq L_i(G'_{in}(v))$.

The algorithm consists of two stages. In the first stage, information is collected. Specifically, during the first t rounds, every vertex $v \in G'$ distributes to distance t the

$i \leftarrow 0$
Already-labeled $\leftarrow \emptyset$
repeat
    Execute $\mathcal{A}$(Already-labeled , $L_i(G'_{in}(v))$)
    Already-labeled $\leftarrow$ Already-labeled $\cup L_i(G'_{in}(v))$
    $i \leftarrow i + 1$
until v is labeled.

FIG. 3.2. *The labeling algorithm: code for $v \in G'$.*

fact that it belongs to G'. All the vertices of G participate in this stage. Since G is t-oriented, each vertex $v \in G'$ knows $G'_{in}(v)$ within t rounds.

In the second stage of the algorithm, every vertex $v \in G'$ uses $\mathcal{A}$, the extension algorithm, to label $G'_{in}(v)$. The labeling is computed in iterations. In the ith iteration, v labels $L_i(G'_{in}(v))$. The code for $v \in G'$ for this stage appears in Figure 3.2.

We denote by $\mathcal{A}(H, V)$ the application of $\mathcal{A}$ when the labeled graph $H \subseteq G'_{in}(v)$ is extended by an independent set V and all the edges which connect H and V in G'. On each iteration of the **repeat** loop, an additional layer of $G'_{in}(v)$ is labeled. Denote by $label_v(u)$ the label assigned by v to $u \in G'_{in}(v)$, when v executes $\mathcal{A}$. In particular, $label_v(v)$ is the label that v assigns to itself.

The next lemma shows that the labels v assigns to vertices in $G'_{in}(v)$ are identical to the labels those vertices assign to themselves.

LEMMA 3.18. *If $u \in G'_{in}(v)$, then $label_u(u) = label_v(u)$.*

Proof. We show, by induction on $i \geq 0$, that $label_u(u) = label_v(u)$ for every $u \in (G'_{in}(v) \cap L_i(G'))$.

The base case is $i = 0$. Note that $L_0(G')$ contains the sources of G'. Consider some $u \in L_0(G')$ and note that $label_u(u)$ is determined in the first iteration, when u executes $\mathcal{A}(\emptyset, u)$. Every v such that $u \in G'_{in}(v)$ assigns a label to u in the first iteration by executing $\mathcal{A}(\emptyset, L_0(G'_{in}(v)))$. There may be some other vertices in addition to u in $L_0(G'_{in}(v))$, but since the label of u depends only on its connected component which includes only u, and since $\mathcal{A}$ is deterministic, $label_u(u) = label_v(u)$.

For the induction step, assume that the claim holds for all vertices in $L_i(G')$ for $i < j$. Consider $u \in L_j(G')$, and note that $label_u(u)$ is determined in the jth iteration, when u executes $\mathcal{A}(\bigcup_{i<j} L_i(G'_{in}(u)), u)$. Every v such that $u \in G'_{in}(v)$ assigns $label_v(u)$ when it executes $\mathcal{A}(\bigcup_{i<j} L_i(G'_{in}(v)), L_j(G'_{in}(v)))$. The connected component of u in $\bigcup_{i<j} L_i(G'_{in}(v))$ is $\bigcup_{i<j} L_i(G'_{in}(u))$. By the induction assumption, all the vertices of both $\bigcup_{i<j} L_i(G'_{in}(u))$ and $\bigcup_{i<j} L_i(G'_{in}(v))$ are labeled identically by v and by u. Thus, since $\mathcal{A}$ is deterministic, $label_u(u) = label_v(u)$. □

The entire labeling of G' consists of the labels $label_v(v)$. By Lemma 3.18, it is identical to the labeling produced by $\mathcal{A}$ when applied to G' sequentially, layer by layer. Thus, it is in $\mathcal{L}$. □

By Theorem 3.10, we have the following corollary.

COROLLARY 3.19. *For every graph G of size n, after a randomized preprocessing that takes $O((\log n)^2)$ rounds, any extendible labeling problem can be solved on every $G' \subseteq G$ within $O((\log n)^2)$ rounds.*

Note that for the preprocessing suggested in the above results we assume that n is known in advance.

Proposition 3.16 and Theorem 3.17 imply that for every graph G and a fixed $k \geq 1$, a k-dense coloring of every $G' \subseteq G$ can be found distributively within t rounds,

assuming the existence of a t-orientation of G. In particular, by Corollary 3.19, there is a randomized distributed preprocessing that takes $O((\log n)^2)$ rounds, and enables us to find a k-dense coloring of every $G' \subseteq G$, within $O((\log n)^2)$ rounds. Note that the lower bound for k-dense coloring, from Theorem 3.5, is $\Omega(\log\, n/\log\log\, n)$.

Since the k-dense coloring problem is extendible, Theorem 3.5 and Theorem 3.17 imply the following corollary.

COROLLARY 3.20. *Let $t(n)$ be the maximal $t(G)$ among graphs of size n. Then $t(n) = \Omega(\log\, n/\log\log\, n)$.*

4. Resource allocation. In this section we study the resource allocation problem. This problem, in contrast to labeling problems, has an "ongoing" nature and has to be repetitively solved for each instance. However, as will be shown below, we employ techniques and results that were developed for labeling problems.

An instance of the resource allocation problem is a *communication graph* G, where the vertices represent processors, and there is an edge between any pair of processors that may compete on some resource. The resource requirements of a processor may vary. The current requirements are represented formally in a dynamic *conflict graph* C, where the vertices are processors waiting to execute their jobs, and there is an edge between two processors that compete on some resource. Clearly, $C \subseteq G$. We denote the degree of processor p_i in the conflict graph C by δ_i and the maximum number of rounds required to complete a job by μ.

An algorithm for the resource allocation problem decides when each waiting processor can use the resources and execute its job; it should satisfy the following properties:

1. *Exclusion:* No two conflicting jobs are executed simultaneously. (This is a safety property.)
2. *No starvation:* The request of any processor is eventually granted. (This is a liveness property.)

The *response time* for a request is the number of rounds that elapse from the processor's request to use resources until it executes the job. A good algorithm should minimize the response time. We also consider the following property, which guarantees better exploitation of the resources and reduces the average response time.

DEFINITION 4.1. *An algorithm for the resource allocation is* k-compact *for every waiting processor p_i if in every k rounds either p_i runs or some conflicting neighbor of p_i runs.*

In section 4.1 we prove by reduction to the lower bound for k-dense coloring (which was proved earlier) that for every $k \geq 1$ there is no efficient distributed algorithm which is k-compact. Specifically, we show a lower bound of $\Omega(\lg n/\lg\lg n)$ on the response time of any resource allocation algorithm that is k-compact for any $k \geq 1$. Section 4.2 presents the compact coloring problem, which is used later for compact resource allocation. In section 4.3 we present a distributed μ-compact algorithm for resource allocation, which uses the t-orientation preprocessing.

4.1. A lower bound for k-compact resource allocation. We show that given a conflict graph C and $k \geq 1$, any k-compact resource allocation algorithm can be used to label C such that the labeling is a $\lceil \frac{k}{\mu} \rceil$-dense coloring. Together with the lower bound proved in Theorem 3.5 this implies the lower bound for compact resource allocation.

Let G be a communication graph and let C be a conflict graph. The *one-shot resource allocation problem* is to schedule the resources for C in a way that satisfies the safety and liveness conditions. A *slow* execution for a given set of jobs is an

execution where each job uses the resources for exactly μ rounds. (This terminology is borrowed from Rhee [19].)

For a specific algorithm, consider a slow execution with respect to the one-shot resource allocation problem. That is, the algorithm has to schedule only one "batch" of jobs, each of which needs the resources for the same running time, μ. Clearly, this is a special case of the resource allocation problem and any lower bound for this case applies to the general problem.

Let t_0 be the first round in which some processor starts executing its job; the no-starvation property guarantees the existence of t_0. Associate with each processor p_i a label $\lambda(p_i)$ such that $\lambda(p_i) = c$ if and only if p_i starts executing its job in the interval $[t_0 + (c-1)\mu\,,\, t_0 + c\mu)$. Such an interval exists by the no-starvation property, and hence the labeling is well defined.

CLAIM 4.1. *The labeling λ is a $(\lceil \frac{k}{\mu} \rceil + 2)$-dense coloring of the conflict graph.*

Proof. By the mutual exclusion property, and since the execution is slow, λ is a legal coloring.

Assume now that $\lambda(p_i) = c > \lceil \frac{k}{\mu} \rceil + 2$. That is, p_i starts executing its job in the interval $[t_0 + (c-1)\mu\,,\, t_0 + c\mu)$. Since the algorithm is k-compact, in every k rounds, either p_i starts executing its job, or there exists some conflicting processor p_j which executes its job. In the latter case, there is a conflicting processor, p_j, which starts executing its job in the interval $[t_0 + (c-2)\mu - k\,,\, t_0 + (c-1)\mu)$. By the definition of λ, p_j is labeled c', $(c-2) - \frac{k}{\mu} \leq c' \leq c - 1$, as needed. □

Together with Theorem 3.5, this implies the following theorem.

THEOREM 4.2. *For every $k \geq 1$, there is no k-compact distributed algorithm for the resource allocation problem with response time less than $\frac{\mu}{2k+6\mu}(\log n / \log\log n)$.*

4.2. Compact coloring. In this section we introduce the compact coloring problem and its properties. In the next section, we use these properties to show that processors joining the conflict graph C at different times in our algorithm agree on the same colors for processors in C.

DEFINITION 4.3. *A coloring is* compact *if every vertex v with color j has neighbors with all colors $1, \ldots, j-1$.*

Note that every compact coloring is 1-dense. On the other hand, consider a graph that is a line of length 4, whose vertices are colored $1, 2, 3, 4$. This is a 1-dense coloring which is not compact.

For a given compact coloring, let C_i denote the set of vertices colored with i; since the coloring is compact, C_i is a maximal independent set in $V \setminus \bigcup_{j<i} C_j$. Consider the following extension algorithm for a labeling Ψ, denoted by A_c: For every $v \in V$, define $\Psi(v)$ to be the smallest number c such that no neighbor u of v has $\Psi(u) = c$. Clearly the following lemma holds.

LEMMA 4.4. *Compact coloring is an extendible labeling problem.*

The next lemma claims that if we remove all the vertices colored 1 by A_c from a graph G, we obtain a graph G' such that, for every vertex v in G', if v was colored c in G, then v is colored $c-1$ when applying A_c to G'.

LEMMA 4.5. *Let $G = (V, E)$ be a graph, and let $\Psi : V \to N$ be the compact coloring of G produced by A_c. Let $G' = (V', E')$ be the graph obtained by deleting all the vertices for which $\Psi(v) = 1$ from G, and let $\Psi' : V' \to N$ be the compact coloring of G' produced by A_c. Then for every $v \in V'$, $\Psi'(v) = \Psi(v) - 1$.*

Proof. To prove the lemma, we consider the algorithm A_m^* which iteratively executes the maximal independent set (MIS) extension algorithm, A_m, on a given graph G (see Figure 4.1). Recall that A_m labels a vertex v with 1 if v has no neighbor

$V_1 = V$
$E_1 = E$
$i \leftarrow 1$
Repeat
    Execute A_m on $G_i = (V_i, E_i)$
    $\mathrm{M}IS_i \leftarrow \mathrm{M}IS(G_i)$ produced by A_m
    For every $v \in \mathrm{M}IS_i$, $\Phi(v) \leftarrow i$
    $V_{i+1} \leftarrow V_i \setminus \mathrm{M}IS_i$
    $E_{i+1} \leftarrow E_i \setminus \{(u,v) \mid u, v \in \mathrm{M}IS_i\}$
    $i \leftarrow i + 1$
Until G_i is empty

FIG. 4.1. *Algorithm* A_m^*.

from a lower layer which is labeled 1; otherwise, v is labeled with 0. A_m^* is useful for studying A_c, due to the following claim.

CLAIM 4.2. *For every graph G, the labeling Φ produced by A_m^* is identical to the labeling Ψ produced by A_c.*

Proof. Recall that, given an acyclically oriented graph G, a vertex v is in the ith layer, $L_i(G)$, if and only if the longest directed path to v is of length i. The proof is by induction on the layers of G.

For the base case, consider a vertex $v \in L_0(G)$. Note that $L_0(G)$ contains the sources of G. Both A_c and A_m^* color every source v with 1.

For the induction step, assume that the claim holds for all vertices in $L_j(G), j < i$, and let $v \in L_i(G)$. Assume that $\Phi(v) = k$, that is, $v \in \mathrm{M}IS_k$. Consider the neighbors of v from lower layers at the end of iteration k of A_m^* in which v joins $\mathrm{M}IS_k$. By A_c, v is colored k if and only if v has neighbors from lower layers with all colors $\{1, \ldots, k-1\}$, and no neighbor from lower layers which is colored k.

Since every iteration of A_m^* produces a maximal independent set, v has neighbors from lower layers in $\mathrm{M}IS_1, \mathrm{M}IS_2, \ldots, \mathrm{M}IS_{k-1}$. By the induction hypothesis, this implies that v has neighbors which are colored $1, \ldots, k-1$ by A_c. By A_m, v joins $\mathrm{M}IS_k$ if and only if v has no neighbor from lower layers in $\mathrm{M}IS_k$. Thus, by the induction hypothesis, v has no neighbor from a lower layer which is colored k by A_c. Therefore, $\Psi(v) = k$, as needed. □

Proof of Lemma 4.4. By Claim 4.2, $\Phi(v) = \Psi(v)$ for every $v \in G$. In particular, $\mathrm{M}IS_1(G)$ is the set of vertices with $\Psi(v) = 1$. Remove $\mathrm{M}IS_1(G)$ from G. By Claim 4.2, the resulting graph is G'. Let Φ' be the coloring produced by applying A_m^* to G'.

Consider the execution of A_m^* on G. By A_m^*, $\mathrm{M}IS_1(G)$ is removed from G after the first iteration of that execution. Since the resulting graph is G', the remainder of this execution on G is identical to the execution of A_m^* on G'. That is, the execution of A_m^* on G' is identical to the suffix of the execution on G starting from the second iteration. Hence, $v \in \mathrm{M}IS_i(G)$ if and only if $v \in \mathrm{M}IS_{i-1}(G')$. This implies that for every $v \in V'$, $\Phi'(v) = \Phi(v) - 1$. By Claim 4.2, for every $v \in V'$, $\Psi'(v) = \Psi(v) - 1$. □

By repeatedly removing the set of vertices which are colored 1, we obtain the following corollary.

COROLLARY 4.6. *Let $G = (V, E)$ be a graph, and let $\Psi : V \to N$ be the compact coloring of G, produced by the extension algorithm A_c. For a fixed integer $z \geq 0$, let $G' = (V', E')$ be the graph obtained by deleting all the vertices for which $\Psi(v) \in$*

$\{1, \ldots, z\}$ from G, and let $\Psi' : V' \to N$ be the compact coloring of G' produced by A_c. Then for every $v \in V'$, $\Psi'(v) = \Psi(v) - z$.

This corollary is used in our resource allocation algorithm to show that processors joining the conflict graph C at different times agree on the same colors for processors in C.

4.3. A distributed μ-compact resource allocation algorithm. In this section we describe a μ-compact distributed algorithm for the resource allocation problem whose response time is $\delta_i\mu + 2(t(G)+1)$, where $t(G)$ is the orientation number of the communication graph G.

We assume that μ is known in advance and processors can fix *running phases*, each consisting of μ rounds. In addition, processors submit their requests for resources in *entrance phases*, each consisting of $t(G)+1$ rounds. A processor wishing to execute a job waits for the beginning of the next entrance phase and then submits its request. This adds at most $t(G)+1$ rounds to the response time of every request. The partitions of rounds to entrance phases and running phases are identical with respect to all the processors. Therefore, processors submit requests in batches, with $t(G)+1$ rounds between two successive batches.

The algorithm uses a preprocessing which finds an acyclic orientation of G which achieves the orientation number of G; we use $p_i \to p_j$ to denote that p_i is oriented to p_j. The orientation and entrance phases induce an orientation of the dynamic conflict graph C as follows: An edge $\langle p_i, p_j \rangle$ is directed $p_i \Rightarrow p_j$ if p_i requests resources in an earlier entrance phase than p_j or if p_j and p_i request resources in the same entrance phase and $p_i \to p_j$.

For each entrance phase, the processors are partitioned into the following three sets:

1. *Idle.* Processors that do not need resources, and processors that are currently executing their jobs.
2. *Requesting.* Processors that request resources in the current entrance phase.
3. *Waiting.* Processors that requested resources in previous entrance phases and are still waiting for their running phase.

The object of the algorithm is to use the t-orientation in order to merge the requesting processors with the waiting processors in a manner that does not delay the waiting processors and provides short response time for the new requests. The code for processor p_i appears in Figure 4.2. As in section 3.2, we denote by $C_{in}(p_i)$ the subgraph of C such that $p_j \in C_{in}(p_i)$ if and only if there is a directed path $p_j \Rightarrow \cdots \Rightarrow p_i$ in C.

Intuitively, the algorithm proceeds as follows. Each requesting processor p_i transmits its requests and collects the current state of $C_{in}(p_i)$. Upon having the initial state of $C_{in}(p_i)$, denoted by $C^0_{in}(p_i)$, the running phase of p_i is determined by a compact coloring of $C^0_{in}(p_i)$. If p_i is colored k, then p_i executes its job in the kth running phase, counting from the first running phase that begins after the end of the current entrance phase. The waiting processors update the conflict graph and transmit it to the requesting processors. At the beginning of each entrance phase the updated state of $C_{in}(p_i)$ is obtained from the previous state by deleting the set of processors that will begin executing their jobs in the next $t(G)+1$ rounds.

First, we prove that every requesting processor p_i learns about processors that may influence its color during its entrance phase.

LEMMA 4.7. *A requesting processor, p_i, knows $C^0_{in}(p_i)$ within at most $t(G)+1$ rounds after the beginning of its entrance phase.*

Proof. The proof is by induction on the entrance phase. For the base case,

Do every entrance phase:
  If you do not need resources:
    In the next $t(G)+1$ rounds:
      Transmit to your neighbors all the messages you receive.
  In order to execute a job:
    In the next round:
      Receive from your neighbors the part of $C_{in}(p_i)$ which consists of processors who made requests in previous entrance phases.
    In the next $t(G)$ rounds:
      Distribute that part of $C_{in}(p_i)$ and your request.
      Transmit to your neighbors all the messages you receive.
    Construct $C_{in}(p_i)$ by combining the old part you already know with the parts you received in the last $t(G)$ rounds.
    Use A_c to find a compact coloring of $C_{in}(p_i)$.
    If you are colored k, then execute your job in the kth running phase.
    For every $p_j \in C_{in}(p_i)$,
      If p_j is colored k, then p_j executes its job in the kth running phase.
  If you are waiting:
      Update $C_{in}(p_i)$:
        Remove processors that will start executing their job in the next $t(G)+1$ rounds according to your compact coloring.
        Remove processors which are not connected to you anymore.
      Distribute $C_{in}(p_i)$ to your neighbors.
    In the next $t(G)$ rounds:
      Transmit to your neighbors all the messages you receive.

FIG. 4.2. *The distributed algorithm: code for p_i.*

consider a processor p_i that requests resources in the first entrance phase. Directed paths to p_i contain only other processors that request resources in the first entrance phase. Since G was t-oriented, the distance between each processor in $C_{in}(p_i)$ and p_i is at most $t(G)$. Therefore, p_i knows $C_{in}(p_i)$ after at most $t(G)$ rounds.

For the induction step, let p_i be a processor that requests resources in the rth entrance phase, $r > 1$. Let $\rho = p_j \Rightarrow \cdots \Rightarrow p_i$ be a directed path to p_i in C. By the algorithm, no processor that enters with p_i is directed to a processor from an earlier entrance phase. Thus, ρ can be divided into two parts $p_j \Rightarrow \cdots \Rightarrow p_k \Rightarrow p_l \Rightarrow \cdots \Rightarrow p_i$ such that $p_j, \ldots, p_k$ request resources strictly before the rth entrance phase and $p_l, \ldots, p_i$ request resources in the rth entrance phase.

Two successive entrance phases are separated by $t(G)+1$ rounds. Therefore, by the inductive hypothesis, when p_i joins, p_k already knows the path $p_j \Rightarrow \cdots \Rightarrow p_k$. By the algorithm, p_l receives from p_k this part of ρ in the first round of the rth entrance phase. Since the graph is t-oriented, p_i receives messages from all the vertices in $p_l \Rightarrow \cdots \Rightarrow p_i$ within $t(G)$ rounds and can reconstruct ρ. □

The next lemma states that for every $p_j \in C^0_{in}(p_i)$, the assignments of a running phase to p_j as done by p_i and p_j are identical. That is, p_j is colored k in the compact coloring of $C^0_{in}(p_i)$ if and only if p_j is going to execute its job in the kth running phase, counting from the first running phase that begins after the end of p_i's entrance phase.

LEMMA 4.8. *For every requesting processor p_i and for every $p_j \in C^0_{in}(p_i)$, the*

running phase assigned to p_j by p_i is k if and only if p_j executes its job in phase k.

Proof. The proof is by induction on the entrance phase. For the base case, consider a processor p_i that requests resources in the first entrance phase. Since p_j is in $C^0_{in}(p_i)$, p_j also submits requests in the first entrance phase. By Lemma 4.7, p_i knows $C^0_{in}(p_i)$ within $t(G)+1$ rounds. By Lemma 4.4, the compact coloring problem is extendible. Therefore, by Lemma 3.18 (which refers to Algorithm A_c), p_j assigns color k to itself if and only if p_i assigns color k to p_j in $C^0_{in}(p_i)$. Since p_i and p_j start counting from the same round, the running phases are counted identically by p_i and p_j. This implies the lemma.

For the induction step, assume that the induction hypothesis holds for all processors that request resources before the rth entrance phase, and let p_i be a processor that submits requests at the rth entrance phase. By the algorithm, at the first round of phase r, every waiting processor p_l removes from $C_{in}(p_l)$ processors that will start executing their jobs during entrance phase r, according to the compact coloring calculated by p_l. $C_{in}(p_l)$ includes only processors that request resources before the rth entrance phase. Thus, by the induction hypothesis, these updates reflect correctly the current state of $C_{in}(p_l)$. This fact, together with Lemma 4.7, implies that p_i obtains $C^0_{in}(p_i)$ within $t(G)+1$ rounds. Consider a processor $p_j \in C^0_{in}(p_i)$. There are two cases.

Case 1. p_j is a processor that requests resources in entrance phase r. By the definition of $C_{in}(p)$, $C^0_{in}(p_j) \subseteq C^0_{in}(p_i)$, and using Lemma 3.18, p_j assigns color k to itself if and only if p_i assigns color k to p_j in $C^0_{in}(p_i)$.

Case 2. p_j is a processor that requests resources in entrance phase r'. Note that $r > r'$, since $C^0_{in}(p_i)$ does not include any processors that request resources after p_i. Let x be the ratio between the length of one entrance phase and the length of one running phase, that is, $t(G)+1 = x \cdot \mu$. Let s denote the number of the first running phase to begin after the rth entrance phase, and let s' denote the number of the first running phase to begin after the r'th entrance phase, that is, $s' = s - \lfloor x(r-r') \rfloor$. Assume that p_j assigns to itself color c' at entrance phase r'. By the algorithm, p_j will execute its job at running phase $s'+c'$. In addition, during the $r-r'$ entrance phases between r' and r, p_j removes from $C_{in}(p_j)$ all the processors that will start executing their jobs during that period. Formally, all the processors colored by p_j with $1, \ldots, \lfloor x(r-r') \rfloor$ are removed from $C_{in}(p_j)$. The induction hypothesis implies that the updated $C_{in}(p_j)$ at the beginning of the rth entrance phase contains only processors which are still waiting.

By Corollary 4.6, a compact coloring of the updated $C_{in}(p_j)$ assigns color $c = c' - \lfloor x(r-r') \rfloor$ to p_j. Since $C_{in}(p_j) \subseteq C^0_{in}(p_i)$, Lemma 3.18 implies that p_i assigns color c to p_j in $C^0_{in}(p_i)$. Thus, p_i determines that p_j executes its job in running phase number $s + c = s' + \lfloor x(r-r') \rfloor + c = s' + c'$, and that completes the proof. ∎

We can now prove the main properties of the algorithm.

LEMMA 4.9 (safety). *For every two processors p_i and p_j, if $need_i \cap need_j \neq \emptyset$, then p_i and p_j do not run simultaneously.*

Proof. If $need_i \cap need_j \neq \emptyset$, then p_i and p_j are neighbors in C. Assume, without loss of generality, that $p_i \Rightarrow p_j$ and hence $p_j \in C_{in}(p_i)$. Since $C_{in}(p_i)$ is legally colored, p_i and p_j have different colors in $C_{in}(p_i)$. By Lemma 4.8, the running phase that p_i determines for p_j is identical to the running phase that p_j determines for itself. Therefore, p_i and p_j belong to different running phases. Thus, by the algorithm, p_i and p_j do not run simultaneously. ∎

We now show that the schedule becomes μ-compact at most $2(t(G)+1)$ rounds

after a processor initiates a request for resources.

LEMMA 4.10. *For every waiting processor p_i, after the first $2(t(G)+1)$ rounds, in every μ rounds either p_i runs or some conflicting neighbor of p_i runs.*

Proof. By the algorithm, for every waiting processor p_i, the coloring of $C_{in}(p_i)$ is compact. Thus, if p_i is colored c, it has neighbors with all colors $1,\ldots,c-1$. Therefore, there is at least one neighbor of p_i which runs in each of the running phases $1,\ldots,c-1$. The first running phase in this count of the running phases begins at most $2(t(G)+1)$ rounds after the request was initiated by p_i. Hence, after that round the schedule is μ-compact. □

The response time for a processor p_i consists of three components: First, p_i waits for the beginning of the next entrance phase, which takes at most $t(G)+1$ rounds. Then, during the entrance phase, p_i collects $C^0_{in}(p_i)$. By Lemma 4.7, this takes $t(G)+1$ rounds. Finally, p_i waits for its running phase. By the μ-compact property (Lemma 4.10), p_i waits at most δ_i running phases, each taking μ rounds. This implies the following theorem.

THEOREM 4.11. *There exists an algorithm for the resource allocation problem whose response time is $\delta_i\mu + 2(t(G)+1)$.*

Remark. In our algorithm, δ_i captures the number of processors that issued competing resource requests before or simultaneously with p_i. That is, a processor is not delayed because of processors that request resources after it. Note that, in general, it does not mean that the algorithm guarantees a FIFO ordering. Thus, a processor p_j that issued its request later than p_i may execute its job earlier (while p_i is still waiting). This happens only if p_i needs a "popular" resource that was not requested by p_j.

4.4. Discussion. As presented, the algorithm assumes that the system is synchronous and that the local computing power at the processors is unlimited.

First, we remark that the algorithm can easily be changed to work in asynchronous systems by employing a simple synchronizer, such as α [1]. Since our algorithm relies on synchronization only between neighboring processors, synchronizer α allows us to run the algorithm correctly. The details, which are straightforward, are omitted.

Second, we remark that the local computation performed in our algorithm is fairly moderate. The most consuming step is the computation of a compact coloring; this is done by repeated application of A_m, which in turn greedily assigns colors to nodes. Furthermore, this computation can be integrated with the collection of information from neighboring nodes. In this way, the compact coloring is computed in iterations that overlap the iterations in which information is collected; the local computation at each node reduces to choosing a color based on the colors of its neighbors.

5. Conclusions and open problems. This work addressed the power of unrestricted preprocessing, in particular, the t-orientation preprocessing. Several open questions remain.

1. We derive a lower bound on the number of communication rounds needed for a k-compact resource allocation. Is there a lower bound on the number of communication rounds needed for a resource allocation algorithm that guarantees only the safety and liveness properties?
2. Our lower bound for k-dense coloring depends on k, while our upper bound for this problem is the same for all values of k. Can these bounds be tightened? In particular, is there an algorithm for k-dense coloring whose complexity depends on k?

3. We show that the t-orientation preprocessing helps in some labeling problems. Are there other helpful types of preprocessing?
4. We show that $t(G) \leq O((\log n)^2)$ for every graph G of size n, and that there exists a graph G of size n such that $t(G) = \Omega(\log n / \log\log n)$. Can the upper bound be reduced to $O(\log n / \log\log n)$? In particular, is there a distributed algorithm that achieves a better orientation? Is there a nonrandomized distributed algorithm that achieves a good orientation?
5. Is it NP-hard to determine $t(G)$ for a given graph?

Acknowledgments. We would like to thank Roy Meshulam for bringing Erdös' theorem to our attention and for pointing out the existence of graphs with $t(G) = \Omega(\log n / \log\log n)$. We also thank Amotz Bar-Noy for helpful discussions. An anonymous referee provided many comments that improved the presentation.

REFERENCES

[1] B. Awerbuch, *Complexity of network synchronization*, J. ACM, 32 (1985), pp. 804–823.
[2] B. Awerbuch and D. Peleg, *Sparse partitions*, in Proc. IEEE Symposium on Foundation of Computer Science, St. Louis, MO, 1990, pp. 503–513.
[3] B. Awerbuch and M. Saks, *A dining philosophers algorithm with polynomial response time*, in Proc. IEEE Symposium on Foundation of Computer Science, St. Louis, MO, 1990, pp. 65–74.
[4] V. C. Barbosa and E. Gafni, *Concurrency in heavily loaded neighborhood-constrained systems*, ACM Trans. Programming Languages and Systems, 11 (1989), pp. 562–584.
[5] J. Bar-Ilan and D. Peleg, *Distributed resource allocation algorithms*, in Proc. International Workshop on Distributed Algorithms, Haifa, Israel, 1992, pp. 276–291.
[6] A. Bar-Noy, M. Bellare, M. Halldórsson, H. Shachnai, and T. Tamir, *On chromatic sums and distributed resource allocation*, Inform. and Comput., 140 (1998), pp. 183–202.
[7] K. Chandy and J. Misra, *The drinking philosophers problem*, ACM Trans. Programming Languages and Systems, 6 (1984), pp. 632–646.
[8] M. Choy and A. K. Singh, *Efficient fault tolerant algorithms in distributed systems*, in Proc. 24th ACM Symposium on Theory of Computing, Victoria, BC, Canada, 1992, pp. 593–602.
[9] R. Cole and U. Vishkin, *Deterministic coin tossing and accelerating cascades: Micro and macro techniques for designing parallel algorithms*, in Proc. 18th ACM Symposium on Theory of Computing, Berkeley, CA, 1986, pp. 206–219.
[10] P. Erdös, *Graph theory and probability*, Canad. J. Math., 11 (1959), pp. 34–38.
[11] A. Goldberg, S. Plotkin, and G. Shannon, *Parallel symmetry-breaking in sparse graphs*, in Proc. 19th ACM Symposium on Theory of Computing, New York, NY, 1987, pp. 315–324.
[12] N. Linial, *Distributive algorithms—Global solutions from local data*, in Proc. IEEE Symposium on Foundation of Computer Science, Los Angeles, CA, 1987, pp. 331–335.
[13] N. Linial, *Local-Global Phenomena in Graphs*, Technical Report 9, Dept. Computer Science, Hebrew University, Jerusalem, Israel, 1993.
[14] N. Linial and M. Saks, *Decomposing graphs into regions of small diameter*, in Proc. 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, SIAM, Philadelphia, 1991, pp. 320–330.
[15] A. Mayer, M. Naor, and L. Stockmeyer, *Local computations on static and dynamic graphs*, in Proc. 3rd Israel Symposium on Theory and Computing Systems, Tel Aviv, Israel, 1995, pp. 268–278.
[16] M. Naor and L. Stockmeyer, *What can be computed locally?*, in Proc. 25th ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 184–193.
[17] A. Panconesi and A. Srinivasan, *Improved distributed algorithms for coloring and network decomposition problems*, in Proc. 24th ACM Symposium on Theory of Computing, Victoria, BC, Canada, 1992, pp. 581–592.
[18] M. Szegendy and S. Vishwanathan, *Locality based graph coloring*, in Proc. 25th ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 201–207.
[19] I. Rhee, *Efficiency of Partial Synchrony, and Resource Allocation in Distributed Systems*, Ph.D. thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, 1994.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1414–1432

SPACE-EFFICIENT ROUTING TABLES FOR ALMOST ALL NETWORKS AND THE INCOMPRESSIBILITY METHOD*

HARRY BUHRMAN†, JAAP-HENK HOEPMAN‡, AND PAUL VITÁNYI†

Abstract. We use the incompressibility method based on Kolmogorov complexity to determine the total number of bits of routing information for almost all network topologies. In most models for routing, for almost all labeled graphs, $\Theta(n^2)$ bits are necessary and sufficient for shortest path routing. By "almost all graphs" we mean the Kolmogorov random graphs which constitute a fraction of $1-1/n^c$ of all graphs on n nodes, where $c>0$ is an arbitrary fixed constant. There is a model for which the average case lower bound rises to $\Omega(n^2\log n)$ and another model where the average case upper bound drops to $O(n\log^2 n)$. This clearly exposes the sensitivity of such bounds to the model under consideration. If paths have to be short, but need not be shortest (if the stretch factor may be larger than 1), then much less space is needed on average, even in the more demanding models. Full-information routing requires $\Theta(n^3)$ bits on average. For worst-case static networks we prove an $\Omega(n^2\log n)$ lower bound for shortest path routing and all stretch factors <2 in some networks where free relabeling is not allowed.

Key words. computer networks, routing algorithms, compact routing tables, Kolmogorov complexity, incompressibility method, random graphs, average-case complexity, space complexity

AMS subject classifications. 68M10, 68Q25, 68Q30, 68R10, 90B12

PII. S0097539796308485

1. Introduction.

In very large communication networks, like the global telephone network or the Internet connecting the world's computers, the message volume being routed creates bottlenecks, degrading performance. We analyze a tiny part of this issue by determining the optimal space to represent routing schemes in communication networks for almost all static network topologies. The results also give the average space cost over all network topologies.

A universal *routing strategy* for static communication networks will, for every network, generate a *routing scheme* for that particular network. Such a routing scheme comprises a *local routing function* for every node in the network. The routing function of node u returns for every destination $v\neq u$, an edge incident to u on a path from u to v. This way, a routing scheme describes a path, called a *route*, between every pair of nodes u,v in the network. The *stretch factor* of a routing scheme equals the maximum ratio between the length of a route it produces and the shortest path between the endpoints of that route.

It is easy to see that we can do shortest path routing by entering a routing table in each node u, which for each destination node v indicates to what adjacent node w a message to v should be routed first. If u has degree d, it requires a table of at most $n\log d$ bits,[1] and the overall number of bits in all local routing tables never exceeds $n^2\log n$.

*Received by the editors August 20, 1996; accepted for publication (in revised form) June 26, 1997; published electronically April 20, 1999. A preliminary version of this work was presented at the 15th ACM Conference on Principles Distribut. Comput., Philadelphia, PA, 1996. This research was partially supported by the European Union through NeuroCOLT ESPRIT Working Group 8556 and by NWO through NFI project ALADDIN NF 62-376.

http://www.siam.org/journals/sicomp/28-4/30848.html

†CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands (buhrman@cwi.nl, paulv@cwi.nl).

‡Department of Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands (hoepman@cs.utwente.nl).

[1]Throughout, "log" denotes the binary logarithm.

The stretch factor of a routing strategy equals the maximal stretch factor attained by any of the routing schemes it generates. If the stretch factor of a routing strategy equals 1, it is called a *shortest path routing strategy* because then it generates for every graph a routing scheme that will route a message between arbitrary u and v over a shortest path between u and v.

In a *full-information* shortest path routing scheme, the routing function in u must, for each destination v, return all edges incident to u on shortest paths from u to v. These schemes allow alternative, shortest paths to be taken whenever an outgoing link is down.

We consider point to point communication networks on n nodes described by an undirected graph G. The nodes of the graph initially have unique labels taken from a set $\{1, \ldots, m\}$ for some $m > n$. Edges incident to a node v with degree $d(v)$ are connected to *ports*, with fixed labels $1, \ldots, d(v)$, by a so-called port assignment. This labeling corresponds to the minimal local knowledge a node needs to route: (a) a unique identity to determine whether it is the destination of an incoming message, (b) the guarantee that each of its neighbors can be reached over a link connected to exactly one of its ports, and (c) the guarantee that it can distinguish these ports.

1.1. Cost measures for routing tables. The space requirements of a routing scheme are measured as the sum over all nodes of the number of bits needed on each node to encode its routing function. If the nodes are not labeled with $\{1, \ldots, n\}$—the minimal set of labels—we have to add to the space requirement, for each node, the number of bits needed to encode its label. Otherwise, the bits needed to represent the routing function could be appended to the original identity yielding a large label that is not charged for but does contain all necessary information to route.

The cost of representing a routing function at a particular node depends on the amount of (uncharged) information initially there. Moreover, if we are allowed to relabel the graph and change its port assignment before generating a routing scheme for it, the resulting routing functions may be simpler and easier to encode. On a chain, for example, the routing function is much less complicated if we can relabel the graph and number the nodes in increasing order along the chain. We list these assumptions below and argue that each of them is reasonable for certain systems. We start with the options IA, IB, and II for the amount of information initially available at a node:

I. Nodes do not initially know the labels of their neighbors and use ports to distinguish the incident edges. This models the basic system without prior knowledge.
   - IA. The assignment of ports to edges is fixed and cannot be altered. This assumption is reasonable for systems running several jobs where the optimal port assignment for routing may actually be bad for those other jobs.
   - IB. The assignment of ports to edges is free and can be altered before computing the routing scheme (as long as neighboring nodes remain neighbors after reassignment). Port reassignment is justifiable as a local action that usually can be performed without informing other nodes.

II. Nodes know the labels of their neighbors and over which edge to reach them. This information is free. Or, to put it another way, an incident edge carries the same label as the node to which it connects. This model is concerned only with the additional cost of routing messages beyond the immediate neighbors and applies to systems where the neighbors are already known for various

other reasons.[2]

Orthogonal to that, the following three options regarding the labels of the nodes are distinguished:

- α Nodes cannot be relabeled. For large scale distributed systems, relabeling requires global coordination that may be undesirable or simply impossible.
- β Nodes may be relabeled before computing the routing scheme, but the range of the labels must remain $1, \ldots, n$. This model allows a bad distribution of labels to be avoided.
- γ Nodes may be given arbitrary labels before computing the routing scheme, but the number of bits used to store the node's label is added to the space requirements of a node. Destinations are given using the new, complex labels.[3] This model allows us to store additional routing information, e.g., topological information, in the label of a node. This sort of network may be appropriate for centrally designed interconnected networks for multiprocessors and communication networks. A common example of architecture of this type is the binary n-cube network, where the 2^n nodes are labeled with elements of $\{0,1\}^n$ such that there is an edge between each pair of nodes iff their labels differ in exactly one bit position. In this case one can shortest path route using only the labels by successively traversing edges corresponding to flipping successive bits in the positions where source node and destination node differ.

These two orthogonal sets of assumptions, IA, IB, or II and α, β, or γ, define the nine different models we will consider in this paper. We remark that the lower bounds for models without relabeling are less surprising and easier to prove than the bounds for the other models.

1.2. Outline. We determine the optimum space used to represent shortest path routing schemes on almost all labeled graphs, namely, the Kolmogorov random graphs with randomness deficiency at most $c \log n$, which constitute a fraction of at least $1 - 1/n^c$ of all graphs for every fixed constant $c > 0$. These bounds straightforwardly imply the same bounds for the average case over all graphs, provided we choose $c \geq 3$. For an overview of the results, refer to Table 1.1.[4]

We prove that for almost all graphs, $\Omega(n^2)$ bits are necessary to represent the routing scheme, if relabeling is not allowed and nodes know their neighbors (II $\wedge$ α) or nodes do not know their neighbors (IA $\vee$ IB).[5] Partially matching this lower bound, we show that $O(n^2)$ bits are sufficient to represent the routing scheme if the

[2]We do not consider models that give neighbors for free and, at the same time, allow free port assignment. Given a labeling of the edges by the nodes to which they connect, the actual port assignment doesn't matter at all and can in fact be used to represent $d(v) \log d(v)$ bits of the routing function. Namely, each assignment of ports corresponds to a permutation of the ranks of the neighbors—the neighbors at port i move to position i. There are $d(v)!$ such permutations.

[3]In this model it is assumed that a routing function cannot tell valid from invalid labels and that a routing function always receives a valid destination label as input. Requiring otherwise makes the problem harder.

[4]In this table, arrows indicate that the bound for that particular model follows from the bound found by tracing the arrow. In particular, the average-case lower bound for model IA $\wedge$ β is the same as the IA $\wedge$ γ bound found by tracing $\rightarrow$. The reader may have guessed that a ? marks an open question.

[5]We write A $\vee$ B to indicate that the results hold under model A or model B. Similarly, we write A $\wedge$ B to indicate the result holds only if the conditions of both model A and model B hold simultaneously. If only one of the two "dimensions" is mentioned, the other may be taken arbitrarily (i.e., IA is shorthand for (IA $\wedge$ α) $\vee$ (IA $\wedge$ β) $\vee$ (IA $\wedge$ γ)).

TABLE 1.1

Size of shortest path routing schemes, overview of results. The results presented in this paper are quoted with exact constants and asymptotically (with the lower order of magnitude terms suppressed). This table contains only results on shortest path routing, not the other results in this paper.

	No relabeling (α)	*Permutation* (β)	*Free relabeling* (γ)
Worst case—*lower bounds*			
Port assignment free (IB)	$\rightarrow$	$\Omega(n^2 \log n)$ [5]	$n^2/32$ [Thm 4.2]
Neighbors known (II)	$(n^2/9) \log n$ [Thm 4.4]	$\Omega(n^2)$ [4]	$\Omega(n^{7/6})$ [10]
Average case—*upper bounds*			
Port assignment fixed (IA)	$(n^2/2) \log n$ [Thm 3.6]	$\leftarrow$	$\leftarrow$
Port assignment free (IB)	$3n^2$ [Thm 3.1]	$\leftarrow$	$\leftarrow$
Neighbors known (II)	$3n^2$ [Thm 3.1]	$\leftarrow$	$6n \log^2 n$ [Thm 3.2]
Average case—*lower bounds*			
Port assignment fixed (IA)	$(n^2/2) \log n$ [Thm 4.3]	$\rightarrow$	$n^2/32$ [Thm 4.2]
Port assignment free (IB)	$n^2/2$ [Thm 4.1]	$\rightarrow$	$n^2/32$ [Thm 4.2]
Neighbors known (II)	$n^2/2$ [Thm 4.1]	?	?

port assignment may be changed or if nodes do know their neighbors (IB $\vee$ II). In contrast, for almost all graphs, the lower bound rises to asymptotically $(n^2/2) \log n$ bits if both relabeling and changing the port assignment are not allowed (IA $\wedge$ α), and this number of bits is also sufficient for almost all graphs. And, again for almost all graphs, the upper bound drops to $O(n \log^2 n)$ bits if nodes know the labels of their neighbors and nodes may be arbitrarily relabeled (II $\wedge$ γ).

Full-information shortest path routing schemes are shown to require, on almost all graphs, asymptotically $n^3/4$ bits to be stored if relabeling is not allowed (α), and this number of bits is also shown to be sufficient for almost all graphs. (The obvious upper bound for all graphs is n^3 bits.)

For stretch factors larger than 1 we obtain the following results. When nodes know their neighbors (II), for almost all graphs, routing schemes achieving stretch factors s with $1 < s < 2$ can be stored using a total of $O(n \log n)$ bits.[6] Similarly, for almost all graphs in the same models (II), $O(n \log \log n)$ bits are sufficient for routing with stretch factor ≥ 2. Finally, for stretch factors $\geq 6 \log n$ on almost all graphs again in the same model (II), the routing scheme occupies only $O(n)$ bits.

For worst-case static networks we prove, by construction of explicit graphs, an $\Omega(n^2 \log n)$ lower bound on the total size of any routing scheme with stretch factor < 2 if nodes may not be relabeled (α).

The novel incompressibility technique based on Kolmogorov complexity [9] has already been applied in many areas but not so much in a distributed setting. A methodological contribution of this paper is to show how to apply the incompressibility method to obtain results in distributed computing for *almost all* objects concerned, rather than for the *worst-case* object. This hinges on our use of Kolmogorov random graphs in a fixed family of graphs. Our results also hold *averaged over all* objects concerned.

Independent recent work [8, 7] applies Kolmogorov complexity to obtain related *worst-case* results mentioned in the next section. They show, for example, that for each n *there exist* graphs on n nodes which may not be relabeled (α) that require in the worst case $\Omega(n^3)$ bits to store a *full-information* shortest path routing scheme.

[6] For Kolmogorov random graphs which have diameter 2 by Lemma 2.6, routing schemes with $s = 1.5$ are the only ones possible in this range.

We prove for the same model that for *almost all* graphs, full-information routing $n^3/4$ bits in total is necessary and sufficient (asymptotically).

1.3. Related work. Previous upper and lower bounds on the total number of bits necessary and sufficient to store the routing scheme in worst-case static communication networks are due to Peleg and Upfal [10] and Fraigniaud and Gavoille [4].

In [10] it was shown that for any stretch factor $s \geq 1$, the total number of bits required to store the routing scheme for some n-node graph is at least $\Omega(n^{1+1/(2s+4)})$ and that there exist routing schemes for all n-node graphs, with stretch factor $s = 12k+3$, using $O(k^3 n^{1+1/k} \log n)$ bits in total. For example, with stretch factor $s = 15$ we have $k = 1$ and their method guarantees $O(n^2 \log n)$ bits to store the routing scheme. The lower bound is shown in the model where nodes may be arbitrarily relabeled and where nodes know their neighbors (II $\wedge$ γ). Free port assignment in conjunction with a model where the neighbors are known (II), however, cannot be allowed. Otherwise, each node would gain $n \log n$ bits to store the routing function (see footnote 2).

Fraigniaud and Gavoille [4] showed that for stretch factors $s < 2$ there are routing schemes that require a total of $\Omega(n^2)$ bits to be stored in the worst case if nodes may be relabeled by permutation (β). This was improved for shortest path routing by Gavoille and Pérennès [5], who showed that for each $d \leq n$ there are shortest path routing schemes that require a total of $\Omega(n^2 \log d)$ bits to be stored in the worst case for some graphs with maximal degree d if nodes may be relabeled by permutation and the port assignment may be changed (IB $\wedge$ β). This last result is clearly optimal for the worst case both for general networks ($d = \Theta(n)$) and bounded degree networks ($d < n$). In [7] it was shown that for each $d \geq 3$ there are networks for which any routing scheme with stretch factor < 2 requires a total of $\Omega(n^2/\log^2 n)$ bits.

Interval routing on a graph $G = (V, E)$, $V = \{1, \ldots, n\}$, is a routing strategy where for each node i, for each incident edge e of i, a (possibly empty) set of pairs of node labels represents disjoint intervals with wraparound. Each pair indicates the initial edge on a shortest path from i to any node in the interval, and for each node $j \neq i$ there is such a pair. We are allowed to permute the labels of graph G to optimize the interval setting.

Gavoille and Pérennès [5] show that there exist graphs for each bounded degree $d \geq 3$ such that for each interval routing scheme, each of $\Omega(n)$ edges are labeled by $\Omega(n)$ intervals. This shows that interval routing can be worse than straightforward coding of routing tables, which can be trivially done in $O(n^2 \log d)$ bits total. (This improves [7], showing that there exist graphs such that for each interval routing scheme some incident edge on each of $\Omega(n)$ nodes is labeled by $\Omega(n)$ intervals and that for each $d \geq 3$ there are graphs of maximal node degree d such that for each interval routing scheme some incident edge on each of $\Omega(n)$ nodes is labeled by $\Omega(n/\log n)$ intervals.)

Flammini, van Leeuwen, and Marchetti-Spaccamela [3] provide history and background on the compactness (or lack thereof) of interval routing using probabilistic proof methods. To the best of our knowledge, one of the authors of that paper, Jan van Leeuwen, was the first to formulate explicitly the question of what exactly is the minimal size of the routing functions, and he also recently drew our attention to this group of problems.

2. Kolmogorov complexity. The Kolmogorov complexity [6] of x is the length of the *shortest* effective description of x. That is, the *Kolmogorov complexity* $C(x)$ of a finite string x is simply the length of the shortest program, say, in Fortran

(or in Turing machine codes) encoded in binary, which prints x without any input. A similar definition holds conditionally in the sense that $C(x|y)$ is the length of the shortest binary program which computes x given y as input. It can be shown that the Kolmogorov complexity is absolute in the sense of being independent of the programming language up to a fixed additional constant term which depends on the programming language but not on x. We now fix one canonical programming language once and for all as a reference and thereby $C()$.

For the theory and applications, see [9]. Let $x, y, z \in \mathcal{N}$, where $\mathcal{N}$ denotes the natural numbers. Identify $\mathcal{N}$ and $\{0,1\}^*$ according to the correspondence $(0,\epsilon), (1,0), (2,1), (3,00), (4,01), \ldots$. Hence, the length $|x|$ of x is the number of bits in the binary string x. Let $T_1, T_2, \ldots$ be a standard enumeration of all Turing machines. Let $\langle \cdot, \cdot \rangle$ be a standard invertible effective bijection from $\mathcal{N} \times \mathcal{N}$ to $\mathcal{N}$. This can be iterated to $\langle \langle \cdot, \cdot \rangle, \cdot \rangle$.

DEFINITION 2.1. *Let U be an appropriate universal Turing machine such that $U(\langle \langle i, p \rangle, y \rangle) = T_i(\langle p, y \rangle)$ for all i and $\langle p, y \rangle$. The* Kolmogorov complexity *of x given y (for free) is*

$$C(x|y) = \min\{|p| : U(\langle p, y \rangle) = x, p \in \{0,1\}^*\}.$$

2.1. Kolmogorov random graphs. One way to express irregularity or *randomness* of an individual network topology is by a modern notion of randomness like Kolmogorov complexity. A simple counting argument shows that for each y in the condition and each length n, there exists at least one x of length n which is *incompressible* in the sense of $C(x|y) \geq n$; 50% of all x's of length n are incompressible but for one bit ($C(x|y) \geq n-1$), 75% of all x's are incompressible but for two bits ($C(x|y) \geq n-2$), and in general a fraction of $1 - 1/2^c$ of all strings cannot be compressed by more than c bits [9].

DEFINITION 2.2. *Each labeled graph $G = (V, E)$ on n nodes $V = \{1, 2, \ldots, n\}$ can be coded by a binary string $E(G)$ of length $n(n-1)/2$. We enumerate the $n(n-1)/2$ possible edges (u, v) in a graph on n nodes in standard lexicographical order without repetitions and set the ith bit in the string to 1 if the ith edge is present and to 0 otherwise. Conversely, each binary string of length $n(n-1)/2$ encodes a graph on n nodes. Hence we can identify each such graph with its corresponding binary string.*

We define the high complexity graphs in a particular family $\mathcal{G}$ of graphs.

DEFINITION 2.3. *A labeled graph G on n nodes of a family $\mathcal{G}$ of graphs has* randomness deficiency *at most $\delta(n)$ and is called $\delta(n)$-*random *in $\mathcal{G}$ if it satisfies*

$$C(E(G)|n, \delta, \mathcal{G}) \geq \log |\mathcal{G}| - \delta(n). \tag{2.1}$$

In this paper $\mathcal{G}$ is the set of all labeled graphs on n nodes. Then, $\log |\mathcal{G}| = n(n-1)/2$, that is, precisely the length of the encoding of Definition 2.2. In what follows we just say "$\delta(n)$-random" with $\mathcal{G}$ understood.

Elementary counting shows that *a fraction* of at least

$$1 - 1/2^{\delta(n)}$$

of all labeled graphs on n nodes in $\mathcal{G}$ has that high complexity [9].

2.2. Self-delimiting binary strings. We need the notion of self-delimiting binary strings.

DEFINITION 2.4. *We call x a* proper prefix *of y if there is a z such that $y = xz$ with $|z| > 0$. A set $A \subseteq \{0,1\}^*$ is* prefix-free *if no element in A is the proper*

prefix of another element in A. A 1:1 *function* $E : \{0,1\}^* \to \{0,1\}^*$ *(equivalently,* $E : \mathcal{N} \to \{0,1\}^*$*) defines a* prefix-code *if its range is prefix-free. A simple prefix-code we use throughout is obtained by reserving one symbol, say* 0*, as a stop sign and encoding*

$$\bar{x} = 1^{|x|}0x,$$
$$|\bar{x}| = 2|x| + 1.$$

Sometimes we need the shorter prefix-code x'*:*

$$x' = \overline{|x|}x,$$
$$|x'| = |x| + 2\lceil \log(|x| + 1)\rceil + 1.$$

We call $\bar{x}$ *or* x' *a* self-delimiting *version of the binary string* x*. We can effectively recover both* x *and* y *unambiguously from the binary strings* $\bar{x}y$ *or* $x'y$*. For example, if* $\bar{x}y = 111011011$*, then* $x = 110$ *and* $y = 11$*. If* $\bar{x}\bar{y} = 1110110101$*, then* $x = 110$ *and* $y = 1$*. The self-delimiting form* $x' \ldots y'z$ *allows the concatenated binary subdescriptions to be parsed and unpacked into the individual items* $x, \ldots, y, z$*; the code* x' *encodes a separation delimiter for* x *using* $2\lceil \log(|x| + 1)\rceil$ *extra bits, and so on* [9].

2.3. Topological properties of Kolmogorov random graphs. High complexity labeled graphs have many specific topological properties, which seems to contradict their randomness. However, randomness is not "lawlessness" but rather enforces strict statistical regularities, for example, to have diameter exactly 2. Note that randomly generated graphs have diameter 2 with high probability. In another paper [2] two of us explored the relationship between high probability properties of random graphs and properties of individual Kolmogorov random graphs. For this discussion it is relevant to mention that, in a precisely quantified way, *every* Kolmogorov random graph individually possesses all simple properties which hold with high probability for randomly generated graphs.

LEMMA 2.5. *The degree* d *of every node of a* $\delta(n)$*-random labeled graph on* n *nodes satisfies*

$$|d - (n-1)/2| = O\left(\sqrt{(\delta(n) + \log n)n}\right).$$

Proof. Assume that there is a node such that the deviation of its degree d from $(n-1)/2$ is greater than k, that is, $|d - (n-1)/2| > k$. From the lower bound on $C(E(G)|n, \delta, \mathcal{G})$ corresponding to the assumption that G is random in $\mathcal{G}$, we can estimate an upper bound on k, as follows.

In a description of $G = (V, E)$ given n, δ, we can indicate which edges are incident on node i by giving the index of the interconnection pattern (the characteristic sequence of the set $V_i = \{j \in V - \{i\} : (i,j) \in E\}$ in $n-1$ bits, where the jth bit is 1 if $j \in V_i$ and 0 otherwise) in the ensemble of

$$m = \sum_{|d-(n-1)/2|>k} \binom{n-1}{d} \leq 2^n e^{-2k^2/3(n-1)} \tag{2.2}$$

possibilities. The last inequality follows from a general estimate of the tail probability of the binomial distribution with s_n the number of successful outcomes in n experiments with probability of success $p = \frac{1}{2}$. Namely, by Chernoff's bounds, in the form used in [1, 9],

$$\Pr(|s_n - pn| > k) \leq 2e^{-k^2/3pn}. \tag{2.3}$$

To describe G, it then suffices to modify the old code of G by prefixing it with

(i) a description of this discussion in $O(1)$ bits;

(ii) the identity of node i in $\lceil \log(n+1) \rceil$ bits;

(iii) the value of k in $\lceil \log(n+1) \rceil$ bits, possibly adding nonsignificant 0's to pad up to this amount;

(iv) the index of the interconnection pattern in $\log m$ bits (we know n, k, and hence $\log m$); followed by

(v) the old code for G with the bits in the code denoting the presence or absence of the possible edges that are incident on node i deleted.

Clearly, given n we can reconstruct the graph G from the new description. The total description we have achieved is an effective program of

$$\log m + 2\log n + n(n-1)/2 - n + O(1)$$

bits. This must be at least the length of the shortest effective binary program, which is $C(E(G)|n, \delta, \mathcal{G})$, satisfying (2.1). Therefore,

$$\log m \geq n - 2\log n - O(1) - \delta(n).$$

Since we have estimated in (2.2) that

$$\log m \leq n - (2k^2/3(n-1))\log e,$$

it follows that $k \leq \sqrt{\frac{3}{2}(\delta(n) + 2\log n + O(1))(n-1)/\log e}$. □

LEMMA 2.6. *Every $o(n)$-random labeled graph on n nodes has diameter* 2.

Proof. The only graphs with diameter 1 are the complete graphs which can be described in $O(1)$ bits, given n, and hence are not random. It remains to consider $G = (V, E)$ is an $o(n)$-random graph with diameter greater than 2, which contradicts (2.1) from some n onwards.

Let i, j be a pair of nodes with distance greater than 2. Then we can describe G by modifying the old code for G as follows:

(i) a description of this discussion in $O(1)$ bits;

(ii) the identities of $i < j$ in $2\log n$ bits;

(iii) the old code $E(G)$ of G with all bits representing presence or absence of an edge (j, k) between j and each k with $(i, k) \in E$ deleted. We know that all the bits representing such edges must be 0 since the existence of any such edge shows that $(i, k), (k, j)$ is a path of length 2 between i and j, contradicting the assumption that i and j have distance > 2. This way we save at least $n/4$ bits since we save bits for as many edges (j, k) as there are edges (i, k), that is, the degree of i, which is $n/2 \pm o(n)$ by Lemma 2.5.

Since we know the identities of i and j and the nodes adjacent to i (they are in the prefix of code $E(G)$ where no bits have been deleted), we can reconstruct G from this discussion and the new description, given n. Since by Lemma 2.5 the degree of i is at least $n/4$, the new description of G, given n, requires at most

$$n(n-1)/2 - n/4 + O(\log n)$$

bits, which contradicts (2.1) for large n. □

LEMMA 2.7. *Let $c \geq 0$ be a fixed constant, and let G be a $c\log n$-random labeled graph. Then from each node i all other nodes are either directly connected to i or are directly connected to one of the least $(c+3)\log n$ nodes directly adjacent to i.*

Proof. Given i, let A be the set of the least $(c+3)\log n$ nodes directly adjacent to i. Assume by way of contradiction that there is a node k of G that is not directly connected to a node in $A \bigcup \{i\}$. We can describe G as follows:

(i) a description of this discussion in $O(1)$ bits;

(ii) a literal description of i in $\log n$ bits;

(iii) a literal description of the presence or absence of edges between i and the other nodes in $n-1$ bits;

(iv) a literal description of k and its incident edges in $\log n + n - 2 - (c+3)\log n$ bits;

(v) the encoding $E(G)$ with the edges incident with nodes i and k deleted, saving at least $2n-2$ bits.

Altogether the resultant description has

$$n(n-1)/2 + 2\log n + 2n - 3 - (c+3)\log n - 2n + 2$$

bits, which contradicts the $c \log n$-randomness of G by (2.1). □

In the description we have explicitly added the adjacency pattern of node i, which we deleted again later. This zero-sum swap is necessary to be able to unambiguously identify the adjacency pattern of i in order to reconstruct G. Since we know the identities of i and the nodes adjacent to i (they are the prefix where no bits have been deleted), we can reconstruct G from this discussion and the new description, given n.

3. Upper bounds. We give methods to route messages over Kolmogorov random graphs with compact routing schemes. Specifically, we show that in general (on almost all graphs) one can use shortest path routing schemes occupying at most $O(n^2)$ bits. If one can relabel the graph in advance, and if nodes know their neighbors, shortest path routing schemes are shown to occupy only $O(n \log^2 n)$ bits. Allowing stretch factors larger than 1 reduces the space requirements—to $O(n)$ bits for stretch factors of $O(\log n)$.

Let G be an $O(\log n)$-random labeled graph on n nodes. By Lemma 2.7 we know that from each node u we can shortest path route to each other node through the least $O(\log n)$ directly adjacent nodes of u. So we route through node v. Once the message reaches node v, its destination is either node v or a direct neighbor of node v (which is known in node v by assumption). Therefore, routing functions of size $O(n \log\log n)$ bits per node can be used to do shortest path routing. However, we can do better.

THEOREM 3.1. *Let G be an $O(\log n)$-random labeled graph on n nodes. Assume that the port assignment may be changed or nodes know their neighbors (*IB $\vee$ II*). Then for shortest path routing it suffices to have local routing functions stored in $3n$ bits per node. Hence the complete routing scheme is represented by $3n^2$ bits.*

Proof. Let G be as in the statement of the theorem. By Lemma 2.7 we know that from each node u we can route via shortest paths to each node v through the $O(\log n)$ directly adjacent nodes of u that have the least indexes. Assume we route through node v. Once the message has reached node v, its destination is either node v or a direct neighbor of node v (which is known in node v by assumption).

Let $A_0 \subseteq V$ be the set of nodes in G which are not directly connected to u. Let $v_1, \ldots, v_m$ be the $O(\log n)$ least indexed nodes directly adjacent to node u (Lemma 2.7) through which we can shortest path route to all nodes in A_0. For $t = 1, 2 \ldots, l$ define $A_t = \{w \in A_0 - \bigcup_{s=1}^{t-1} A_s : (v_t, w) \in E\}$. Let $m_0 = |A_0|$, and define $m_{t+1} = m_t - |A_{t+1}|$. Let l be the first t such that $m_t < n/\log\log n$. Then we claim that v_t is connected by an edge in E to at least $1/3$ of the nodes not connected by edges in E to nodes $u, v_1, \ldots, v_{t-1}$.

CLAIM 1. $|A_t| > m_{t-1}/3$ *for* $1 \leq t \leq l$.

Proof. Suppose, by way of contradiction, that there exists a least $t \leq l$ such that $||A_t| - m_{t-1}/2| \geq m_{t-1}/6$. Then we can describe G, given n, as follows:

(i) A description of this discussion in $O(1)$ bits;

(ii) a description of nodes u, v_t in $2 \log n$ bits, padded with 0's if necessary;

(iii) a description of the presence or absence of edges incident with nodes $u, v_1, \ldots, v_{t-1}$ in $r = n - 1 + \cdots + n - (t-1)$ bits. This gives us the characteristic sequences of $A_0, \ldots, A_{t-1}$ in V, where a *characteristic sequence* of A in V is a string of $|V|$ bits with, for each $v \in V$, the vth bit equal to 1 if $v \in A$ and the vth bit equal to 0 otherwise;

(iv) a self-delimiting description of the characteristic sequence of A_t in $A_0 - \bigcup_{s=1}^{t-1} A_s$, using Chernoff's bound (2.3), in at most $m_{t-1} - \frac{2}{3}(\frac{1}{6})^2 \, m_{t-1} \log e + O(\log m_{t-1})$ bits;

(v) the description $E(G)$ with all bits corresponding to the presence or absence of edges between v_t and the nodes in $A_0 - \bigcup_{s=1}^{t-1} A_s$ deleted, saving m_{t-1} bits. Furthermore, we also delete all bits corresponding to the presence or absence of edges incident with $u, v_1, \ldots, v_{t-1}$, saving a further r bits.

This description of G uses at most

$$n(n-1)/2 + O(\log n) + m_{t-1} - \frac{2}{3}\left(\frac{1}{6}\right)^2 m_{t-1} \log e - m_{t-1}$$

bits, which contradicts the $O(\log n)$-randomness of G by (2.1), because $m_{t-1} > n/\log\log n$. □

Recall that l is the least integer such that $m_l < n/\log\log n$. We construct the local routing function $F(u)$ as follows:

(i) A table of intermediate routing node entries for all the nodes in A_0 in increasing order. For each node w in $\bigcup_{s=1}^{l} A_s$ we enter in the wth position in the table the unary representation of the least intermediate node v with $(u, v), (v, w) \in E$ followed by a 0. For the nodes that are not in $\bigcup_{s=1}^{l} A_s$ we enter a 0 in their position in the table, indicating that an entry for this node can be found in the second table. By Claim 1, the size of this table is bounded by

$$n + \sum_{s=1}^{l} \frac{1}{3}\left(\frac{2}{3}\right)^{s-1} sn \leq n + \sum_{s=1}^{\infty} \frac{1}{3}\left(\frac{2}{3}\right)^{s-1} sn \leq 4n.$$

(ii) A table with explicitly binary coded intermediate nodes on a shortest path for the ordered set of the remaining destination nodes. Those nodes had a 0 entry in the first table and there are at most $m_l < n/\log\log n$ of them, namely, the nodes in $A_0 - \bigcup_{s=1}^{l} A_s$. Each entry consists of the code of length $\log\log n + O(1)$ for the position in increasing order of a node out of $v_1, \ldots, v_m$ with $m = O(\log n)$ by Lemma 2.7. Hence this second table requires at most $2n$ bits.

The routing algorithm is as follows: The direct neighbors of u are known in node u and are routed without a routing table. If we route from start node u to target node w, which is not directly adjacent to u, then we do the following: If node w has an entry in the first table, then route over the edge coded in unary; otherwise find an entry for node w in the second table.

Altogether, we have $|F(u)| \leq 6n$. Adding another $n-1$ in case the port assignment may be chosen arbitrarily, this proves the theorem with $7n$ instead of $6n$. Slightly

more precise counting and choosing l such that m_l is the first such quantity $< n/\log n$ shows $|F(u)| \leq 3n$. □

If we allow arbitrary labels for the nodes, then shortest path routing schemes of $O(n \log^2 n)$ bits suffice on Kolmogorov random graphs, as witnessed by the following theorem.

THEOREM 3.2. *Let $c \geq 0$ be a constant, and let G be a $c \log n$-random labeled graph on n nodes. Assume that nodes know their neighbors and that nodes may be arbitrarily relabeled ($\mathrm{II} \wedge \gamma$), and we allow the use of labels of $(1 + (c+3)\log n)\log n$ bits. Then we can shortest path route with local routing functions stored in $O(1)$ bits per node (hence the complete routing scheme is represented by $(c+3)n \log^2 n + n \log n + O(n)$ bits).*

Proof. Let c and G be as in the statement of the theorem. By Lemma 2.7 we know that from each node u we can shortest path route to each node w through the first $(c+3)\log n$ directly adjacent nodes $f(u) = v_1, \ldots, v_m$ of u. By Lemma 2.6, G has diameter 2. Relabel G such that the label of node u equals u followed by the original labels of the first $(c+3)\log n$ directly adjacent nodes $f(u)$. This new label occupies $(1 + (c+3)\log n)\log n$ bits. To route from source u to destination v do the following.

If v is directly adjacent to u, we route to v in one step in our model (nodes know their neighbors). If v is not directly adjacent to u, we consider the immediate neighbors $f(v)$ contained in the name of v. By Lemma 2.7, at least one of the neighbors of u must have a label whose original label (stored in the first $\log n$ bits of its new label) corresponds to one of the labels in $f(v)$. Node u routes the message to any such neighbor. This routing function can be stored in $O(1)$ bits. □

Without relabeling, routing using less than $O(n^2)$ bits is possible if we allow stretch factors larger than 1. The next three theorems clearly show a trade-off between the stretch factor and the size of the routing scheme.

THEOREM 3.3. *Let $c \geq 0$ be a constant, and let G be a $c \log n$-random labeled graph on n nodes. Assume that nodes know their neighbors (II). For routing with any stretch factor > 1 it suffices to have $n - 1 - (c+3)\log n$ nodes with local routing functions stored in at most $\lceil \log(n+1) \rceil$ bits per node and $1 + (c+3)\log n$ nodes with local routing functions stored in $3n$ bits per node (hence the complete routing scheme is represented by less than $(3c+20)n \log n$ bits). Moreover, the stretch is at most 1.5.*

Proof. Let c and G be as in the statement of the theorem. By Lemma 2.7 we know that from each node u we can shortest path route to each node w through the first $(c+3)\log n$ directly adjacent nodes $v_1, \ldots, v_m$ of u. By Lemma 2.6, G has diameter 2. Consequently, each node in V is directly adjacent to some node in $B = \{u, v_1, \ldots, v_m\}$. Hence it suffices to select the nodes of B as routing centers and store, in each node $w \in B$, a shortest path routing function $F(w)$ to all other nodes occupying $3n$ bits (the same routing function as constructed in the proof of Theorem 3.1 if the neighbors are known). Nodes $v \in V - B$ route any destination unequal to their own label to some fixed directly adjacent node $w \in B$. Then $|F(v)| \leq \lceil \log(n+1) \rceil + O(1)$, and this gives the bit count in the theorem.

To route from an originating node v to a target node w, the following steps are taken. If w is directly adjacent to v, we route to w in one step in our model. If w is not directly adjacent to v, then we first route in one step from v to its directly connected node in B and then via a shortest path to w. Altogether, this takes either two or three steps, whereas the shortest path has length 2. Hence the stretch factor is at most 1.5, which for graphs of diameter 2 (i.e., all $c \log n$-random graphs by Lemma 2.6) is the

only possibility between stretch factors 1 and 2. This proves the theorem. □

THEOREM 3.4. *Let $c \geq 0$ be a constant, and let G be a $c \log n$-random labeled graph on n nodes. Assume that the nodes know their neighbors (*II*). For routing with stretch factor* 2 *it suffices to have $n-1$ nodes with local routing functions stored in at most $\log \log n$ bits per node and one node with its local routing function stored in $3n$ bits (hence the complete routing scheme is represented by $n \log \log n + 3n$ bits).*

Proof. Let c and G be as in the statement of the theorem. By Lemma 2.6, G has diameter 2. Therefore the following routing scheme has stretch factor 2: Let node 1 store a shortest path routing function. All other nodes store only a shortest path to node 1. To route from an originating node v to a target node w, the following steps are taken: If w is an immediate neighbor of v, we route to w in one step in our model. If not, we first route the message to node 1 in at most two steps and then from node 1 through a node v to node w in, again, two steps. Because node 1 stores a shortest path routing function, either $v = w$ or w is a direct neighbor of v.

Node 1 can store a shortest path routing function in at most $3n$ bits using the same construction as used in the proof of Theorem 3.1 (if the neighbors are known). The immediate neighbors of 1 route either to 1 or directly to the destination of the message. For these nodes, the routing function occupies $O(1)$ bits. For nodes v at distance 2 of node 1 we use Lemma 2.7, which tells us that we can shortest path route to node 1 through the first $(c+3)\log n$ directly adjacent nodes of v. Hence, to represent this edge takes $\log \log n + \log(c+3)$ bits, and hence the local routing function $F(v)$ occupies at most $\log \log n + O(1)$ bits. □

THEOREM 3.5. *Let $c \geq 0$ be a constant, and let G be a $c \log n$-random labeled graph on n nodes. Assume that nodes know their neighbors (*II*). For routing with stretch factor $(c+3)\log n$ it suffices to have local routing functions stored in $O(1)$ bits per node (hence the complete routing scheme is represented by $O(n)$ bits).*

Proof. Let c and G be as in the statement of the theorem. From Lemma 2.7 we know that from each node u we can shortest path route to each node v through the first $(c+3)\log n$ directly adjacent nodes of u. By Lemma 2.6, G has diameter 2. So the local routing function—representable in $O(1)$ bits—is to route directly to the target node if it is a directly adjacent node, otherwise simply traverse the first $(c+3)\log n$ incident edges of the starting node and look in each of the visited nodes to see whether the target node is a directly adjacent node. If so, the message is forwarded to that node, otherwise it is returned to the starting node in order to try the next node. Hence each message for a destination at distance 2 traverses at most $2(c+3)\log n$ edges.

Strictly speaking we do not use routing tables at all. We use the fact that a message can go back and forth several times to a node. The header of the message can code some extra information as a tag "failed." In this case it is possible to describe an $O(1)$ bit size routing function which allows one to extract the destination from the header without knowing about $\log n$, for example, by the use of self-delimiting encoding. □

THEOREM 3.6. *Let G be an $O(\log n)$-random labeled graph on n nodes. Assume that nodes do not know their neighbors and relabeling and changing the port assignment are not allowed (*IA $\wedge$ α*). Then for shortest path routing it suffices that each local routing function uses $(n/2)\log n(1+o(1))$ bits (hence the complete routing scheme uses at most $(n^2/2)\log n(1+o(1))$ bits to be stored).*

Proof. At each node we can give the neighbors by the positions of the 1's in a binary string of length $n-1$. Since each node has at most $n/2 + o(n)$ neighbors by

Lemma 2.5, a permutation of port assignments to neighbors can have Kolmogorov complexity at most $(n/2)\log n(1+o(1))$ [9]. This permutation π describes part of the local routing function by determining, for each direct neighbor, the port through which to route messages for that neighbor. If G is $O(\log n)$-random, then by Theorem 3.1 we require only $O(n)$ bits of additional routing information in each node. Namely, because the assignment of ports (outgoing edges) to direct neighbors is known by permutation π, we can use an additional routing table in $3n$ bits per node to route to the remaining nonneighbor nodes as described in the proof of Theorem 3.1. In total this gives $(n^2/2)\log n(1+o(1))$ bits. □

Our last theorem of this section determines the upper bounds for full-information shortest path routing schemes on Kolmogorov random graphs.

THEOREM 3.7. *For full-information shortest path routing on $o(n)$-random labeled graphs on n nodes where relabeling is not allowed (α), the local routing function occupies at most $n^2/4+o(n^2)$ bits for every node (hence the complete routing scheme takes at most $n^3/4+o(n^3)$ bits to be stored).*

Proof. Since for $o(n)$-random labeled graphs on n the node degree of every node is $n/2+o(n)$ by Lemma 2.5, we can describe in each source node the appropriate outgoing edges (ports) for each destination node by the 1's in a binary string of length $n/2+o(n)$. For each source node it suffices to store at most $n/2+o(n)$ such binary strings corresponding to the nonneighboring destination nodes. In each node we can give the neighbors by the positions of the 1's in a binary string of length $n-1$. Moreover, in each node we can give the permutation of port assignments to neighbors in $(n/2)\log n(1+o(1))$ bits. This leads to a total of at most $(n^2/4)(1+o(1))$ bits per node and hence to $(n^3/4)(1+o(1))$ bits to store the overall routing scheme. □

4. Lower bounds. The first two theorems of this section together show that $\Omega(n^2)$ bits are indeed necessary to route on Kolmogorov random graphs in all models we consider, except for the models where nodes know their neighbors *and* label permutation or relabeling is allowed (II $\wedge$ β or II $\wedge$ γ). Hence the upper bound in Theorem 3.1 is tight up to order of magnitude.

THEOREM 4.1. *For shortest path routing in $o(n)$-random labeled graphs where relabeling is not allowed and nodes know their neighbors (*II $\wedge$ α*), each local routing function must be stored in at least $n/2-o(n)$ bits per node (hence the complete routing scheme requires at least $n^2/2-o(n^2)$ bits to be stored).*

Proof. Let G be an $o(n)$-random graph. Let $F(u)$ be the local routing function of node u of G, and let $|F(u)|$ be the number of bits used to store $F(u)$. Let $E(G)$ be the standard encoding of G in $n(n-1)/2$ bits as in Definition 2.2. We now give another way to describe G using some local routing function $F(u)$:

(i) a description of this discussion in $O(1)$ bits;

(ii) a description of u in exactly $\log n$ bits, padded with 0's if necessary;

(iii) a description of the presence or absence of edges between u and the other nodes in V in $n-1$ bits;

(iv) a self-delimiting description of $F(u)$ in $|F(u)|+2\log|F(u)|$ bits;

(v) the code $E(G)$ with all bits deleted corresponding to edges $(v,w)\in E$ for each v and w such that $F(u)$ routes messages to w through the least intermediary node v. This saves at least $n/2-o(n)$ bits since there are at least $n/2-o(n)$ nodes w such that $(u,w)\notin E$ by Lemma 2.5, and since the diameter of G is 2 by Lemma 2.6, there is a shortest path $(u,v),(v,w)$ for some v. Furthermore, we delete all bits corresponding to the presence or absence of edges between u and the other nodes in V, saving another $n-1$ bits. This corresponds to the $n-1$ bits for edges connected to u, which we

added in one connected block (item (iii)) above.

In the description, we have explicitly added the adjacency pattern of node u, which we deleted elsewhere. This zero-sum swap is necessary to be able to unambiguously identify the adjacency pattern of u in order to reconstruct G given n, as follows. Reconstruct the bits corresponding to the deleted edges using u and $F(u)$ and subsequently insert them in the appropriate positions of the remnants of $E(G)$. We can do so because these positions can be simply reconstructed in increasing order. In total this new description has

$$n(n-1)/2 + O(1) + O(\log n) + |F(u)| - n/2 + o(n)$$

bits, which must be at least $n(n-1)/2 - o(n)$ by (2.1). Hence, $|F(u)| \geq n/2 - o(n)$, which proves the theorem. □

THEOREM 4.2. *Let G be an $o(n)$-random labeled graph on n nodes. Assume that the neighbors are not known (*IA $\vee$ IB*) but relabeling is allowed (γ). Then for shortest path routing the complete routing scheme requires at least $n^2/32 - o(n^2)$ bits to be stored.*

Proof. In the proof of this theorem we need the following combinatorial result.

CLAIM 2. *Let k and n be arbitrary natural numbers such that $1 \leq k \leq n$. Let x_i for $1 \leq i \leq k$ be natural numbers such that $x_i \geq 1$. If $\sum_{i=1}^{k} x_i = n$, then*

$$\sum_{i=1}^{k} \lceil \log x_i \rceil \leq n - k.$$

Proof. The proof is by induction on k. If $k = 1$, then $x_1 = n$, and clearly $\lceil \log n \rceil \leq n - 1$ if $n \geq 1$. Supposing the claim holds for k and arbitrary n and x_i, we now prove it for $k' = k + 1$, n, and arbitrary x_i. Let $\sum_{i=1}^{k'} x_i = n$. Then $\sum_{i=1}^{k} x_i = n - x_{k'}$. Now

$$\sum_{i=1}^{k'} \lceil \log x_i \rceil = \sum_{i=1}^{k} \lceil \log x_i \rceil + \lceil \log x_{k'} \rceil.$$

By the induction hypothesis the first term on the right-hand side is less than or equal to $n - x_{k'} - k$, so

$$\sum_{i=1}^{k'} \lceil \log x_i \rceil \leq n - x_{k'} - k + \lceil \log x_{k'} \rceil = n - k' + \lceil \log x_{k'} \rceil + 1 - x_{k'}.$$

Clearly $\lceil \log x_{k'} \rceil + 1 \leq x_{k'}$ if $x_{k'} \geq 1$, which proves the claim. □

Recall that in model γ each router must be able to output its own label. Using the routing scheme we can enumerate the labels of all nodes. If we cannot enumerate the labels of all nodes using less than $n^2/32$ bits of information, then the routing scheme requires at least that many bits of information and we are done. So assume we can (this includes models α and β, where the labels are not charged for, but can be described using $\log n$ bits). Let G be an $o(n)$-random graph.

CLAIM 3. *Given the labels of all nodes, we can describe the interconnection pattern of a node u using the local routing function of node u plus an additional $n/2 + o(n)$ bits.*

Proof. Apply the local routing function to each of the labels of the nodes in turn (these are given by assumption). This will return for each edge a list of destinations reached over that edge. To describe the interconnection pattern, it remains to encode for each edge which of the destinations reached is actually its immediate neighbor. If edge i routes x_i destinations, this will cost $\lceil \log x_i \rceil$ bits. By Lemma 2.5 the degree of a node in G is at least $n/2 - o(n)$. Then in total, $\sum_{i=1}^{n/2-o(n)} \lceil \log x_i \rceil$ bits will be sufficient; separations need not be encoded because they can be determined using the knowledge of all x_i's. Using Claim 2 finishes the proof. □

Now we show that there are $n/2$ nodes in G whose local routing function requires at least $n/8 - 3\log n$ bits to describe (which implies the theorem).

Assume, by way of contradiction, that there are $n/2$ nodes in G whose local routing function requires at most $n/8 - 3\log n$ bits to describe. Then we can describe G as follows:

(i) a description of this discussion in $O(1)$ bits;

(ii) the enumeration of all labels in at most $n^2/32$ (by assumption);

(iii) a description of the $n/2$ nodes in this enumeration in at most n bits;

(iv) the interconnection patterns of these $n/2$ nodes in $n/8 - 3\log n$ plus $n/2 + o(n)$ bits each (by assumption and using Claim 3); this amounts to $n/2(5n/8 - 3\log n) + o(n^2)$ bits in total with separations encoded in another $n \log n$ bits;

(v) the interconnection patterns of the remaining $n/2$ nodes *only among themselves* using the standard encoding, in $1/2(n/2)^2$ bits.

This description altogether uses

$$\begin{aligned} &O(1) + n^2/32 + n + n/2(5n/8 - 3\log n) \\ &\quad + o(n^2) + n\log n + 1/2(n/2)^2 \\ &= n^2/2 - n^2/32 + n + o(n^2) - n/2\log n \end{aligned}$$

bits, contradicting the $o(n)$-randomness of G by (2.1). We conclude that on at least $n/2$ nodes, a total of $n^2/16 - o(n^2)$ bits are used to store the routing scheme. □

If neither relabeling nor changing the port assignment is allowed, the next theorem implies that for shortest path routing on almost all such "static" graphs one cannot do better than storing part of the routing tables literally, in $(n^2/2)\log n$ bits. Note that it is known [5] that there are *worst-case* graphs (even in models where relabeling is allowed) such that $n^2 \log n - O(n^2)$ bits are required to store the routing scheme, and this matches the trivial upper bound for all graphs exactly. But in our Theorem 4.3 we show that in a certain restricted model for *almost all* graphs asymptotically $(n^2/2)\log n$ bits are required and by Theorem 3.6 that many bits are also sufficient.

THEOREM 4.3. *Let G be an $o(n)$-random labeled graph on n nodes. Assume that nodes do not know their neighbors and relabeling and changing the port assignment are not allowed (*IA $\wedge$ α*). Then for shortest path routing each local routing function must be stored in at least $(n/2)\log n - O(n)$ bits per node (hence the complete routing scheme requires at least $(n^2/2)\log n - O(n^2)$ bits to be stored).*

Proof. If the graph cannot be relabeled and the port assignment cannot be changed, the adversary can set the port assignment of each node to correspond to a permutation of the destination nodes. Since each node has at least $n/2 - o(n)$ neighbors by Lemma 2.5, such a permutation can have Kolmogorov complexity as high as $(n/2)\log n - O(n)$ [9]. Because the neighbors are not known, the local routing function must determine, for each neighbor node, the port through which to route messages for that neighbor node. Hence the local routing function completely describes the

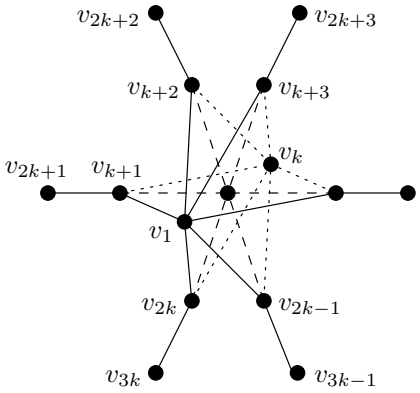


FIG. 4.1. *Graph* G_k.

permutation, given the neighbors, and thus it must occupy at least $(n/2)\log n - O(n)$ bits per node. □

Note that in this model (IA $\wedge$ α) we can trivially find by the same method a lower bound of $n^2 \log n - O(n^2)$ bits for specific graphs like the complete graph and this matches exactly the trivial upper bound in the worst case. However, Theorem 4.3 shows that for this model for *almost all* labeled graphs asymptotically 50% of this number of bits of total routing information is both necessary and sufficient.

Even if stretch factors between 1 and 2 are allowed, the next theorem shows that $\Omega(n^2 \log n)$ bits are necessary to represent the routing scheme in the worst case.

THEOREM 4.4. *For routing with stretch factor* < 2 *in labeled graphs where relabeling is not allowed* (α)*, there exist graphs on* n *nodes (almost* $(n/3)!$ *such graphs) where the local routing function must be stored in at least* $(n/3)\log n - O(n)$ *bits per node at* $n/3$ *nodes (hence the complete routing scheme requires at least* $(n^2/9)\log n - O(n^2)$ *bits to be stored).*

Proof. Consider the graph G_k with $n = 3k$ nodes depicted in Figure 4.1. Each node v_i in $v_{k+1}, \ldots, v_{2k}$ is connected to v_{i+k} and to each of the nodes $v_1, \ldots, v_k$. Fix a labeling of the nodes $v_1, \ldots, v_{2k}$ with labels from $\{1, \ldots, 2k\}$. Then any labeling of the nodes $v_{2k+1}, \ldots, v_{3k}$ with labels from $\{2k+1, \ldots, 3k\}$ corresponds to a permutation of $\{2k+1, \ldots, 3k\}$ and vice versa.

Clearly, for any two nodes v_i and v_j with $1 \le i \le k$ and $2k+1 \le j \le 3k$, the shortest path from v_i to v_j passes through node v_{j-k} and has length 2, whereas any other path from v_i to v_j has length at least 4. Hence any routing function on G_k with stretch factor < 2 routes such v_j from v_i over the edge (v_i, v_{j-k}). Then at each of the k nodes $v_1, \ldots, v_k$ the local routing functions corresponding to any two labelings of the nodes $v_{2k+1}, \ldots, v_{3k}$ are different. Hence each representation of a local routing function at the k nodes v_i, $1 \le i \le k$, corresponds one-to-one to a permutation of $\{2k+1, \ldots, 3k\}$. So given such a local routing function, we can reconstruct the permutation (by collecting the response of the local routing function for each of the nodes $k+1, \ldots, 3k$ and grouping all pairs reached over the same edge). The number of such permutations is $k!$. A fraction of at least $1 - 1/2^k$ of such permutations π has Kolmogorov complexity $C(\pi) = k \log k - O(k)$ [9]. Because π can be reconstructed given any of the k local routing functions, these k local routing functions must each have Kolmogorov complexity $k \log k - O(k)$, too. This proves the theorem for n a multiple of 3. For $n = 3k-1$ or $n = 3k-2$ we can use G_k, dropping v_k and v_{k-1}.

Note that the proof requires only that there be no relabeling; apart from that the direct neighbors of a node may be known and ports may be reassigned.

By the above calculation there are at least $(1 - 1/2^{n/3})(n/3)!$ labeled graphs on n nodes for which the theorem holds. □

Our last theorem shows that for full-information shortest path routing schemes on Kolmogorov random graphs one cannot do better than the trivial upper bound.

THEOREM 4.5. *For full-information shortest path routing on $o(n)$-random labeled graphs on n nodes where relabeling is not allowed (α), the local routing function occupies at least $n^2/4 - o(n^2)$ bits for every node (hence the complete routing scheme requires at least $n^3/4 - o(n^3)$ bits to be stored).*

Proof. Let G be a graph on nodes $\{1, 2, \ldots, n\}$ satisfying (2.1) with $\delta(n) = o(n)$. Then we know that G satisfies Lemmas 2.5 and 2.6. Let $F(u)$ be the local routing function of node u of G, and let $|F(u)|$ be the number of bits used to encode $F(u)$. Let $E(G)$ be the standard encoding of G in $n(n-1)/2$ bits as in Definition 2.2. We now give another way to describe G using some local routing function $F(u)$:

(i) a description of this discussion in $O(1)$ bits;

(ii) a description of u in $\log n$ bits (if it is less, pad the description with 0's);

(iii) a description of the presence or absence of edges between u and the other nodes in V in $n - 1$ bits;

(iv) a description of $F(u)$ in $|F(u)| + O(\log |F(u)|)$ bits (the logarithmic term to make the description self-delimiting);

(v) the code $E(G)$ with all bits deleted corresponding to the presence or absence of edges between each w and v such that v is a neighbor of u and w is not a neighbor of u. Since there are at least $n/2 - o(n)$ nodes w such that $(u, w) \notin E$ and at least $n/2 - o(n)$ nodes v such that $(u, v) \in E$, by Lemma 2.5, this saves at least $(n/2 - o(n))^2$ bits.

From this description we can reconstruct G, given n, by reconstructing the bits corresponding to the deleted edges from u and $F(u)$ and subsequently inserting them in the appropriate positions to reconstruct $E(G)$. We can do so because $F(u)$ represents a full-information routing scheme implying that $(v, w) \in E$ iff (u, v) is among the edges used to route from u to w. In total this new description has

$$n(n-1)/2 + O(\log n) + |F(u)| - n^2/4 + o(n^2)$$

bits, which must be at least $n(n-1)/2 - o(n)$ by (2.1). We conclude that $|F(u)| = n^2/4 - o(n^2)$, which proves the theorem.

Note that the proof requires only that there be no relabeling; apart from that the direct neighbors of a node may be known and ports may be reassigned. □

5. Average case. What about the average cost, taken over all labeled graphs of n nodes, of representing a routing scheme for graphs over n nodes? The results above concerned precise overwhelmingly large fractions of the set of all labeled graphs. The numerical values of randomness deficiencies and bit costs involved show that these results are actually considerably stronger than the corresponding average case results which are straightforward.

DEFINITION 5.1. *For each labeled graph G, let $T_S(G)$ be the minimal total number of bits used to store a routing scheme of type S (where S indicates shortest path routing, full-information routing, and the like). The* average *minimal total number of bits to store a routing scheme for S-routing over labeled graphs on n nodes is $\sum T_S(G)/2^{n(n-1)/2}$ with the sum taken over all graphs G on nodes $\{1, 2, \ldots, n\}$. (That is, the uniform average over all the labeled graphs on n nodes.)*

The results on Kolmogorov random graphs above have the following corollaries. The set of $(3\log n)$-random graphs constitutes a fraction of at least $(1-1/n^3)$ of the set of all graphs on n nodes. The trivial upper bound on the minimal total number of bits for all routing functions together is $O(n^2\log n)$ for shortest path routing on all graphs on n nodes (or $O(n^3)$ for full-information shortest path routing). Simple computation shows that the average total number of bits to store the routing scheme for graphs of n nodes is (asymptotically and ignoring lower order of magnitude terms as in Table 1.1) as follows:

1. $\leq 3n^2$ for shortest path routing in model IB $\vee$ II (Theorem 3.1);
2. $\leq 6n\log^2 n$ for shortest path routing in model II $\wedge$ γ, where the average is taken over the initially labeled graphs on n nodes with labels in $\{1,2,\ldots,n\}$ *before* they were relabeled with new and longer labels giving routing information (Theorem 3.2);
3. $\leq 38n\log n$ for routing with any stretch factor s for $1<s<2$ in model II (Theorem 3.3);
4. $\leq n\log\log n$ for routing with stretch factor 2 in model II (Theorem 3.4);
5. $O(n)$ for routing with stretch factor $6\log n$ in model II (Theorem 3.5 with $c=3$);
6. $\geq n^2/2$ for shortest path routing in model α (Theorem 4.1);
7. $\geq n^2/32$ for shortest path routing in model IA and IB (under all relabeling conventions, Theorem 4.2);
8. $=(n^2/2)\log n$ for shortest path routing in model IA $\wedge$ α (Theorems 3.6 and 4.3);
9. $=n^3/4$ for full-information shortest path routing in model α (Theorems 3.7 and 4.5).

6. Conclusion. The space requirements for compact routing for almost all labeled graphs on n nodes, and hence for the average case of all graphs on n nodes, are conclusively determined in this paper. We introduce a novel application of the incompressibility method. The next question arising in compact routing is the following: For practical purposes the class of all graphs is too broad in that most graphs have high node degree (around $n/2$). Such high node degrees are unrealistic in real communication networks for large n. So the question that arises is: How do we extend the current treatment to almost all graphs on n nodes of maximal node degree d, where d ranges from $O(1)$ to n? Clearly, for shortest path routing $O(n^2\log d)$ bits suffice, and [5] showed that for each $d<n$ there are shortest path routing schemes that require a total of $\Omega(n^2\log d)$ bits to be stored in the worst case for some graphs with maximal degree d, where we allow that nodes are relabeled by permutation and the port assignment may be changed (IB $\wedge$ β). This does not hold for average routing, since by our Theorem 3.1 $O(n^2)$ bits suffice for $d=\Theta(n)$. (Trivially, $O(n^2)$ bits suffice for routing in every graph with $d=O(1)$.) We believe it may be possible to show by an extension of our method that $\Theta(n^2)$ bits (independent of d) are necessary and sufficient for shortest path routing in almost all graphs of maximum node degree d, provided d grows unboundedly with n.

Another research direction is to resolve the questions addressed in this paper for Kolmogorov random unlabeled graphs, in particular with respect to the free relabeling model (insofar as they do not follow a fortiori from the results presented here).

Acknowledgments. We thank Jan van Leeuwen, Evangelos Kranakis, and Danny Krizanc for helpful discussions and the anonymous referees for comments and corrections.

REFERENCES

[1] L.G. Valiant and D. Angluin, *Fast probabilistic algorithms for Hamiltonian circuits and matchings*, J. Comput. System Sci., 18 (1979), pp. 155–193.

[2] H.M. Buhrman, M. Li, J. Tromp, and P.M.B. Vitányi, *Kolmogorov random graphs and the incompressibility method*, SIAM J. Comput., to appear.

[3] M. Flammini, J. van Leeuwen, and A. Marchetti-Spaccamela, *The complexity of interval routing on random graphs*, in Proceedings 20th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 969, Springer-Verlag, Heidelberg, 1995, pp. 37–49.

[4] P. Fraigniaud and C. Gavoille, *Memory requirement for universal routing schemes*, in Proceedings 14th Annual ACM Symposium on Principles of Distributed Computing, ACM Press, New York, 1995, pp. 223–230.

[5] C. Gavoille and S. Pérennès, *Memory requirements for routing in distributed networks*, in Proceedings 15th Annual ACM Symposium on Principles of Distributed Computing, ACM Press, New York, 1996, pp. 125–133.

[6] A.N. Kolmogorov, *Three approaches to the quantitative definition of information*, Problems Inform. Transmission, 1 (1965), pp. 1–7.

[7] E. Kranakis and D. Krizanc, *Lower bounds for compact routing schemes*, in Proceedings Thirteenth Symposium on Theoretical Aspects in Computer Science, Lecture Notes in Comput. Sci. 1046, Springer-Verlag, Heidelberg, 1996, pp. 529–540.

[8] E. Kranakis, D. Krizanc, and J. Urrutia, *Compact routing and shortest path information*, in Proceedings Second International Colloquium on Structural Information and Communication Complexity, vol. 2, Carleton University Press, Ottawa, 1996, pp. 101–112.

[9] M. Li and P.M.B. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd ed., Springer-Verlag, New York, 1997.

[10] D. Peleg and E. Upfal, *A trade-off between space and efficiency for routing tables*, J. ACM, 36 (1989), pp. 510–530.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1433–1459

COMPUTING WITH VERY WEAK RANDOM SOURCES*

ARAVIND SRINIVASAN† AND DAVID ZUCKERMAN‡

Abstract. We give an efficient algorithm to extract randomness from a very weak random source using a small additional number t of truly random bits. Our work extends that of Nisan and Zuckerman [*J. Comput. System Sci.,* 52 (1996), pp. 43–52] in that t remains small even if the entropy rate is well below constant. A key application of this is in running randomized algorithms using such a very weak source of randomness. For any fixed $\gamma > 0$, we show how to simulate RP algorithms in time $n^{O(\log n)}$ using the output of a δ-source with min-entropy R^γ. Such a weak random source is asked once for R bits; it outputs an R-bit string according to any probability distribution that places probability at most 2^{-R^γ} on each string. If $\gamma > 1/2$, our simulation also works for BPP; for $\gamma > 1 - 1/(k+1)$, our simulation takes time $n^{O(\log^{(k)} n)}$ ($\log^{(k)}$ is the logarithm iterated k times). We also give a polynomial-time BPP simulation using Chor–Goldreich sources of min-entropy $R^{\Omega(1)}$, which is optimal. We present applications to time-space tradeoffs, expander constructions, and to the hardness of approximation. Of independent interest is our randomness-efficient Leftover Hash Lemma, a key tool for extracting randomness from weak random sources.

Key words. derandomization, expander graphs, hashing lemmas, hardness of approximation, imperfect sources of randomness, measures of information, pseudorandomness, pseudorandom generators, randomized computation, time-space tradeoffs

AMS subject classifications. 60C05, 68Q15, 94A17

PII. S009753979630091X

1. Introduction.

Randomness plays a vital role in almost all areas of computer science, including simulations, algorithms, network constructions, cryptography, and distributed algorithms. In practice, programs get their "random" bits by using pseudorandom number generators. Yet even in practice there are reports of algorithms giving quite different results under different pseudorandom generators; see, e.g., [FLW] for such reports on Monte-Carlo simulations, and [Hsu, HRD] for the deviant performance of some *RNC* algorithms for graph problems.

Other approaches involve using a physical source of randomness, such as a Zener diode, or using the last digits of a real-time clock. Not only is it unclear whether such bits will be random, but it is impossible to test them for "randomness." We can run certain statistical tests on the bits, but we cannot run all possible ones. It is

*Received by the editors March 25, 1996; accepted for publication (in revised form) December 11, 1997; published electronically April 20, 1999. Part of this work was done while the authors attended the Workshop on Probability and Algorithms organized by Joel Spencer and Michael Steele, which was held at the Institute for Mathematics and Its Applications at the University of Minnesota, Minneapolis, MN, September 20–24, 1993. A preliminary version of this work appears in *Proc. IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 264–275.

http://www.siam.org/journals/sicomp/28-4/30091.html

†Dept. of Information Systems and Computer Science, National University of Singapore, Singapore 119260, Republic of Singapore (aravind@iscs.nus.edu.sg). The research of this author was done in parts at the Institute for Advanced Study, Princeton, NJ (partially supported by grant 93-6-6 of the Alfred P. Sloan Foundation), at the DIMACS Center, Rutgers University, Piscataway, NJ (partially supported by NSF grant STC91-19999 and by the New Jersey Commission on Science and Technology), and at the National University of Singapore (partially supported by National University of Singapore Academic Research Fund grant RP960620).

‡Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712 (diz@cs.utexas.edu). The research of this author was partially supported by NSF NYI grant CCR-9457799. Part of this work was done while this author was visiting The Hebrew University of Jerusalem and was supported by a Lady Davis Postdoctoral Fellowship.

therefore natural and important to ask whether randomness is just as helpful even if the source of randomness is defective or weak. Thus we model a weak random source as outputting bits that are slightly random but not perfectly random.

There have been two different directions taken in the study of weak random sources. The first is an attempt to describe a weak random source arising in practice. Thus Blum [Blu] looked at the model where bits are output by a Markov chain and showed how to extract perfectly random bits from such a source. Santha and Vazirani [SV] then looked at a model where the only fact known about the source is that each bit has some randomness. More precisely, we have the following.

DEFINITION 1.1 (see [SV]). *A semirandom source with parameter α outputs bits $X_1X_2,\ldots,X_R$, such that for all $i \le R$, and for all $x_1,\ldots,x_i \in \{0,1\}$,*

$$\alpha \le Pr[X_i = x_i | X_1 = x_1,\ldots,X_{i-1} = x_{i-1}] \le 1-\alpha.$$

Simulations must work for all such sources; we do not want to assume any knowledge about the source, except that it is semirandom with the value of the parameter α being known. Santha and Vazirani proved that it is impossible to extract even a single almost-random bit from one such source (so Vazirani [Va1, Va3] used two independent sources). In light of this result, one might give up hope of simulating randomized algorithms with one semirandom source. Nevertheless, [VV] and [Va2] showed how to efficiently simulate all algorithms in RP and BPP, with one semirandom source, for any constant $\alpha > 0$.

Note that these simulations use $R = poly(n)$ bits from the semirandom source, where n denotes the length of the input to the RP or BPP machine; we shall let n and R denote "length of the input" and "number of bits requested," respectively, when discussing RP or BPP simulations in this paper.

Chor and Goldreich [CG] generalized the Santha–Vazirani model by assuming that no sequence of l bits has too high a probability of being output. More precisely, we have the following.

DEFINITION 1.2 (see [CG]). *A blockwise δ-source outputs bits as blocks $Y_1,\ldots,Y_s$, where Y_i has length l_i, such that for all $i, y_1,\ldots,y_i$, $Pr[Y_i = y_i | Y_1 = y_1,\ldots,Y_{i-1} = y_{i-1}] \le 2^{-\delta l_i}$.*[1]

Note that δ and the block-lengths l_i are assumed to be known parameters of these sources. Later we will allow the l_i to vary, but for the first four sections we assume that all $l_i = l$. A semirandom source corresponds to $l = 1$. For $l = O(\log n)$ and constant $\delta > 0$, Chor and Goldreich showed how to efficiently simulate any BPP algorithm using one blockwise δ-source. They further showed how to obtain almost-random bits from four independent such sources at a constant rate.

The other direction researchers have taken is natural mathematically, although it does not appear to correspond as well to weak sources in practice. These models are called bit-fixing models: some of the bits are perfectly random, while others are controlled by an adversary. Cohen and Wigderson [CW] distinguish three models based on three different adversaries: oblivious bit-fixing sources [CG+], nonoblivious bit-fixing sources [BL, KKL], and adaptive bit-fixing sources [LLS]. Only for the first model could researchers do something better than they could for general weak random sources (see [CW]).

The two directions in weak random sources were united by the model of δ-sources [Zu1, Zu2], which generalizes all the previous models.

[1] We modify the notation in [CG] to better conform with ours.

DEFINITION 1.3. *For any number R of random bits requested, a δ-source outputs an R-bit string such that no string has probability more than $2^{-\delta R}$ of being output.*

As usual, we associate a source with its distribution D. Let $D(x)$ denote the probability assigned to x under distribution D.

DEFINITION 1.4. *The* min-entropy *of a distribution D is* $\min_x\{-\log_2 D(x) \mid D(x) > 0\}$.

Thus, a δ-source is equivalent to a source with min-entropy at least δR. We often go back and forth between these two terminologies.

In [Zu2], Zuckerman showed how to efficiently simulate any BPP algorithm using a δ-source, for any fixed $\delta > 0$. Because a δ-source is the most general model, this implies that BPP algorithms can be simulated using any model of a source where randomness is output at a constant rate. In light of this, what is left to do? The answer is to extend the result for subconstant δ. From an information-theoretic viewpoint, it is necessary for the R bits to have min-entropy only R^γ, for an arbitrary but fixed $\gamma > 0$ [CW]. (Of course, if BPP = P, then no random bits are necessary: this information-theoretic result holds for an abstract model of BPP, where random bits are really necessary. Such an abstract model, wherein the "witness set" W can be *any* sufficiently large set, is introduced in Definition 2.3.) Can we achieve this information-theoretic lower bound? Previously, the only weak source where this information-theoretic lower bound could be achieved was the oblivious bit-fixing source: Cohen and Wigderson showed how to simulate BPP even if the adversary fixes all but R^γ bits and leaves the other bits unbiased and independent, matching the above lower bound [CW].

In this paper, we come close to our goal: we give a time $n^{O(\log n)}$ simulation for any RP algorithm using a δ-source with min-entropy R^γ. For $\gamma > 1/2$, our simulations also work for BPP and approximation algorithms. Moreover, for $\gamma > 1 - 1/(k+1)$, our simulations take time $n^{O(\log^{(k)} n)}$, giving a polynomial-time simulation for $\gamma > 1 - 1/(2\log^* n)$. (Here $\log^{(k)}$ denotes the logarithm base 2, iterated k times; i.e., $\log^{(1)} x = \log_2 x$, $\log^{(2)} x = \log_2 \log_2 x$, etc.) Furthermore, we give a simple algorithm to simulate BPP and approximation algorithms using the Chor–Goldreich blockwise δ-source, as long as there are at least R^γ blocks and the min-entropy of the R bits is at least R^γ (i.e., $\delta R \geq R^\gamma$), for any fixed $\gamma > 0$. (The second condition may not be implied by the first condition if $\delta l < 1$.) Using $l = 1$, this gives a simulation of BPP and approximation algorithms for the Santha–Vazirani source with min-entropy R^γ. We also generalize Cohen and Wigderson's result on oblivious bit-fixing sources: it is not necessary for n^γ bits to be perfectly independent and uniform, but only "weakly independent" (see section 7). Our BPP simulations also work for approximation algorithms, such as the one for approximating the volume of a convex body [DFK].

Our BPP simulations are corollaries of something even stronger: extractor constructions. An extractor is an algorithm which extracts randomness from a weak source, using a small additional number t of truly random bits. We modify the definition given in [NZ] to account for general families of sources.

DEFINITION 1.5. *Let $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$, and $\epsilon > 0$ be a parameter. E is called an extractor with quasi-randomness ϵ for a family of sources $\mathcal{S}$ on $\{0,1\}^n$ if, for any $S \in \mathcal{S}$, the distribution of $E(x,y) \circ y$ induced by choosing x according to S and y independently and uniformly from $\{0,1\}^t$ is quasi-random (on $\{0,1\}^m \times \{0,1\}^t$) to within ϵ. In particular, when $\mathcal{S}$ is the class of δ-sources on $\{0,1\}^n$, E is called an*

TABLE 1

Min-entropy n^γ for	t truly random bits for extractor	(N, M, d, K)-disperser construction
$\gamma > 0$	NA	$(2^n, 2^{\Omega(n^{\gamma/2}/\log n)}, n^{O(\log n)}, 2^{n^\gamma})$
$\gamma > 1/2$	$O(\log^2 n)$	$(2^n, 2^{n^{2\gamma-1}/\log n}, n^{O(\log n)}, 2^{n^\gamma})$
$\gamma > 2/3$	$O((\log n)\log\log n)$	$(2^n, 2^{n^{3\gamma-2}/\log^3 n}, n^{O(\log\log n)}, 2^{n^\gamma})$
$\gamma > 1 - 1/k$	$O((\log n)\log^{(k)} n)$	$(2^n, 2^{n^{k\gamma-k+1}/\log^{2k-1} n}, n^{O(\log^{(k)} n)}, 2^{n^\gamma})$
$\gamma > 1 - 1/(2\log^* n)$	$O(\log n)$	$(2^n, 2^{\sqrt{n}}, n^{O(1)}, 2^{n^\gamma})$

$(n, m, t, \delta, \epsilon)$-*extractor.*[2]

As in [NZ], we observe that an extractor that adds t random bits yields a BPP simulation taking time $2^t poly(n)$.

In the case when $\mathcal{S}$ is the class of δ-sources, it is often convenient to view the extractors graph theoretically, as in [Sip, San, CW]. Namely, construct a bipartite graph on $\{0,1\}^n \times \{0,1\}^m$, where $x \in \{0,1\}^n$ is adjacent to $z \in \{0,1\}^m$ if and only if $z = E(x, y)$ for some y. Then any set in $\{0,1\}^n$ of size at least $2^{\delta n}$ expands almost uniformly into $\{0,1\}^m$. In particular, it yields efficient constructions of graphs, which are called dispersers in [CW].

DEFINITION 1.6. *An (N, M, d, K)-disperser is a bipartite graph with N nodes on the left side, each with degree at most d, and M nodes on the right side, such that every subset of K nodes on the left side is connected to at least $M/2$ nodes on the right. By an efficient construction of a disperser, we mean that given a node on the left side, its neighbor set can be found in poly$(\log N + \log M + d)$ time deterministically.*

LEMMA 1.7. *If there is an efficient $(n, m, t, \delta, \epsilon)$-extractor for $\epsilon \le 1/2$, then there is an efficiently constructible $(2^n, 2^m, 2^t, 2^{\delta n})$-disperser.*

Our RP simulation also yields a disperser, although it is not an extractor. In Table 1 we summarize our results for δ-sources. The simulations and running times for the five entries of the table are as follows: The first entry implies an RP simulation, while the other four are for BPP. The respective running times are $n^{O(\log n)}$, $n^{O(\log n)}$, $n^{O(\log\log n)}$, $n^{O(\log^{(k)} n)}$, and $poly(n)$.

Just as extractors and dispersers for constant δ have important applications [NZ, WZ], so, too, do our results for subconstant δ. The first application is to a relationship between the RP=P question and time-space tradeoffs. Sipser [Sip] showed that if certain expander graphs can be constructed efficiently, then for some $\epsilon > 0$ and any time bound $t(n)$, either RP = P or all unary languages in $DTIME(t(n))$ are accepted infinitely often in $SPACE(t(n)^{1-\epsilon})$. If we had a polynomial-time simulation of RP using a δ-source with min-entropy R^γ for some $\gamma < 1$, we could construct his expanders as a corollary. Because our simulations take time $n^{O(\log^{(k)} n)}$, we instead show unconditionally that either $RP \subseteq \bigcap_k DTIME(n^{\log^{(k)} n})$ or all unary languages in $DTIME(t(n))$ are accepted infinitely often in $\bigcap_k SPACE(t(n)^{1-1/\log^{(k)} n})$.

Our second application is to improve the expanders constructed in [WZ], and hence all of the applications given there. In [WZ], graphs on n nodes were constructed such that for every pair of disjoint subsets S_1 and S_2 of the vertices with $|S_1| \ge n^\delta$ and $|S_2| \ge n^\delta$, there is an edge joining S_1 and S_2; the graphs so constructed had

[2] To remember the five parameters, it may be helpful to note that the first three refer to lengths of inputs and outputs in (typically) decreasing order of length. The last two parameters refer to the quality of the sampler.

essentially optimal maximum degree $n^{1-\delta+o(1)}$. These expanders were used in [WZ] to explicitly construct

(i) a k-round sorting algorithm using $n^{1+1/k+o(1)}$ comparisons;

(ii) a k-round selection algorithm using $n^{1+1/(2^k-1)+o(1)}$ comparisons;

(iii) a depth-2 superconcentrator of size $n^{1+o(1)}$; and

(iv) a depth-k wide-sense nonblocking generalized connector of size $n^{1+1/k+o(1)}$.

The reader is referred to [WZ] for the definitions and motivations for these constructions. All of these results are optimal to within factors of $n^{o(1)}$. In [WZ], these $n^{o(1)}$ factors were $2^{(\log n)^{4/5+o(1)}}$, improved to $2^{(\log n)^{2/3+o(1)}}$ in the final version of [WZ]. Our results further improve these $n^{o(1)}$ factors to $2^{(\log n)^{1/2+o(1)}}$. In addition, explicit linear-sized n-superconcentrators of depth $(\log n)^{2/3+o(1)}$ were presented in [WZ]; we improve this to $(\log n)^{1/2+o(1)}$ depth. These might seem like small improvements, given the major improvement of simulations using δ-sources. The reasons for this are that our extractors require $n^{\Omega(1)}$ min-entropy from an n-bit source and that our extractors do not extract a sufficiently good fraction of the min-entropy.

Our third application is to the hardness of approximating $\log\log\omega(G)$, where $\omega(G)$ is the maximum size of a clique in an input graph G. Let $\tilde{P}$ denote quasi-polynomial time, $\cup_{c>0} DTIME(2^{(\log n)^c})$. In [Zu2], it was shown that if $N\tilde{P} \neq \tilde{P}$, then approximating $\log\omega(G)$ to within any constant factor is not in $\tilde{P}$. In [Zu3], a randomized reduction was given showing that any iterated log is hard to approximate under a slightly stronger assumption than $N\tilde{P} \neq ZP\tilde{P}$. In particular, if $N\tilde{P} \neq ZP\tilde{P}$, then approximating $\log\log\omega(G)$ to within a constant factor is not in $co - R\tilde{P}$. This used the fact that, with high probability, certain graphs are highly expanding. Our work allows us to deterministically construct graphs that are almost as highly expanding as the nonexplicit constructions, thus making this last reduction deterministic, with a slight loss of efficiency: if $N\tilde{P} \not\subseteq DTIME(2^{(\log n)^{O(\log\log n)}})$, then approximating $\log\log\omega(G)$ to within any constant factor is not in $\tilde{P}$.

As in [Zu1, Zu2, NZ], we achieve our results using only elementary methods; in particular, we do not need expander graphs. Our main technical tool is a modification of the Leftover Hash Lemma. This very useful lemma was first proved in [ILL] and has been used extensively in simulations using δ-sources [Zu1, Zu2, NZ]. This lemma is a pseudorandom property of hash functions. However, a drawback of the lemma is that to hash from s bits to t bits, one needs at least s random bits. We show how a similar lemma can be achieved using only $O(\log s + t)$ random bits. Because this modification was so useful to us here, we believe it will be useful elsewhere, too. A similar lemma was proved independently in [GW]; however, our proof is somewhat simpler. One key consequence of our lemma is an improvement of the extractor of [NZ]. That is, we show how to add a small number t of truly random bits to a δ-source in order to extract almost-random bits; we make t much smaller than in [NZ]. By using this with the ideas of [NZ], we get our first main extractor (see Theorem 5.7). Section 5.5 then introduces some new techniques for using such extraction procedures recursively; this helps improve the quality of our extractor when δ is not "too small."

Section 2 sets up the required preliminary notions. Section 3 presents our first technical tool, the improved Leftover Hash Lemma; section 4 shows how this new Leftover Hash Lemma can be used to run BPP algorithms using very weak Chor–Goldreich sources, and also serves in part as motivation for some of our techniques of sections 5 and 6. Sections 5 and 6 contain some of our main results: simulating BPP and RP algorithms using general weak sources with very low min-entropy. Section 7 uses the result of section 4 to generalize a result of [CW] on oblivious bit-fixing

sources. Applications of our results are presented in section 8. Section 9 concludes with some recent work that our work has led to, in part, and presents open questions. In the Appendix, we show some technical details that are largely borrowed from [NZ].

2. Preliminaries. We use capital letters to denote random variables, sets, distributions, and probability spaces; lowercase letters denote other variables. We often use a correspondence where the lowercase letter denotes an instantiation of the capital letter, e.g., $\vec{x}$ might be a particular input and $\vec{X}$ the random variable being uniformly distributed over all inputs. We ignore round-off errors, assuming when needed that a number is an integer; it can be seen that this does not affect the validity of our arguments. All logarithms are to base 2 unless specified otherwise.

2.1. Basic definitions.

DEFINITION 2.1. *RP is the set of languages $L \subseteq \{0,1\}^*$ such that there is a deterministic polynomial-time Turing machine $M_L(\cdot,\cdot)$ for which*

$$a \in L \Rightarrow Pr[M_L(a,x) \text{ accepts}] \geq 1/2,$$

and

$$a \notin L \Rightarrow Pr[M_L(a,x) \text{ accepts}] = 0,$$

where the probabilities are for an x chosen uniformly in $\{0,1\}^{p(|a|)}$ for some polynomial $p = p_L$. BPP is the set of languages $L \subseteq \{0,1\}^*$ such that there is a deterministic polynomial-time Turing machine $M_L(\cdot,\cdot)$ for which

$$a \in L \Rightarrow Pr[M_L(a,x) \text{ accepts}] \geq 2/3$$

and

$$a \notin L \Rightarrow Pr[M_L(a,x) \text{ accepts}] \leq 1/3,$$

where the probabilities are for an x chosen uniformly in $\{0,1\}^{p(|a|)}$ for some polynomial $p = p_L$.

As is well known, by running M_L on independent random tapes we can change the probabilities $1/2$, $2/3$, and $1/3$ above to $1 - 2^{-poly(|a|)}$, $1 - 2^{-poly(|a|)}$, and $2^{-poly(|a|)}$, respectively, while still retaining a polynomial running time.

Distance between distributions. Let D_1 and D_2 be two distributions on the same space X. The variation distance between them is

$$\|D_1 - D_2\| \doteq \max_{Y \subseteq X} |D_1(Y) - D_2(Y)| = \frac{1}{2} \sum_{x \in X} |D_1(x) - D_2(x)|.$$

A distribution D on X is called ϵ-quasi-random (on X) if the variation distance between D and the uniform distribution on X is at most ϵ.

A convenient fact to remember is that distance between distributions cannot be created out of nowhere. In particular, if $f : X \to Y$ is any function and D_1, D_2 are distributions on X, then $\|f(D_1) - f(D_2)\| \leq \|D_1 - D_2\|$. Also, if E_1 and E_2 are distributions on Y, then $\|D_1 \times E_1 - D_2 \times E_2\| \leq \|D_1 - D_2\| + \|E_1 - E_2\|$. Since this inequality holds for any function, it also holds for random functions f. Next, the triangle inequality is obvious: $\|D_1 - D_3\| \leq \|D_1 - D_2\| + \|D_2 - D_3\|$.

2.2. Simulations using weak random sources. A source that outputs R bits is a probability distribution on $\{0,1\}^R$; we often go back and forth between these two notions. By a simulation using, say, a Chor–Goldreich source, we really mean a simulation that will work for *all* Chor–Goldreich sources. Thus, we talk about a simulation for a family of sources.

To define what simulating RP means, say we wish to test whether a given string a is in an RP language L. If $a \notin L$, then all random strings cause M_L to reject, so there is nothing to do. Suppose $a \in L$; then we wish to find with high probability a witness to this fact. Let W be the set of witnesses, i.e., $W = \{x \in \{0,1\}^r | M_L(a,x) \text{ accepts}\}$, where r denotes the number of random bits used by M_L. One might think that to simulate RP using a source we would need a different algorithm for each language in RP. Instead, we exhibit one simulation that works for *all* W with $|W| \geq 2^{r-1}$; in particular, we do not use the fact that W can be recognized in P.

We note that $s \geq r - O(\log r)$ random bits are required for this "abstract-RP" problem. For if s random bits are used and the algorithm can output $r^{O(1)}$ r-bit strings, then the number of possible outputs $2^s r^{O(1)}$ must exceed 2^{r-1}. As we can ask for at most $r^{O(1)}$ bits from the source, this is what gives the R^γ min-entropy lower bound of [CW].

DEFINITION 2.2. *A polynomial-time algorithm simulates RP using a source from a family of sources $\mathcal{S}$ if, on input any constant $c > 0$ and $R = poly(r)$ bits from any $S \in \mathcal{S}$, it outputs a polynomial number of r-bit strings z_i, such that for all $W \subseteq \{0,1\}^r, |W| \geq 2^{r-1}$, $Pr[(\exists i) z_i \in W] \geq 1 - r^{-c}$.*

If we had a perfect random source, we could make the error exponentially small. Indeed, with sources like the Chor–Goldreich blockwise δ-source, where we can request more bits with independence conditions from the first bits, we can always repeat the algorithm to achieve an exponentially small error. However, with arbitrary sources, it is not obvious that this can be done. Yet it seems reasonable to insist only on a polynomially small error, as we want the error to fool polynomial-time machines.

For BPP, we have no "witnesses" to membership, but by an abuse of notation we use W to denote the set of random strings producing the right answer. As before, a simulation of BPP will produce strings z_i and use these to query whether $a \in L$. The simulation does not have to take the majority of these answers as its answer, but since we do so it makes it simpler to define it that way.

DEFINITION 2.3. *A polynomial-time algorithm simulates BPP using a source from a family of sources $\mathcal{S}$ if, on input any constant $c > 0$ and $R = poly(r)$ bits from any $S \in \mathcal{S}$, it outputs a polynomial number of r-bit strings z_i, such that for all $W \subseteq \{0,1\}^r, |W| \geq \frac{2}{3}2^r$, $Pr[\text{majority of } z_i\text{'s lie in } W] \geq 1 - r^{-c}$.*

Such an algorithm A can also be used to simulate approximation algorithms since, if a majority of numbers lie in a given range, then their median also lies in it. Thus by taking medians instead of majorities, a good approximation can be obtained with probability at least $1 - r^{-c}$.

Our BPP simulations are actually extractor constructions. As in [NZ], an extractor construction yields a BPP simulation; the idea is to run the extractor using all possible 2^t strings y, and then produce an appropriate output.

LEMMA 2.4. *If there is a polynomial-time extractor for $\mathcal{S}$ with parameters $\epsilon = n^{-\Omega(1)}$, $m = n^{\Omega(1)}$, and t, then there is a simulation of BPP using any source S from $\mathcal{S}$ and running in time $2^t n^{O(1)}$.*

Remark. In fact, as observed in [Zu2], for δ-sources we need only $\epsilon \leq 1/3$, say, to achieve even an exponentially small error for $\delta' > \delta$.

In order to make our statements cleaner and boost the size of the output from $\Omega(m)$ to m for reasonable m, we use the following lemma, which is a corollary of a lemma in [WZ].

LEMMA 2.5 (see [WZ]). *Suppose $m \leq \delta n/4$ and, for some integer k, $\epsilon \geq 2^{-\delta n/(5k)}$. If there is an efficient $(n, m/k, t, \delta/2, \epsilon/(2k))$-extractor, then there is an efficient $(n, m, kt, \delta, \epsilon)$-extractor.*

Finally, although we focus on extractors that run in polynomial time, all our extractors can actually be made to run in NC; see the remark at the end of section 3.

3. The Leftover Hash Lemma using fewer random bits. To understand the Leftover Hash Lemma intuitively, imagine that we have an element x chosen uniformly at random from an arbitrary set $A \subseteq \{0,1\}^s$ with $|A| = 2^t$, $t < s$. Thus we have t bits of randomness, but not in a usable form. If we use some additional random bits, the Leftover Hash Lemma allows us to convert the randomness in x into a more usable form. These extra bits are used to pick a uniformly random hash function h mapping s bits to $t - 2k$ bits, where k is a security parameter. Recall that given finite sets A and B, a family H of functions mapping A to B is a *universal family of hash functions* if for any $a_1 \neq a_2 \in A$, and any $b_1, b_2 \in B$, $Pr[h(a_1) = b_1 \text{ and } h(a_2) = b_2] = 1/|B|^2$, where the probability is over h chosen uniformly at random from H. The Leftover Hash Lemma guarantees that the ordered pair $(h, h(x))$ is almost random.

LEFTOVER HASH LEMMA (see [ILL]). *Let $A \subseteq \{0,1\}^s, |A| \geq 2^t$. Let $k > 0$, and let H be a universal family of functions mapping $\{0,1\}^s$ to $\{0,1\}^{t-2k}$. Then the distribution of $(h, h(x))$ is quasi-random within 2^{-k} (on the set $H \times \{0,1\}^{t-2k}$) if (h, x) is chosen uniformly at random from $H \times A$.*

One drawback of this lemma is that to pick a universal hash function mapping s bits to $t - 2k$ bits, one needs at least s bits. One way around this is to use the extractor of [NZ]; however, that is only useful if t/s is large. Here we show how to use only $O(t + \log s)$ bits and achieve a good result. We first recall the following.

DEFINITION 3.1 (see [NN]). *A "d-wise ρ-biased" sample space S of n-bit vectors has the property that if $\vec{X} = (X_1, \ldots, X_n)$ is sampled uniformly at random from S, then for all $I \subseteq \{1, 2, \ldots, n\}, |I| \leq d$, for all $b_1, b_2, \ldots, b_{|I|} \in \{0,1\}$,*

$$\left| Pr_{\vec{X} \in S} \left[\bigwedge_{i \in I} X_i = b_i \right] - 2^{-|I|} \right| \leq \rho. \tag{1}$$

Simplifying the construction in [NN], d-wise ρ-biased spaces of cardinality $O((d \log n/\rho)^2)$ were constructed explicitly in [AG+]. In addition, given the random bits to sample from S, any bit of $\vec{X}$ can be computed in $poly(d, \log n, \log(\rho^{-1}))$ time.

LEMMA 3.2. *Let $A \subseteq \{0,1\}^s, |A| \geq 2^t$, $k > 0$, and $\epsilon \geq 2^{1-k}$. There is an explicit construction of a family F of functions mapping s bits to $t - 2k$ bits, such that the distribution of $(f, f(x))$ is quasi-random within ϵ (on the set $F \times \{0,1\}^{t-2k}$), where f is chosen uniformly at random from F, and x uniformly from A. A random element from F can be specified using $4(t-k)+O(\log s)$ random bits; given such a specification of any $g \in F$ using $4(t-k) + O(\log s)$ bits and given any $y \in \{0,1\}^s$, $g(y)$ can be computed in time $poly(s, t-k)$.*

Proof. Any $g : \{0,1\}^s \to \{0,1\}^{t-2k}$ can be represented in the natural way by a vector in $\{0,1\}^\ell$, where $\ell = (t-2k)2^s$. Now let F be a $2(t-2k)$-wise ρ-biased sample space for ℓ-length bit vectors, where $\rho = (\epsilon^2 2^{2k} - 1)2^{-2t+2k}$; ρ is nonnegative since $\epsilon \geq 2^{1-k}$. (That is, F is a family of functions, where each element of F is a function that maps $\{0,1\}^s$ to $\{0,1\}^{t-2k}$.) F can be sampled using $2(\log(t-2k) + \log\log \ell +$

$\log \rho^{-1}) + O(1) \leq 4(t-k) + O(\log s)$ random bits. The lemma's claim about $g(y)$ being efficiently computable follows from the above mentioned fact that individual bits of any string in the support of a small-bias space can be computed efficiently.

We now show that the distribution of $(f, f(x))$ is quasi-random. We follow the proof of the Leftover Hash Lemma due to Rackoff (see [IZ]).

DEFINITION 3.3. *The* collision probability $cp(D)$ *of a distribution* D *on a set* S *is* $Pr[y_1 = y_2]$, *where* y_1 *and* y_2 *are chosen* independently *from* S *according to* D.

For the distribution D of $(f, f(x))$,

$$cp(D) = Pr_{x_1,x_2 \in A, f_1,f_2 \in F}[f_1 = f_2, f_1(x_1) = f_2(x_2)],$$

where all the random choices are uniform and independent. We show that the collision probability using F is almost the same as it would be using a universal family of hash functions. Now,

$$\begin{aligned} cp(D) &= \frac{1}{|F|} Pr_{x_1,x_2 \in A, f \in F}[f(x_1) = f(x_2)] \\ &\leq \frac{1}{|F|}(Pr_{x_1,x_2 \in A}[x_1 = x_2] + Pr_{x_1,x_2 \in A, f \in F}[f(x_1) = f(x_2) | x_1 \neq x_2]) \\ &= 2^{-t}/|F| + \frac{1}{|F|} Pr_{x_1,x_2 \in A, f \in F}[f(x_1) = f(x_2) | x_1 \neq x_2] \\ &\leq 2^{-t}/|F| + \frac{1}{|F|} \max_{a_1 \neq a_2} Pr_{f \in F}[f(a_1) = f(a_2)]. \qquad (2) \end{aligned}$$

For any $a_1, a_2 \in A$, $a_1 \neq a_2$,

$$\begin{aligned} \frac{1}{|F|} Pr_{f \in F}[f(a_1) = f(a_2)] &= \frac{1}{|F|} \textstyle\sum_{b \in \{0,1\}^{t-2k}} Pr_{f \in F}[f(a_1) = f(a_2) = b] \\ &\leq \frac{1}{|F|} \textstyle\sum_{b \in \{0,1\}^{t-2k}} (2^{-2(t-2k)} + \rho) \text{ (by (1))} \\ &= \frac{1}{|F| 2^{t-2k}} (1 + \epsilon^2 - 2^{-2k}). \end{aligned}$$

Thus, from (2), $cp(D) \leq (1+\epsilon^2)/(|F|2^{t-2k})$. Now, the rest of Rackoff's proof shows that if U is the uniform distribution on $F \times \{0,1\}^{t-2k}$, then $cp(D) \leq (1+\epsilon^2) cp(U) = (1+\epsilon^2)/(|F|2^{t-2k})$ implies the ϵ-quasi-randomness of D. This concludes the proof. □

COROLLARY 3.4. *The conclusion to Lemma* 3.2 *holds if* x *is chosen from any* δ*-source,* $\delta = t/s$.

Proof. The only place where the distribution of x is needed in the above proof is in showing that its collision probability is 2^{-t}; note that 2^{-t} is an upper bound on the collision probability if x is chosen from a δ-source with $\delta = t/s$. This concludes the proof. □

Remark. In order to use Lemma 3.2, we need an irreducible polynomial over $GF[2]$ for the d-wise ρ-biased spaces. For this we use an algebraic result stating that if an integer m is of the form $2 \cdot 3^t$, then $f(z) = z^m + z^{m/2} + 1$ is an explicit irreducible polynomial over $GF[2]$ of degree m (see exercise 3.96, page 146 of [LN]). This allows our extractors to run in NC.

4. Simulating BPP using a blockwise δ-source with min-entropy R^γ. We now show how to simulate BPP using a Chor–Goldreich source with min-entropy R^γ for any fixed $\gamma > 0$; in fact, we build an extractor for Chor–Goldreich sources. Note that δ-sources are much weaker than these sources: in these sources, we know that each block has "a lot of randomness," even conditional on the previous blocks' values. In a δ-source, an adversary can, for instance, locate a lot of the randomness in certain positions that are unknown to us. Nevertheless, there are two reasons for the material of this section: to motivate our main results and the reasons for their difficulty and to construct an extractor that works with sources of min-entropy R^γ for *any* fixed $\gamma > 0$. Furthermore, this section will be useful in generalizing a result of [CW] on oblivious bit-fixing sources; see also section 7.

Note that since we are talking about extractors, we use the more usual symbol n (rather than R) in this section to denote the number of random bits requested from the source.

Suppose we are given a blockwise δ-source with $m\ (= n/l) = n^{\Omega(1)}$ blocks and with the min-entropy δn of the n bits being at least $n^{\Omega(1)}$. First note that we may always increase the block length by grouping successive blocks together. Suppose we want the output of E to be quasi-random to within n^{-c}. By choosing the block-length l appropriately, we may assume that the min-entropy of a block, $b = \delta l$, satisfies $b \geq 4(c+2)\log n$. We may also assume that $b = \Theta(\log n)$, since a blockwise δ-source is trivially a blockwise δ-source, if $\delta' < \delta$. We then choose a family F that satisfies Lemma 3.2 with parameters $k = (c+2)\log n$ and $\epsilon = n^{-(c+1)}$. Now we use the following modification of a lemma from the final version of [Zu2] which, using the Leftover Hash Lemma in the manner of [IZ], essentially strengthened related lemmas in [Va2] and [CG].

LEMMA 4.1. *Let F be a function family mapping l bits to $b - 2k$ bits, satisfying Lemma 3.2 with parameters $k = (c+2)\log n$ and $\epsilon = n^{-(c+1)}$. Let D be a blockwise δ-source on $\{0,1\}^{ml}$. If $\vec{Y} = Y_1, \ldots, Y_m$ is chosen according to D, and f is chosen uniformly at random from F, then the distribution of $(f, f(Y_1), \ldots, f(Y_m))$ is quasi-random to within $m\epsilon$.*

Proof. The proof is by backward induction, as in [NZ]. We proceed by induction from $i = m$ to $i = 0$ on the statement that, for any sequence of values $y_1, \ldots, y_i$, the distribution of $(f, f(Y_{i+1}), \ldots, f(Y_m))$ conditioned on $Y_1 = y_1, \ldots, Y_i = y_i$ is quasi-random to within $(m-i)\epsilon$. This is obvious for $i = m$. Suppose it is true for $i+1$. Fix the conditioning $Y_1 = y_1, \ldots, Y_i = y_i$ from now on, and let D_{i+1} denote the induced distribution on Y_{i+1}. We now use the obvious fact that if a statement is true for each element of a set, then it is also true for an element chosen randomly from the set, using any probability distribution. Since, by the induction hypothesis, for every y_{i+1}, the induced distribution on $(f, f(Y_{i+2}), \ldots, f(Y_m))$ is quasi-random to within $(m-i-1)\epsilon$, we have that the distribution $(Y_{i+1}, f, f(Y_{i+2}), \ldots, f(Y_m))$ is within $(m-i-1)\epsilon$ of the distribution $D_{i+1} \times U_{i+1}$, where U_{i+1} is the uniform distribution on $F \times \{0,1\}^{(m-i-1)(b-2k)}$. Thus, the distribution of $(f, f(Y_{i+1}), \ldots, f(Y_m))$ is within $(m-i-1)\epsilon$ of the distribution of $(f, f(Y_{i+1}), z_{i+2}, \ldots, z_m)$ obtained by choosing Y_{i+1} according to D_{i+1}, and $(f, z_{i+2}, \ldots, z_m)$ independently and uniformly at random from $F \times \{0,1\}^{(m-i-1)(b-2k)}$ (since $\|g(D_1) - g(D_2)\| \leq \|D_1 - D_2\|$ for any two distributions D_1 and D_2 and any function g). Using Corollary 3.4, the distribution of $(f, f(Y_{i+1}), z_{i+2}, \ldots, z_m)$ is quasi-random to within ϵ, and the lemma follows from the triangle inequality for variation distance. ∎

Sampling from F requires $O(\log l + b + \log(1/\epsilon)) = O(\log n)$ random bits, and thus

we have a good extractor for these sources. Using Lemma 2.4, we get the following.

THEOREM 4.2. *For any fixed $\gamma > 0$, BPP can be simulated using a blockwise δ-source as long as there are at least n^γ blocks and if the min-entropy of the n bits is at least n^γ (i.e., $\delta n \geq n^\gamma$). In fact, an explicit extractor $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^s$ that runs in NC can be built for this family of sources, with $t = O(\log n)$, $s = n^{\Omega(1)}$, and with the quasi-randomness of the output being $n^{-\Omega(1)}$.*

5. Simulating BPP using a δ-source with min-entropy $R^{1/2+\epsilon}$. We first present some intuition and preliminary ideas behind our extractor construction.

5.1. Intuition and preliminaries. To construct an extractor for a given δ-source D outputting n-bit strings, we follow the same high-level approach as does [NZ]; the crucial differences arise from the fact that the work of [NZ] focused on constant δ, while we need to work with a δ that goes to zero (fairly quickly) with n. We then introduce additional new ideas to bootstrap this construction in subsection 5.5.

The high-level idea is to first convert the output of D into a blockwise δ'-source with suitable block-lengths $l_1, l_2, \ldots, l_s$; this construction is called a *blockwise converter* and is described in subsection 5.3. ($\delta' = \delta^{1-o(1)}$.) This construction is a slight modification of that in [NZ] and hence, many of the details are shown in the Appendix (one important difference is that the l_i values are chosen differently). We output $O(\log n)$ blocks, expending $O(\log n)$ truly random bits for each block; hence, we use $O(\log^2 n)$ random bits here in total. Once this is done, we need to extract quasi-random bits from such a blockwise δ'-source, and such a construction, called a *blockwise extractor*, is presented in subsection 5.2. Here is where we need to replace the application of the Leftover Hash Lemma by our improved version; only $O(\log n)$ truly random bits are needed for this task.

The reason why the above approach works only for min-entropy $n^{1/2+\Omega(1)}$ is as follows. In constructing the blockwise converter, our approach can ensure an output that is (close to) a blockwise δ'-source only if, in particular, $\sum_i l_i = O(\delta n)$. Given $O(\delta n)$ bits from a δ-source, one can intuitively expect these to have at most $O(\delta^2 n)$ bits of randomness; thus, since we wish to extract $n^{\Omega(1)}$ (quasi-)random bits, we need $\delta^2 n = n^{\Omega(1)}$, i.e., $\delta = n^{-1/2+\Omega(1)}$, which is equivalent to min-entropy $n^{1/2+\Omega(1)}$.

The above outline suggests an extractor for min-entropy $n^{1/2+\Omega(1)}$, using $O(\log^2 n)$ bits. How can this be improved (to $O(\log n)$ ideally)? We present a partial solution as a bootstrapping approach in subsection 5.5, which has a lesser randomness requirement for relatively "large" δ, e.g., for $\delta = n^{-1/3+\Omega(1)}$. An interesting open question is whether we can efficiently construct an appropriate blockwise converter that outputs a total of $O(n)$ bits using only $O(\log n)$ truly random bits; this will suffice to give a near-optimal extractor.

We now proceed formally. We use our new Leftover Hash Lemma to modify the extractor developed in [NZ]. Part of the extractor there used the original Leftover Hash Lemma to show how to extract quasi-random bits from a certain kind of blockwise δ-source B. However, since the original Leftover Hash Lemma needs a number of random bits proportional to the logarithm of the size of the domain of the hash functions, the block-sizes in B had to decrease at the rate of $(1 + \delta/4)$, thus requiring $O((\log n)/\delta)$ blocks overall. However, our new construction allows the block-sizes of B to decrease at a constant rate independent of δ, thus requiring only $O(\log n)$ blocks. This is because now, if we need to hash from a δ-source on l bits to $\delta l/2$ bits, Lemma 3.2 and Corollary 3.4 guarantee a hash function family having error $2^{1-\delta l/4}$ which can be described using $3\delta l + O(\log l)$ bits (this is at most $4\delta l$ bits, provided

$1/\delta \leq cl/\log l$ for a sufficiently small constant c). To see this, just plug in $s = l$, $t = \delta l$, $k = \delta l/4$, and $\epsilon = 2^{1-k}$ in Corollary 3.4.

As in [NZ], our extractor first converts a δ-source into a blockwise δ-source with blocks of varying lengths and then extracts good bits from the blockwise δ-source. Unlike [NZ], we define these two intermediate constructions explicitly here.

DEFINITION 5.1. (i) *$E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^{l_1+\cdots+l_s}$ is an $(n, (l_1, \ldots, l_s), t, \delta, \delta', \epsilon)$ blockwise converter if, for x chosen from a δ-source on $\{0,1\}^n$ and y independently and uniformly at random from $\{0,1\}^t$, $E(x,y) \circ y$ is within ϵ of a distribution $D \times U$, where D is some blockwise δ'-source with block-lengths $l_1, \ldots, l_s$ and U is the uniform distribution on $\{0,1\}^t$.* (ii) *$E : \{0,1\}^{l_1+\cdots+l_s} \times \{0,1\}^t \to \{0,1\}^m$ is an $((l_1, \ldots, l_s), m, t, \delta, \epsilon)$ blockwise extractor if, for x chosen from a blockwise δ-source with block-lengths $l_1, \ldots, l_s$ and y independently and uniformly at random from $\{0,1\}^t$, $E(x,y) \circ y$ is ϵ-quasi-random on $\{0,1\}^m$.*

Lemma 5.2, implicit in [NZ], shows how to combine a blockwise converter and a blockwise extractor.

LEMMA 5.2. *Suppose we are given an efficient $(n, (l_1, \ldots, l_s), t_1, \delta, \delta', \epsilon_1)$ blockwise converter and an efficient $((l_1, \ldots, l_s), m, t_2, \delta', \epsilon_2)$ blockwise extractor. Then we can construct an efficient $(n, m, t_1 + t_2, \delta, \epsilon_1 + \epsilon_2)$-extractor.*

Proof. For the proof, just run the converter on the output of the δ-source and then run the extractor on the output of the converter. □

Given Lemma 5.2, we now focus on constructing an appropriate blockwise converter and a blockwise extractor; these are described in subsections 5.2 and 5.3, respectively.

5.2. A blockwise extractor. Lemma 4.1 gives a blockwise extractor. However, our blockwise converter will be useful only if the number of blocks s in the blockwise δ-source is small; Lemma 4.1 results in $s = n^{\Theta(1)}$, which is too high for our purposes. We therefore use the following blockwise extractor C, which is similar to the one in [NZ] except that we use our improved version of the Leftover Hash Lemma.

Function C. The function C has four parameters: r, the number of bits used to describe a member of the hash family; s, the number of blocks; l_s, the smallest block size; and δ, the quality of the source. (To avoid details that may be distracting at this point, we discuss the reasons for the bounds on some of these parameters at the end of this subsection.) C works only if r bits suffice to hash from l_s bits to $\delta l_s/2$ bits to get a distribution that is quasi-random to within $2^{1-r/16}$, as prescribed by Corollary 3.4. Thus, as explained in the last paragraph of this subsection,

$$c \log l_s \leq r \leq 3\delta l_s + O(\log l_s) \leq 4\delta l_s \tag{3}$$

for a suitably large constant c.

We define $r_s = r$ and $r_{i-1}/r_i = 9/8$, and then the block lengths $l_i = \max(r_i/(4\delta), l_s)$. Then Lemma 3.2 ensures for each i a fixed family of hash functions $H_i = \{h : \{0,1\}^{l_i} \to \{0,1\}^{\delta l_i/2}\}$ with $|H_i| \leq 2^{r_i}$, so we assume $|H_i| = 2^{r_i}$.

1. INPUT: $x_1 \in \{0,1\}^{l_1}, \ldots, x_s \in \{0,1\}^{l_s}$; $y \in \{0,1\}^r$.
2. $h_s \leftarrow y$.
3. For $i = s$ down to 1 do $h_{i-1} \leftarrow h_i \circ h_i(x_i)$.
4. OUTPUT (a vector in $\{0,1\}^{r_0 - r}$): h_0, excluding the bits of h_s.

By choosing s large enough, we can ensure that $l_0 = r_0/(4\delta)$. Specifically, suppose $s \geq \log_{9/8}(4\delta l_s/r)$. Then, $r_0/(4\delta) = r(9/8)^s/(4\delta) \geq l_s$ and hence, by the definition of l_i, $r_0 = 4\delta l_0$.

LEMMA 5.3. *C is an $((l_1, \ldots, l_s), r_0 - r, r, \delta, 4 \cdot 2^{-r/16})$ blockwise extractor.*

Proof. This proof is very similar to a corresponding proof in [NZ]. Let $\vec{X} = X_1, \ldots, X_l$ be chosen according to D, a blockwise δ-source on $\{0,1\}^{l_1+\cdots+l_s}$, and Y be chosen uniformly from $\{0,1\}^r$. Let $r_s = r$ and $r_{i-1}/r_i = 9/8$. Then $r_i \le 4\delta l_i$. We will prove by induction from $i = s$ down to $i = 0$ the following claim, which clearly implies the lemma.

Claim. For any sequence of values $x_1, \ldots, x_i$, the distribution of h_i conditioned on $X_1 = x_1, \ldots, X_i = x_i$ is quasi-random to within ϵ_i, where $\epsilon_i = \sum_{j=i+1}^{s} 2^{1-r_j/16}$.

This claim is clearly true for $i = s$. Now suppose it is true for $i+1$. Fix the conditioning $X_1 = x_1, \ldots, X_i = x_i$, and let D_{i+1} denote the induced distribution on X_{i+1}. Since, by the induction hypothesis, for every x_{i+1} the induced distribution on h_{i+1} is quasi-random, we have that the distribution (X_{i+1}, h_{i+1}) is within ϵ_{i+1} of the distribution $D_{i+1} \times U_{i+1}$, where U_{i+1} is the uniform distribution on H_{i+1}. Thus, the distribution of h_i is within ϵ_{i+1} of the distribution obtained by choosing x_{i+1} according to D_{i+1}, and h_{i+1} independently and uniformly at random in H_{i+1}. Using Corollary 3.4 this second distribution is quasi-random to within $2^{1-r_{i+1}/16}$. □

We now explain (3). If the upper bound on r does not hold, then the error cannot be made small enough: the error will be $O(2^{-\delta l_s/4})$ rather than $O(2^{-r/16})$. If the lower bound does not hold, then the domain is too large for our hash family to work. Next, to understand the equations $l_i = \max(r_i/(4\delta), l_s)$, the reader should first think of $r = 4\delta l_s$ and $l_{i-1}/l_i = 9/8$. The reason we choose l_s larger is to reduce the error ϵ of the blockwise converter in Lemma 5.5. Note that Lemma A.1 allows smaller errors for larger values of l.

5.3. A blockwise converter. Our blockwise converter is a small modification of that in [NZ]. For our simulations of RP and BPP, it would suffice to change the k-wise independence in [NZ] to pairwise independence, as was done in [Zu2]. However, by using an improved analysis of k-wise independence from [BR], we can give a good extractor for a wider range of parameters. The results of this subsection are essentially taken from [NZ], with the only changes being this improved analysis of [BR]. Thus, in order to not obscure the main new ideas, we just present a sketch of the blockwise converter and leave the necessary details to the Appendix.

In order to define our blockwise converter, we first show how to extract one block from a δ-source. The way to do this is as follows. Intuitively, a δ-source has many bits which are somewhat random. We wish to obtain l of these somewhat random bits. This is not straightforward, as we do not know which of the n bits are somewhat random. We therefore pick the l bits at random using k-wise independence.

Choosing l out of n elements. We divide the n elements into disjoint sets $A_1, \ldots, A_l$ of size $m = n/l$, i.e., $A_i = \{(i-1)m+1, (i-1)m+2, \ldots, im\}$. We then use $k \log n$ random bits to choose $X_1, \ldots, X_l$ k-wise independently, where the range of X_i is A_i. (In other words, each X_i is uniformly distributed in A_i, and any k of the X_i's are mutually independent.) Our (random) output is $S = \{X_1, \ldots, X_l\}$. Methods to construct such k-wise independent random variables using $k \log n$ random bits are well known; see, e.g., [ABI, Lub].

Extracting one block. The function B. B has two parameters: l, the size of the output, and k, the amount of independence used.

1. INPUT: $x \in \{0,1\}^n$; $y \in \{0,1\}^t$ (where $t = k \log n$).
2. Use y to choose a set $\{i_1, \ldots, i_l\} \subset \{1, \ldots, n\}$ of size l using k-wise independence, as described above.
3. OUTPUT (a vector in $\{0,1\}^l$): $x_{i_1}, \ldots, x_{i_l}$ (here x_j is the jth bit of x).

The next key lemma, Lemma 5.4, is proved in the Appendix.

LEMMA 5.4. *If D is a δ-source on $\{0,1\}^n$ and $\vec{X}$ is chosen according to D, then for all but an ϵ fraction of $y \in \{0,1\}^t$ the distribution of $B(\vec{X}, \vec{y})$ is within ϵ from a δ'-source. Here $\delta' = c\delta/\log \delta^{-1}$, $k \leq (\delta' l)^{1-\beta}$, and $\epsilon = (\delta' l)^{-c\beta k}$ for some sufficiently small positive constant c.*

5.3.1. The blockwise converter. We can now define our blockwise converter A. A has parameters $(l_1, \ldots, l_s)$, the lengths of the output blocks, and k, the amount of independence.

1. INPUT: $x \in \{0,1\}^n$; $y_1 \in \{0,1\}^{k \log n}, \ldots, y_s \in \{0,1\}^{k \log n}$.
2. For $i = 1, \ldots, s$ do $z_i \leftarrow B(x, y_i)$. (We use B with parameters l_i and k.)
3. OUTPUT: $z_1 \circ \cdots \circ z_s$.

Using essentially the same proof as in [NZ], we can show the following.

LEMMA 5.5. *Let $l_{min} = \min(l_1, \ldots, l_s)$, and suppose $k \leq (\delta' l_{min})^{1-\beta}$ and $l_1 + \cdots + l_s < \delta n/4$. Then A is an $(n, (l_1, \ldots, l_s), sk \log n, \delta, \delta', \epsilon)$ blockwise converter. Here $\delta' = c\delta/\log \delta^{-1}$ and $\epsilon = 2s(\delta' l_{min})^{-c'\beta k}$, where c' is from Lemma 5.4 and $c = c'/4$.*

In order to make the error small for the blockwise converter, we set the length of the smallest block $l_s = n^{\Theta(1)}$. This gives the following.

COROLLARY 5.6. *Suppose $l_1 + \cdots + l_s < \delta n/4$ and, for some constant $\beta > 0$, all $l_i \geq n^{\beta}(\log \epsilon^{-1})/\delta$. Then we can construct an efficient $(n, (l_1, \ldots, l_s), O(s \log \epsilon^{-1}), \delta, \delta', \epsilon)$ blockwise converter, where $\delta' = c\delta/\log \delta^{-1}$, c from Lemma 5.5.*

Proof. Choose $k = c''(\log \epsilon^{-1}/\log n)$ for a large enough constant c''. □

5.4. Choosing parameters and the basic extractor. It may help the reader, in the following discussion, to keep in mind the sample parameter values $\delta = n^{-1/4}$ and $\epsilon = 1/n$. The general parameter list for the extractor is given after Corollary 5.8 for reference.

From the discussion about the parameters of function C, all parameter lengths are determined by the smallest block-length l_s. We choose $l_s \geq n^{1/4}$ and large enough so that the error from C is small, but small enough to ensure $s \geq \log_{9/8}(4\delta l_s/r)$. Therefore, by the remark after the description of C, $r_0 = 4\delta' l_0$ and the output m is large enough. These choices are summarized below. E is our main extractor, obtained by combining the converter A with the blockwise extractor C and invoking Lemma 5.2 using the following values for the parameters.

Parameters of E for $\delta = n^{-1/4}$, $\epsilon = 1/n$, and $\beta = 1/4$.

1. The parameter n is given.
2. $\delta' = c\delta/\log \delta^{-1}$, where c is from Lemma 5.5. Thus $\delta' = \Theta(n^{-1/4}/\log n)$.
3. r is chosen to be the smallest integer such that $4 \cdot 2^{-r/16} \leq \epsilon/2$. So, $r = \Theta(\log n)$.
4. $l_s = n^{\beta/2} r/\delta'$; thus $l_s = \Theta(n^{3/8} \log^2 n)$.
5. Set $l_i = \max((r/4\delta')(9/8)^{s-i}, l_s)$.
6. s is chosen to be the largest integer such that $\sum_{i=1}^{s} l_i \leq \delta n/4$; thus $s = \Theta(\log n)$.
7. k is chosen so that $2s(\delta' l_s)^{-c'k\beta/2} \leq \epsilon/2$, where c' is from Lemma 5.4. So, $k = \Theta(1)$.
8. The length of the second parameter to E is given by $t = s(k \log n) + r$. Thus $t = \Theta(\log^2 n)$.
9. The length of the output of E is $m = 4\delta' l_0 - r = \Theta(\sqrt{n}/\log n)$.

Thus, by Lemmas 5.2 and 2.5, we deduce the following.

THEOREM 5.7. *For any $\beta > 0$ and any parameters $\delta = \delta(n)$ and $\epsilon = \epsilon(n)$ with $1/\sqrt{n} \le \delta \le 1/2$ and $2^{-\delta^2 n^{1-\beta}} \le \epsilon \le 1/n$, there is an efficient $(n, m = \delta^2 n/\log \delta^{-1}, t = O((\log n)\log \epsilon^{-1}), \delta, \epsilon)$-extractor.*

Proof. Using Lemma 5.2 gives output $m = \Omega(\delta^2 n/\log \delta^{-1})$; by applying Lemma 2.5 we can improve this to $m = \delta^2 n/\log \delta^{-1}$ while increasing t by a constant factor. □

Then Lemma 2.4 gives Corollary 5.8.

COROLLARY 5.8. *Any BPP algorithm can be simulated in $n^{O(\log n)}$ time using a δ-source, if the min-entropy of the R output bits is at least R^γ for any fixed $\gamma > 1/2$.*

We now present the parameter list of E in full generality.

Parameters of *E*. General case.

1. The parameters n, δ, and ϵ are given. We assume $1/\sqrt{n} \le \delta \le 1/2$ and for some constant $\beta > 0$, $2^{-\delta^2 n^{1-\beta}} \le \epsilon \le 1/n$.
2. $\delta' = c\delta/\log \delta^{-1}$, where c is from Lemma 5.5.
3. r is chosen to be the smallest integer such that $4 \cdot 2^{-r/16} \le \epsilon/2$. Thus $r = \Theta(\log \epsilon^{-1}) = O(\delta^2 n^{1-\beta})$.
4. $l_s = n^{\beta/2} r/\delta'$. We need $4\delta' l_s \ge r$ for function C. Also, $l_s = O(\delta n^{1-\beta/2} \log \delta^{-1})$.
5. Set $l_i = \max((r/4\delta')(9/8)^{s-i}, l_s)$.
6. s is chosen to be the largest integer such that $\sum_{i=1}^{s} l_i \le \delta n/4$. Since $s = O(\log n)$, $sl_s = o(\delta n)$; this and $l_0 = \Theta(\delta n)$ imply $(r/4\delta')(9/8)^s = \Theta(\delta n)$. Therefore $s \ge \log_{9/8}(4\delta' l_s/r)$, as required for the function C.
7. k is chosen so that $2s(\delta' l_s)^{-c'k\beta/2} \le \epsilon/2$, where c' is from Lemma 5.4. Since $\delta' l_s \ge n^{\beta/2}$, $k = \Theta((\log \epsilon^{-1})/\log n)$. Also, since $\delta' l_s \ge n^{\beta/2} \log \epsilon^{-1}$, $k \le (\delta' l)^{1-\beta/2}$.
8. The length of the second parameter to E is given by $t = s(k \log n) + r$. Thus $t = O((\log n)\log \epsilon^{-1})$.
9. The length of the output of E is given by $m = 4\delta' l_0 - r$. Thus $m = \Omega(\delta^2 n/\log \delta^{-1})$.

5.5. Bootstrapping to improve the extractor. We now use extractor E above recursively to get extractors which need fewer truly random additional bits, if δ is "much larger" than $n^{-1/2}$, say, $\delta = n^{-1/4}$. In particular, we show that BPP can be simulated in polynomial time if $\delta^{\log^* R} R = R^{\Omega(1)}$, where R is the number of random bits requested from the δ-source. Thus, taking $R \ge n^2$, say, as long as $\delta > n^{-1/\log^* n}$, we can simulate BPP in polynomial time; this is a significant extension of the work of [Zu2]. All of this follows from Lemma 5.9, which shows how, by bootstrapping, to get away with fewer truly random bits than E above needs. Basically, we replace one of the hash functions in the function C by the t bits output by an extractor. This way, we replace truly random bits by quasi-random bits that we extract from the source itself (i.e., we use the source's own randomness to further extract more bits). Therefore, instead of repeatedly hashing to build up an $n^{\Omega(1)}$-bit string, we need only build up a t-bit string and then apply the extractor.

LEMMA 5.9. *Suppose we are given an efficient $(n, (n_0, l_1, l_2, \ldots, l_{s-1}), t_1, \delta, \delta', \epsilon_1)$ blockwise converter A, an efficient $((l_1, \ldots, l_{s-1}), m_0, t_2, \delta', \epsilon_2)$ blockwise extractor C, and an efficient $(n_0, m, t_0 = m_0, \delta', \epsilon_3)$-extractor E. Then we can construct an efficient $(n, m, t_1 + t_2, \delta, \epsilon_1 + \epsilon_2 + \epsilon_3)$-extractor.*

Proof. Use A to add t_1 bits Y_1 and output a blockwise δ-source with blocks $X_0, X_1, \ldots, X_{s-1}$ with lengths $n_0, l_1, \ldots, l_{s-1}$. Use C to add t_2 bits Y_2 and convert $X_1, \ldots, X_{s-1}$ into a nearly uniform string Y_0 of length $m_0 = t_0$. As in the proof of

Lemma 5.3, the distribution of (X_0, Y_0, Y_1, Y_2) is within $\epsilon_1 + \epsilon_2$ of some distribution $D \times U$, where D is a δ'-source and U is the uniform distribution on $t_0 + t_1 + t_2$-bit strings. Therefore, by the extractor property for E, $(E(X_0, Y_0), Y_1, Y_2)$ is quasi-random to within $\epsilon_1 + \epsilon_2 + \epsilon_3$. □

Ideally, the extractor would add $O(\log \epsilon^{-1})$ truly random bits (for $\epsilon \leq 1/n$). The following corollary shows how an extractor using $u \log \epsilon^{-1}$ truly random bits can be improved to one using $O((\log u)(\log \epsilon^{-1}))$ additional bits.

COROLLARY 5.10. *Suppose n, δ, and ϵ are such that $1/\sqrt{n} \leq \delta \leq 1/2$ and for some constant $\beta > 0$, $2^{-\delta^2 n^{1-\beta}} \leq \epsilon \leq 1/n$. Set $n_0 = \delta n/8$ and $\delta' = c\delta/\log \delta^{-1}$, where c is from Lemma 5.5. Then, given an efficient $(n_0, m, t = u \log \epsilon^{-1}, \delta', \epsilon')$-extractor for $t \leq c'\delta n/\log n$ for a sufficiently small constant c', we can construct an efficient $(n, m, O((\log u)(\log \epsilon^{-1})), \delta, \epsilon + \epsilon')$-extractor.*

Proof. We first modify the blockwise extractor defined in subsection 5.2 so that its output is of length $t = u \log \epsilon^{-1}$. This requires only $s = O(\log u)$ blocks. More precisely, we define l_s and l_i as in subsection 5.4, but we choose s to ensure that the output length $r_0 - r = t$. Since $r = \Theta(\log \epsilon^{-1})$ and $r_{i-1}/r_i = 9/8$, this gives $s = O(\log u)$, as claimed. We therefore have an $((l_1, \ldots, l_s), t, O(\log \epsilon^{-1}), \delta', \epsilon/2)$ blockwise extractor.

We then use an $(n, (n_0, l_1, l_2, \ldots, l_{s-1}), O((\log u) \log \epsilon^{-1}), \delta, \delta', \epsilon/2)$ blockwise converter defined in subsection 5.3. Note that we have $n_0 + l_1 + l_2 + \cdots + l_s < \delta n/4$ as needed for Lemma 5.5. This inequality follows from $l_1 + l_2 + \cdots + l_s \leq s \cdot t \leq c' \cdot O(\delta n)$, and we can choose c' small enough. Now apply Lemma 5.9. □

Let $\log^{(k)}$ denote the logarithm iterated k times. We can now show the following.

THEOREM 5.11. *For any $\beta > 0$ and any parameters $\delta = \delta(n)$, $\epsilon = \epsilon(n)$, and $k = k(n)$ with $n^{-1/k} \leq \delta \leq 1/2$ and $2^{-\delta^k n^{1-\beta}} \leq \epsilon \leq 1/n$, there is an efficient $(n, m = \delta^k n/(\log \delta^{-1})^{2k-3}, t = O((\log^{(k-1)} n) \log \epsilon^{-1}), \delta, \epsilon)$-extractor. For the value $k = \log^* n - 1$, this gives an efficient $(n, m = (\delta/\log^2 \delta^{-1})^{\log^* n} n, t = O(\log \epsilon^{-1}), \delta, \epsilon)$-extractor for $\delta \geq n^{-1/2 \log^* n}$ and $2^{-\delta^{\log^* n} n^{1-\beta}} \leq \epsilon \leq 1/n$.*

Proof. Apply Corollary 5.10 repeatedly k times. Letting $m(n, \delta)$ denote the output length of the current extractor as a function of the input n and the quality δ, we see that each application of Corollary 5.10 causes the output length to decrease by a factor of $m(n_0, \delta')/m(n, \delta)$, which in our case is $\Theta(\delta/\log^2 \delta^{-1})$. Finally, use Lemma 2.5 to eliminate the Ω in front of the output m. □

COROLLARY 5.12. *For any constant $\gamma > 1 - 1/(k+1)$, any BPP algorithm can be simulated using a δ-source with min-entropy R^γ in time $n^{O(\log^{(k)} n)}$. For $\gamma > 1 - 1/(2 \log^* n)$, any BPP algorithm can be simulated using a δ-source in polynomial time.*

Remark. By applying random walks on expanders instead of k-wise independence, as in Lemma A.3, we can construct extractors for slightly smaller values of ϵ than given in Theorem 5.11: $\epsilon \geq 2^{-\delta^{2 \log^* n} n}$. However, this is not usually in the range of interest.

6. Simulating RP using a δ-source with min-entropy R^γ. Recall the RP simulation problem. We are given some fixed $\gamma > 0$, an error parameter $\kappa \in [0, 1)$, any r, and some hidden $W \subseteq \{0,1\}^r$ such that $|W| \geq 2^{r-1}$; we want to use a distribution on $\{0,1\}^R$ with min-entropy R^γ to produce a set of strings which intersects W with probability at least $1 - \kappa$. (Corollary 5.8 solves this for $\gamma > 1/2$; we now focus on an arbitrary fixed $\gamma > 0$.) Call this problem RPSIM(R, γ, κ, r). Since r will not change throughout our discussion (but R will), we let $T(R, \gamma, \kappa)$ denote its time complexity.

We shall focus on this problem for $R = R_0 \doteq r^{c(\gamma)}$, where the constant $c(\gamma)$ will be spelled out later; henceforth, R will denote an arbitrary integer in $[r^\gamma, R_0]$.

The difficulty in achieving any simulation with min-entropy less than $\sqrt{R}$ is that the basic extractor outputs a string of length less than $\delta^2 R$. One factor of δ is lost by the blockwise extractor, Lemma 5.3, and the other by the blockwise converter, Lemma 5.5. Our approach is to have the converter output a larger string with the hope of getting a larger blockwise source. If this works, we are done; if not, we show that the converter's output is a higher quality source than the original source. We can then proceed recursively.

This approach will lead to some problems with the error κ becoming too large. We handle this by using a remark in section 2.3 of [Zu2]. That remark implies the following useful inequality, which holds for any R, $\gamma' < \gamma$, and $\kappa < 1$:

$$T(R, \gamma, \kappa 2^{R^{\gamma'} - R^\gamma}) \leq T(R, \gamma', \kappa). \tag{4}$$

Here, it will be useful to think of κ as "large," i.e., close to 1. The bound (4) therefore says that we can get a high-quality solution for γ (i.e., the "error" $\kappa 2^{R^{\gamma'} - R^\gamma}$ is very low) as long as we can get even a rather low-quality solution (i.e., the error κ is "large") for an appropriate $\gamma' < \gamma$. Concretely, define $T_1(R, \gamma) \doteq T(R, \gamma, 1 - 1/\log^2 R)$ and $T_0(R, \gamma) \doteq T(R, \gamma, 1/\log^2 R_0)$ (the subscripts 0 and 1 refer to κ close to 0 and 1); recall that R denotes an arbitrary integer in $[r^\gamma, R_0]$. Then (4) shows, for instance, that

$$T_0(R, \gamma) \leq T_1(R, \gamma' = \gamma - 1/\log R_0). \tag{5}$$

Our approach will yield an algorithm upper bounding $T_1(R, \gamma')$ in terms of T_0, thus leading to a recurrence for T_0 via (5). Before that, we present some useful preliminaries.

6.1. Some useful results. Given random variables X and Y and elements x and y in the respective supports of X and Y, let $\mathcal{P}_{X,Y}(y|x)$ denote $Pr[(Y = y)|(X = x)]$. Given this, we can define $\mathcal{P}(Y|X)$ to be the random variable which, for all x and y in the respective supports of X and Y, takes on the deterministic value of $\mathcal{P}_{X,Y}(y|x)$ if $X = x$ and $Y = y$. Thus, for instance,

$$Pr_{X,Y}[\mathcal{P}(Y|X) > b] = \sum_{x,y:\ \mathcal{P}_{X,Y}(y|x) > b} Pr[(X = x) \wedge (Y = y)].$$

LEMMA 6.1. *Given a source outputting an R-bit string X with associated distribution D, partition $\{1, 2, \ldots, R\}$ into any two sets S_1 and S_2. Let X_1 and X_2 be the restrictions of X to S_1 and S_2, respectively, and let D_1 be the distribution induced on S_1. If $Pr_X[D(X) \leq 2^{-\ell}] \geq p$, then for any $p' \in [0, p]$, either $Pr_{X_1}[D_1(X_1) \leq 2^{-\ell/2}] \geq p'$ or $Pr_{X_1,X_2}[\mathcal{P}(X_2|X_1) \leq 2^{-\ell/2}] \geq p - p'$.*

Proof. Given any $x \in \{0,1\}^R$, let x_{S_1} and x_{S_2} denote its restrictions to S_1 and S_2, respectively. Now,

$$Pr_X[D(X) \leq 2^{-\ell}] = \sum_{x \in \{0,1\}^R:\ D(x) \leq 2^{-\ell}} D(x). \tag{6}$$

For any $x \in \{0,1\}^R$, $D(x) = D_1(x_{S_1}) \cdot \mathcal{P}_{X_1,X_2}(x_{S_2}|x_{S_1})$; thus, if $D(x) \leq 2^{-\ell}$, then $D_1(x_{S_1}) \leq 2^{-\ell/2}$ or $\mathcal{P}_{X_1,X_2}(x_{S_2}|x_{S_1}) \leq 2^{-\ell/2}$. Thus,

$$\sum_{x\in\{0,1\}^R:\ D(x)\le 2^{-\ell}} D(x) \le \sum_{x\in\{0,1\}^R:\ D_1(x_{S_1})\le 2^{-\ell/2}} D(x) + \sum_{x\in\{0,1\}^R:\ \mathcal{P}_{X_1,X_2}(x_{S_2}|x_{S_1})\le 2^{-\ell/2}} D(x). \tag{7}$$

However, by definition, the first and second terms in the right-hand side are, respectively, $Pr_{X_1}[D_1(X_1) \le 2^{-\ell/2}]$ and $Pr_{X_1,X_2}[\mathcal{P}(X_2|X_1) \le 2^{-\ell/2}]$. The lemma now follows from (6) and (7), using the given assumption that $Pr_X[D(X) \le 2^{-\ell}] \ge p$. □

LEMMA 6.2. *Suppose random variables $U \in \{0,1\}^{r_1}$ and $V \in \{0,1\}^{r_2}$ are such that $Pr_{U,V}[\mathcal{P}(V|U) > 2^{-\ell}] \le p$. Then, if $r_2 \ge \ell$, there is a random variable $W \in \{0,1\}^{r_2}$ such that* (i) *for all u and w in the respective supports of U and W, $Pr_{U,W}[(W = w)|(U = u)] \le 2^{-\ell}$; and* (ii) *the distribution of $U \circ V$ is within p of the distribution of $U \circ W$.*

Proof. Fix any u in the support of U, and let D_u denote the distribution of V conditional on $U = u$. Let $V_u = \{v : \mathcal{P}_{U,V}(v|u) > 2^{-\ell}\}$. Consider a distribution D'_u obtained by altering D_u such that for all $v \in V_u$, $D'_u(v) = 2^{-\ell}$; this is done by increasing the probabilities $D_u(v')$ for $v' \in \{0,1\}^{r_2} - V_u$ in some way. Now the condition $r_2 \ge \ell$ guarantees a way of doing this such that for all $v' \in \{0,1\}^{r_2} - V_u$, $D'_u(v') \le 2^{-\ell}$; it is now immediate that we have satisfied requirement (i) of the lemma.

In the above process, the only strings $u \circ v$ whose probabilities were decreased were those such that $\mathcal{P}_{U,V}(v|u) > 2^{-\ell}$. Now, it is easily seen that for any two distributions D_1 and D_2 on the same set S,

$$\|D_1 - D_2\| = \sum_{a\in S: D_1(a)>D_2(a)} (D_1(a) - D_2(a)) \le \sum_{a\in S: D_1(a)>D_2(a)} D_1(a).$$

Thus, since $Pr_{U,V}[\mathcal{P}(V|U) > 2^{-\ell}] \le p$ by assumption, requirement (ii) of the lemma is proved. □

6.2. The algorithm. We now present an algorithm for RPSIM$(R, \gamma' = \gamma - 1/\log R_0, 1 - 1/\log^2 R, r)$; we will bound its running time $T_1(R, \gamma')$ in terms of T_0. Fix a δ-source outputting R-bit strings with min-entropy $R^{\gamma'}$; thus, $\delta = \delta(R, \gamma')$ is given by $\delta R = R^{\gamma'}$ (so $\delta = R^{\gamma'-1}$). We may assume that the number of bits used to describe a hash function in the function C is $r = 4\delta l_s$, because a larger l_s was necessary only to reduce the error ϵ of the extractor. Since $r = \Theta(\log R)$, we have $l_s = \Theta(R^{1-\gamma'}/\log R)$. We also set $k = 2$ in the function B, i.e., use pairwise independence; thus, the error parameter ϵ in the statement of Lemma 5.4 is at most $R^{-\alpha}$, where α is a positive constant that depends only on γ. We also let δ' denote $c(\delta/2)/\log(2/\delta) = \Theta(\delta/\log R)$, where c is from the statement of Lemma 5.4. The block-lengths l_i are given by $l_i = l_s(9/8)^{s-i}$; we defer the presentation of s for now, but just note here that s will be $\Theta(\log R)$.

Since $k = 2$, the function B from subsection 5.3 uses strings of length $2\log R$ to index l-element subsets of $\{1, 2, \ldots, R\}$. For $y \in \{0,1\}^{2\log R}$, denote this subset by $S(l, y)$. Given $x \in \{0,1\}^R$ and any $S \subseteq \{1, 2, \ldots, R\}$, let x_S denote the sequence of bits of x indexed by S. Thus $B_l(x, y) = x_{S(y,l)}$ (note that we are subscripting the function B by the output length l). Recall that the output of the blockwise converter A, with parameters $(l_1, \ldots, l_s)$ and $k = 2$, is $A(x, (y_1, \ldots, y_s)) = B_{l_1}(x, y_1) \circ B_{l_2}(x, y_2) \circ \cdots \circ B_{l_s}(x, y_s)$.

The following lemma will be crucial.

LEMMA 6.3. *Let $\vec{X}$ denote a random string drawn from the given δ-source. For each i, $1 \le i \le s$, at least one of the following holds:*

(P1) *There exist $y_1, y_2, \ldots, y_i \in \{0,1\}^{2\log R}$ such that the distribution of $B_{l_1}(\vec{X}, y_1) \circ \cdots \circ B_{l_i}(\vec{X}, y_i)$ is within $i(\epsilon + 1/(3s))$ of the distribution of a blockwise δ'-source with block-lengths $l_1, l_2, \ldots, l_i$; or*

(P2) *there exist $y_1, y_2, \ldots, y_{i-1} \in \{0,1\}^{2\log R}$ such that the distribution of $B_{l_1}(\vec{X}, y_1) \circ \cdots \circ B_{l_{i-1}}(\vec{X}, y_{i-1})$ is within $1 - 1/(3s)$ of a distribution on $\{0,1\}^{l_1+l_2+\cdots+l_{i-1}}$ with min-entropy $R^{\gamma'}/2$.*

Proof. The proof is by induction on i. (P1) is true for the base case $i = 1$, by Lemma 5.4. We assume the lemma for $i = j \ge 1$ and prove for $i = j + 1$. If (P2) is true for $i = j$, so it is for $i = j + 1$; so we assume that (P1) is true for $i = j$ and prove the lemma for $i = j + 1$.

Let $y_1^*, y_2^*, \ldots, y_j^*$ be the values of $y_1, y_2, \ldots, y_j$ that make (P1) true for $i = j$. Let $S = S(y_1^*, l_1) \cup S(y_2^*, l_2) \cup \cdots \cup S(y_j^*, l_j)$, and let D_1 denote the distribution placed by the given δ-source on $\vec{X}_S$. Substituting $p = 1$, $p' = 1/(3s)$, $\ell = R^{\gamma'}$, $S_1 = S$, and $S_2 = \{1, 2, \ldots, n\} - S_1$ in Lemma 6.1, we see that one of two cases holds: (a) $Pr_{\vec{X}}[D_1(\vec{X}_S) \le 2^{-R^{\gamma'}/2}] \ge 1/(3s)$ or (b) $Pr_{\vec{X}}[\mathcal{P}(\vec{X}_{S_2} | \vec{X}_S) > 2^{-R^{\gamma'}/2}] \le 1/(3s)$.

If case (a) holds, we see by substituting $r_1 = 0$ in Lemma 6.2 that (P2) holds for $i = j + 1$, with $y_1^*, y_2^*, \ldots, y_j^*$ being the corresponding values of $y_1, y_2, \ldots, y_j$.

So, let us suppose case (b) holds. Then, it is easy to see that $Pr_{\vec{X}}[\mathcal{P}(\vec{X} | \vec{X}_S) > 2^{-R^{\gamma'}/2}] \le 1/(3s)$. So Lemma 6.2 shows that the distribution of $\vec{X}_S \circ \vec{X}$ is within $1/(3s)$ of the distribution of $\vec{X}_S \circ V$, where (i) $V \in \{0,1\}^R$; and (ii) for all x in the support of $\vec{X}_S$ and for all $v \in \{0,1\}^R$, $Pr[(V = v) | (\vec{X}_S = x)] \le 2^{-(\delta/2)R}$. Thus, by Lemma 5.4, there is at least one $y_{j+1}^* \in \{0,1\}^{2\log R}$ such that the distribution of $\vec{X}_S \circ \vec{X}_{S(y_{j+1}^*, l_{j+1})}$ is within $\epsilon + 1/(3s)$ of the distribution of $\vec{X}_S \circ V'$, where (i′) $V' \in \{0,1\}^{l_{j+1}}$; and (ii′) for all x in the support of $\vec{X}_S$ and for all $v' \in \{0,1\}^{l_{j+1}}$, $Pr[(V' = v') | (\vec{X}_S = x)] \le 2^{-\delta' l_{j+1}}$. This, combined with the inductive assumption that $y_1^*, \ldots, y_j^*$ are values of $y_1, \ldots, y_j$ that make (P1) true for $i = j$, shows that (P1) is also true for $i = j + 1$, with $y_1 = y_1^*, y_2 = y_2^*, \ldots, y_{j+1} = y_{j+1}^*$. □

Recall that if (P2) is true for i, it is also true for $i + 1$. Thus, substituting $i = s$ in Lemma 6.3 and noting that the min-entropy does not decrease if we add more bits, we deduce the following.

COROLLARY 6.4. *There exist $y_1, y_2, \ldots, y_s \in \{0,1\}^{2\log R}$ such that the distribution of $A(\vec{X}, (y_1, \ldots, y_s)) = B_{l_1}(\vec{X}, y_1) \circ B_{l_2}(\vec{X}, y_2) \circ \cdots \circ B_{l_s}(\vec{X}, y_s)$ is either* (a) *within $s\epsilon + 1/3 = 1/3 + o(1)$ of the distribution of a blockwise δ'-source with block-lengths $l_1, l_2, \ldots, l_s$; or* (b) *within $1 - 1/(3s)$ of a distribution on $\{0,1\}^{l_1+l_2+\cdots+l_s}$ with min-entropy $R^{\gamma'}/2$.*

Recall that we want an algorithm for RPSIM$(R, \gamma', 1 - 1/\log^2 R, r)$. Also recall that $l_i = l_s(9/8)^{s-i}$; we now choose $s = \Theta(\log R)$ as the largest integer such that $\sum_{i=1}^{s} l_i \le R^{1-\gamma'/2}$. We first apply our function C (we deterministically cycle through all the $R^{O(1)}$ possible choices for the random input seed for C) one by one on $\vec{X}_{S(y_1,l_1)} \circ \vec{X}_{S(y_2,l_2)} \circ \cdots \circ \vec{X}_{S(y_s,l_s)}$, for all the $R^{O(\log R)}$ possible choices for $(y_1, y_2, \ldots, y_s)$. Thus, if case (a) of Corollary 6.4 were true, at least one of the strings output would be quasi-random to within $1/3 + o(1)$. Note that all the output strings will have length $\Omega(\delta' l_1) = \Omega(R^{\gamma/2}/\log R)$. Thus, as long as this is at least r, the probability of at least one of the output strings hitting W is at least $1/2 - 1/3 - o(1) = 1/6 - o(1)$, which is much greater than the required $1 - (1 - 1/\log^2 R) = 1/\log^2 R$ for RPSIM$(R, \gamma', 1 - 1/\log^2 R, r)$.

However, since we do not know if case (a) of Corollary 6.4 holds, we also have to consider the remaining possibility (case (b) of Corollary 6.4) that there exist $y_1, y_2, \ldots, y_s \in \{0,1\}^{2\log R}$ such that the distribution of $A(\vec{X}, (y_1, \ldots, y_s)) = B_{l_1}(\vec{X}, y_1) \circ B_{l_2}(\vec{X}, y_2) \circ \cdots \circ B_{l_s}(\vec{X}, y_s)$ is within $1 - 1/(3s)$ of a distribution on $\{0,1\}^{l_1+\cdots+l_s}$ with min-entropy $R^{\gamma'}/2$. To handle this possibility, we once again exhaustively consider all the $R^{O(\log R)}$ possible choices for $(y_1, y_2, \ldots, y_s)$; for each such choice we run RPSIM$(R^{1-\gamma'/2}, \frac{\gamma' - 1/\log R}{1-\gamma'/2}, 1/\log^2 R_0, r)$ on $A(\vec{X}, (y_1, \ldots, y_s))$. Thus, if case (b) of Corollary 6.4 holds, then by the definition of RPSIM, we will hit W with probability at least $1 - (1 - 1/(3s)) - 1/\log^2 R_0 = \Theta(1/\log R)$, which is again greater than the required $1/\log^2 R$ for RPSIM$(R, \gamma', 1 - 1/\log^2 R, r)$.

The total time taken is

$$R^{O(\log R)} + R^{O(\log R)} T\left(R^{1-\gamma'/2}, \frac{\gamma' - 1/\log R}{1-\gamma'/2}, 1/\log^2 R_0\right),$$
$$= R^{O(\log R)} T_0\left(R^{1-\gamma'/2}, \frac{\gamma' - 1/\log R}{1-\gamma'/2}\right).$$

Thus, noting that $\frac{\gamma' - 1/\log R}{1-\gamma'/2} = (\gamma - \Theta(1/\log r))/(1 - \gamma/2)$, the time complexity of this algorithm can be summarized as

$$T_1(R, \gamma') \leq R^{O(\log R)} T_0(R^{1-\gamma'/2}, (\gamma - \Theta(1/\log r))/(1 - \gamma/2)).$$

Combining with (5), we see that

$$T_0(R, \gamma) \leq R^{O(\log R)} T_0(R^{1-\gamma'/2}, (\gamma - \Theta(1/\log r))/(1 - \gamma/2)). \tag{8}$$

The termination condition for this recurrence given by our extractor E is, say, $T_0(R, 2/3) = r^{O(\log r)}$. Letting $\gamma_0 = \gamma > 0$ be the initial value of γ, the sequence of values taken by the second argument in (8) is given by $\gamma_{i+1} \geq (\gamma_i - \Theta(1/\log r))/(1 - \gamma_i/2)$. Since the $\Theta(1/\log r)$ term can be made sufficiently small relative to γ_0 by taking r large enough, it is not hard to prove by induction on i that $\gamma_i \geq \gamma_0/(1-\gamma_0/4)^i$. Hence, the termination condition $\gamma_i \geq 2/3$ is achieved after a constant number of iterations, for any given constant $\gamma_0 > 0$; thus, $T_0(R, \gamma) = r^{O(\log r)}$. It is also not hard to check that all the output strings have length $\Omega(R^{\gamma/2}/\log R)$. Thus, there are constants $c_1(\gamma)$ and $c_2(\gamma)$ such that as long as we choose R_0 such that $R_0^{\gamma/2}/\log R_0 \geq c_1(\gamma) r$, i.e., as long as $R_0 \geq c_2(\gamma)(r \log r)^{2/\gamma}$, then all the output strings will have length at least r, which suffices for the above process to work.

THEOREM 6.5. *Any RP algorithm can be simulated in $n^{O(\log n)}$ time using a δ-source, if the min-entropy of the R output bits is at least R^γ for any fixed $\gamma > 0$. Correspondingly, there is an efficient construction of a $(2^n, 2^{\Omega(n^{\gamma/2}/\log n)}, n^{O(\log n)}, 2^{n^\gamma})$-disperser.*

7. Sources with many weakly independent bits. As an application of Theorem 4.2, we now show how to simulate BPP using a generalization of the oblivious bit-fixing source of [CW], using Lemma 4.1. Here again, we actually build an extractor for these sources. We need the following definition.

DEFINITION 7.1. *A bit X_i from a distribution on $X_1 X_2, \ldots, X_n$ has* weak independence α *if α is the maximum value in $[0, 1/2]$ such that for every setting $X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, X_{i+1} = x_{i+1}, \ldots, X_n = x_n$ of all the other bits,*

$$\alpha \leq Pr[X_i = 0 | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, X_{i+1} = x_{i+1}, \ldots, X_n = x_n] \leq 1 - \alpha.$$

Thus, a bit of the oblivious bit-fixing source that is not fixed has weak independence 1/2. Note the difference between this definition and the semirandom source of [SV]: we look at a bit conditioned on all the other bits, not just on the previous bits. Indeed, it is not necessarily true that every bit of a semirandom source with parameter α has weak independence close to α.

THEOREM 7.2. *For a source S outputting n bits $(X_1, X_2, \ldots, X_n)$, let α_i denote the weak independence α_i of bit X_i. For any fixed $\gamma > 0$, there is an efficient (explicitly given) extractor $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ for the class of sources with $\sum_{i=1}^n \alpha_i \geq n^\gamma$, where $t = O(\log n)$ and $m = n^{\Omega(1)}$; the output of the extractor is $n^{-\Theta(1)}$-quasi-random. The extractor need only know the value γ and not the quantities α_i.*

Note that this is best possible: if the X_i's are independent with $Pr(X_i = 0) = \alpha_i$, then the entropy of $(X_1, X_2, \ldots, X_n)$ is $k \doteq \sum_{i=1}^n H(\alpha_i)$, where $H(x) = -x \log_2 x - (1-x)\log_2(1-x)$ is the usual binary entropy function, with $H(0) = H(1) \doteq 0$. If $\sum_{i=1}^n \alpha_i = \beta$, then k is maximized when each α_i equals β/n, by the concavity of H; so $k = O(\beta \log(n/\beta))$. Thus if $\beta = n^{o(1)}$, then the entropy of $(X_1, X_2, \ldots, X_n)$ is also $n^{o(1)}$; hence such an extractor construction would not be possible. Thus we indeed need $\sum_{i=1}^n \alpha_i \geq n^\gamma$, for some fixed $\gamma > 0$.

Proof. Although the min-entropy is at least n^γ, we do not know as much about the "location" of the "good bits" as we do for Chor–Goldreich sources. We proceed by showing how to use $O(\log n)$ purely random bits to obtain a source that is a Chor–Goldreich source with high probability; the theorem then follows from Lemma 4.1. The idea is to take a pairwise independent permutation of the bits as in [Zu2] and then divide our string into blocks. We then argue that many weakly independent bits fall in each block. The reason we need weak independence is because a bit's weak independence does not change if the bits are permuted. Thus, we do not need to pick the blocks independently, as in [NZ], or use more complicated methods, as in [Zu2].

Assume, without loss of generality, that n is prime and that $\sum_{i=1}^n \alpha_i = n^\gamma$, and associate the finite field on n elements with $\{1, 2, \ldots, n\}$. Pick a to be a random *nonzero* element of the field and b to be a random field element; the map π is then $\pi(i) = ai + b$. Since $a \neq 0$, π is a permutation. Divide the n bits into $m = n^{\gamma/3}$ blocks $B_1, \ldots, B_m$ of length $l = n^{1-\gamma/3}$, according to π; i.e.,

$$B_i = (X_{\pi^{-1}((i-1)l+1)}, X_{\pi^{-1}((i-1)l+2)}, \ldots, X_{\pi^{-1}(il)}).$$

It suffices to show that with high probability, each of these blocks gets many weakly independent bits: this will give a Chor–Goldreich source. Fix $i \in \{1, 2, \ldots, m\}$ arbitrarily. Define the random variable W_i as the weak independence of block B_i, i.e., $\sum_{j=1}^n Y_j$, where $Y_j = \alpha_j$ if $\pi(j) \in \{(i-1)l+1, (i-1)l+2, il\}$, and 0 otherwise. Note that $E[Y_j] = \alpha_j l/n = \alpha_j/n^{\gamma/3}$ and hence, $E[W_i] = \sum_{j=1}^n E[Y_j] = n^{2\gamma/3}$. Now since π is a pairwise independent permutation, $Pr[\pi(i_1) = j_1 \text{ and } \pi(i_2) = j_2] = 1/(n(n-1))$, for any distinct i_1 and i_2, and any distinct j_1 and j_2. Thus for any j, k, $j \neq k$,

$$E[Y_j Y_k] = |B_i||B_i - 1|\frac{\alpha_j \alpha_k}{n(n-1)} \leq \frac{\alpha_j \alpha_k}{n^{2\gamma/3}} = E[Y_j]E[Y_k].$$

Thus, the variance $Var[W_i]$ of W_i is

$$\begin{aligned} Var[W_i] &= \sum_j (E[Y_j^2] - (E[Y_j])^2) + 2\sum_{j<k}(E[Y_j Y_k] - E[Y_j]E[Y_k]) \\ &\leq \sum_j (E[Y_j^2] - (E[Y_j])^2) \leq \sum_j E[Y_j^2] = \sum_j \frac{\alpha_j^2}{n^{\gamma/3}} \leq n^{2\gamma/3}/2. \end{aligned}$$

So, using Chebyshev's inequality, $Pr[W_i < n^{2\gamma/3}/2] \leq 2n^{-2\gamma/3}$ and hence, $Pr[(\exists i)W_i < n^{2\gamma/3}/2] \leq 2n^{-\gamma/3}$. Hence, with probability at least $1 - 2n^{-\gamma/3}$, we have the output of a Chor–Goldreich source with min-entropy $n^{\Omega(1)}$. Thus, if we now run the extractor of Lemma 4.1 on this output, the quasi-randomness of the final output is at most $\epsilon' + 2n^{-\gamma/3} = n^{-\Theta(1)}$, where $\epsilon' = n^{-\Theta(1)}$ is the amount of quasi-randomness introduced by the extractor of Lemma 4.1. □

8. Applications. Our applications rely heavily on previous work involving these applications. It is often helpful to use the graph-theoretic, or disperser, view of our results.

8.1. Time-space tradeoffs. Our first application is to time-space tradeoffs. Sipser defined the class strong-RP [Sip] as follows.

DEFINITION 8.1. *$A \in$ strong-RP if there is an RP machine accepting A using $q(n)$ random bits and achieving an error probability of at most $2^{-(q(n)-q(n)^\alpha)}$ for some fixed $\alpha < 1$.*

He then showed the following.

THEOREM 8.2 (see [Sip]). *P equals strong-RP or, for some $\epsilon > 0$ and for any time bound $t(n) \geq n$, all unary languages in DTIME($t(n)$) are accepted infinitely often in SPACE($t(n)^{1-\epsilon}$).*

We would like to replace strong-RP in the above theorem by RP. Note the relevance of δ-sources to strong-RP as follows.

LEMMA 8.3. *Strong-RP equals RP if and only if RP can be simulated using a δ-source with min-entropy R^α for some $\alpha < 1$. (For the equivalence, we assume nonoblivious simulations; i.e., the simulation could be different for different languages.)*

Proof of Lemma 8.3. Let $L \in RP$, and suppose M recognizes L using a δ-source with min-entropy R^α for some $\alpha < 1$. Then M errs on fewer than 2^{R^α} R-bit strings. Setting $q(n) = R$ shows that M is a strong-RP machine recognizing L. Conversely, suppose strong-RP equals RP, and again let $L \in RP$. Say M accepts L with error probability at most $2^{-(q(n)-q(n)^\alpha)}$ for some $\alpha < 1$. Then M errs on at most $2^{q(n)^\alpha}$ strings. Thus for a fixed β, $\alpha < \beta < 1$, M accepts L with error probability at most $2^{q(n)^\alpha - q(n)^\beta}$ if the random bits come from a δ-source with min-entropy R^β. □

As we did not quite show that RP equals strong-RP, substituting our result into Sipser's proof gives the following.

THEOREM 8.4. *$RP \subseteq \bigcap_k DTIME(n^{\log^{(k)} n})$ or, for any time bound $t(n) \geq n$, all unary languages in $DTIME(t(n))$ are accepted infinitely often in $\bigcap_k SPACE(t(n)^{1-1/\log^{(k)} n})$.*

8.2. Explicit expanders and related problems. Our second application is to improving the expanders constructed in [WZ], and hence all the applications given there. Call an N-vertex undirected graph N^δ-expanding if there is an edge connecting every pair of disjoint subsets of the vertices, of size N^δ each. In [WZ], such graphs with essentially optimal maximum degree $N^{1-\delta+o(1)}$ were constructed in polynomial time. They were used to explicitly construct some useful combinatorial structures, as mentioned in section 1. All of these results are optimal to within factors of $N^{o(1)}$. In [WZ], these $N^{o(1)}$ factors were $2^{(\log N)^{2/3+o(1)}}$. Our results improve these $N^{o(1)}$ factors to $2^{(\log N)^{1/2+o(1)}}$.

We first borrow the following lemmas from the final version of [WZ].

LEMMA 8.5 (see [WZ]). *If there is an $(n, m, t, \delta, 1/4)$-extractor computable in linear space, then there is an N^δ-expanding graph on $N = 2^n$ nodes with maximum degree $N2^{1+2t-m}$ constructible in Logspace.*

The next lemma shows how to modify an extractor so it extracts almost all the randomness of a δ-source. The intuition is that if x is output from a δ-source and the output of the extractor $E(x,y)$ has length $m=\beta n$, then the string $E(x,y)\circ x$ is close in distribution to a uniform m-bit string concatenated with a $(\delta-\beta)$-source. Thus, we can apply an extractor E' for a $(\delta-\beta)$-source with an independent t-bit string y', and output $E(x,y)\circ E'(x,y')$. The following is based on recursing on this idea.

LEMMA 8.6 (see [WZ]). *Fix positive integers n and k. Suppose that for each $\delta\in[\eta,1]$ we are given an efficient $(n,m(\delta),t(\delta),\delta,\epsilon(\delta))$-extractor, where t and ϵ are nonincreasing functions of δ. Let $f(\delta)=m(\delta)/(\delta n)$. Let $r=\ln(\delta/\eta)/f(\eta)$ or, if f grows at least linearly (i.e., $f(c\delta)\geq cf(\delta)$), let $r=2/f(\eta)$. Then we can construct an efficient $(n,(\delta-\eta)n-k,r\cdot t(\eta),\delta,r(\epsilon(\eta)+2^{-k}))$-extractor.*

We can now prove our improved construction.

THEOREM 8.7. *There is a polynomial-time algorithm that, on input N (in unary) and δ, where $0<\delta=\delta(N)<1$, constructs N^{δ}-expanding graphs on N nodes with maximum degree $N^{1-\delta}2^{(\log N)^{1/2+o(1)}}$.*

Proof. Assume, without loss of generality, that N is a power of 2, with $N=2^n$. Set $\eta=(\log^3 n/n)^{1/2}$, $\epsilon=1/n$, and $k=\log n$. If $\delta<2\eta$, then the complete graph satisfies the theorem. Otherwise, apply Lemma 8.6 to the extractor given by Theorem 5.7 to build an

$$(n,m=(\delta-\eta)n-\log n,t=O(\log^2 n\log\eta^{-1}/\eta),\delta,\epsilon=O(1/\eta n))\text{-extractor.}$$

Then Lemma 8.5 gives an N^{δ}-expanding graph with maximum degree $N^{1-\delta}2^{O(n\eta)}$, i.e., $N^{1-\delta}2^{(\log N)^{1/2+o(1)}}$. □

8.3. The hardness of approximating NP-hard problems. Our third application is to the hardness of approximating $\log\log\omega(G)$, where $\omega(G)$ is the clique number of G. In [Zu2], it was shown that if $N\tilde{P}\neq\tilde{P}$, then approximating $\log\omega(G)$ to within any constant factor is not in $\tilde{P}$ (recall that $\tilde{P}$ denotes quasi-polynomial time). In [Zu3], a randomized reduction was given showing that any iterated log is hard to approximate; in particular, if $N\tilde{P}\neq ZP\tilde{P}$, then approximating $\log\log\omega(G)$ to within a constant factor is not in $co-R\tilde{P}$. This used the fact that with high probability, certain graphs are dispersers. The disperser implied by our RP construction is almost as good. This makes the last reduction above deterministic, with a slight loss of efficiency: if $N\tilde{P}\not\subseteq DTIME(2^{(\log n)^{O(\log\log n)}})$, then approximating $\log\log\omega(G)$ to within any constant factor is not in $\tilde{P}$.

Let quasi-poly(x) be shorthand for $2^{(\log x)^{O(1)}}$. We now present a lemma, which is implicit in the results of [Zu2, Zu3] on the hardness of approximation.

LEMMA 8.8 (see [Zu2, Zu3]). *Suppose there is an explicit construction of an $(N=N(n),n^{\Theta(1)},d=d(n),K=K(n))$-disperser, for all integers n. Let $g_1,g_2\colon[1,\infty)\to\Re^+$ be functions satisfying $g_1(y)\leq g_2(y)$ for all $y\in[1,\infty)$. Suppose that for any input graph G, a number $h(G)\in[g_1(\omega(G)),g_2(\omega(G))]$ can be computed in $\tilde{P}$. Then if $g_1(N)>g_2(K)$, we have $NP\subseteq DTIME($ quasi-poly $(N+2^d))$.*

THEOREM 8.9. *If $N\tilde{P}\not\subseteq DTIME(2^{(\log n)^{O(\log\log n)}})$, then approximating $\log\log\omega(G)$ to within any constant factor is not in $\tilde{P}$. In other words, if we can compute, for some fixed $t>1$, a number in the range*

$$[2^{(\log\omega(G))^{1/t}},2^{(\log\omega(G))^{t}}]$$

in $\tilde{P}$, then $N\tilde{P}\subseteq DTIME(2^{(\log n)^{O(\log\log n)}})$.

Proof. For any fixed $\gamma \in (0,1]$, Theorem 6.5 implies that an $(N, n^{\Theta(1)}, d, K)$-disperser is efficiently constructible in the notation of Lemma 8.8, where $N = 2^{(\log n)^{O(1/\gamma)}}$, $d = (\log n)^{O(\log\log n)}$, and $K = 2^{(\log N)^{\gamma}}$. Thus, by taking $\gamma < 1/t^2, g_1(y) = 2^{(\log y)^{1/t}}$, and $g_2(y) = 2^{(\log y)^t}$, we invoke Lemma 8.8 to conclude that NP and hence $N\tilde{P}$, by a simple padding argument, is contained in $DTIME(2^{(\log n)^{O(\log\log n)}})$. □

9. Later work and open problems. Some of our main contributions—the improved Leftover Hash Lemma and its use in extractors—have served as building blocks for several recent results. First, Saks, Srinivasan, and Zhou [SSZ] have improved our RP simulation to $poly(n)$ time. Second, substantial progress on the BPP simulation question has been made by Ta-Shma [Ta-S], where an $R^{O(\log^{(k)} R)}$ algorithm is given for every fixed positive integer k. Third, ideas from this paper have been extended by Zuckerman to give optimal extractors for constant-rate sources, as well as randomness-optimal samplers [Zu4]. In an exciting new result, Andreev et al. have shown how to simulate BPP using δ-sources with min-entropy R^{γ} for any fixed $\gamma > 0$, in polynomial time [AC+].

An important open question is to efficiently construct, for the class of δ-sources with min-entropy R^{γ} for any fixed $\gamma > 0$, efficient extractors which use $O(\log R)$ purely random bits to extract as many as $(1 - o(1))R^{\gamma}$ bits, which are quasi-random to within $R^{-\Theta(1)}$. It is easy to show that such extractors exist nonconstructively. In [Ta-S] it is shown that polylog(R) bits suffice to do this. Constructing a near-optimal family of dispersers may be an interesting step in this direction.

See [Nis] for a survey of some of the recent results in this area.

Appendix. Details of the blockwise converter. A property of such k-wise independent random variables that we will require follows.

LEMMA A.1. *Let $T \subseteq \{1, 2, \ldots, n\}$, $|T|/n \geq \delta$. Suppose k is even, $4 \leq k \leq (\delta l)^{1-\beta}/8$ for some $\beta > 0$. If S is chosen at random as described above, then*

$$Pr[|S \cap T| \leq \delta l/2] \leq 8(\delta l)^{-\beta k/2}.$$

We use the following lemma from [BR].

LEMMA A.2. *For $k \geq 4$ an even integer, let $Y_1, \ldots, Y_l$ be k-wise independent* 0-1 *random variables, $Y = \sum_{i=1}^{l} Y_i$, and $\mu = E[Y]$. Then for $\alpha > 0$, $Pr[|Y - \mu| \geq \alpha] \leq 8((k\mu + k^2)/\alpha^2)^{k/2}$.*

Proof of Lemma A.1. Define the random variables Y_i to be 1 if and only if $X_i \in T$, and 0 otherwise. Then, $E[Y_i] = |T \cap A_i|/m$. Thus for $Y = \sum_{i=1}^{l} Y_i$, $E[Y] = \sum_{i=1}^{l} |T \cap A_i|/m = |T|/m \geq \delta l$. Setting $\alpha = \delta l/2$ in Lemma A.2 concludes the proof. □

We remark that for certain parameters of the extractor (e.g., if $\delta = \Omega(1)$), it may be better to use the following lemma from [NZ]. The parameters where this is useful are mostly uninteresting: $\epsilon = 2^{-\delta^2 n^{1-o(1)}}$.

LEMMA A.3 (see [NZ]). *There is an absolute constant $c > 0$ such that the following holds: Suppose $ck \leq \delta^2 l$. Then we can use $O(k/\delta + \log n)$ random bits to pick l random variables $X_1, \ldots, X_l$ in $\{1, 2, \ldots, n\}$ such that $Pr[\geq \delta^2 l/16$ of the X_i's lie in $T] \geq 1 - 2^{-k}$.*

Proof of Lemma 5.4**.** To prove Lemma 5.4, we proceed as in [NZ]. Fix a δ-source D. We need the following definitions that are relative to D.

DEFINITION A.4. *For $\vec{x} \in \{0,1\}^n$ and $1 \leq i \leq n$, let $p_i(\vec{x}) = Pr_{\vec{X} \in D}[X_i = x_i | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}]$. Index i is called good in $\vec{x}$ if $p_i(\vec{x}) < 1/2$ or if $p_i(\vec{x}) = 1/2$ and $x_i = 0$.*

The part of the definition with $p_i(\vec{x}) = 1/2$ ensures that exactly one of $x_i = 0$ and $x_i = 1$ is good, for a given prefix.

DEFINITION A.5. *$\vec{x}$ is α-good if there are at least αn indices which are good in x. For $S \subseteq \{1, 2, \ldots, n\}$, $\vec{x}$ is α-good in S if there are at least $\alpha|S|$ indices in S which are good in $\vec{x}$; S is α-informative to within β if $Pr_{\vec{X} \in D}[\vec{X}$ is α-good in $S] \geq 1 - \beta$.*

Denote by S_y the set of l indices chosen using the (k-wise independent) random bits $\vec{y}$, as described in section 5.3. A useful result shown in [NZ] is that for any set of indices $\{i_1, \ldots, i_l\}$ that is δ'-informative to within ϵ, the distribution of $X_{i_1}, \ldots, X_{i_l}$ induced by choosing $\vec{X}$ according to D is ϵ-near a δ'-source. This result, together with Lemma A.6, will clearly prove Lemma 5.4.

LEMMA A.6. *$Pr_{\vec{Y}}[S_Y$ is δ'-informative to within $\epsilon] \geq 1 - \epsilon$.*

Proof. We first need the following result from [NZ]:

$$Pr_{\vec{X} \in D}[\vec{X} \text{ is not } \alpha\text{-good}] \leq 2^{-c_1 \delta n}, \tag{9}$$

where $\alpha = c_1 \delta / \log \delta^{-1}$ for some absolute positive constant c_1.

For any fixed α-good string $\vec{x}$, we can apply Lemma A.1 to the set of good indices and obtain $Pr_Y[\vec{x}$ has $\leq \alpha l/2$ good indices in $S_Y] \leq 8(\alpha l)^{-\beta k/2}$. Using (9), it follows that

$$Pr_{\vec{X},Y}[\vec{X} \text{ has } \leq \alpha l/2 \text{ good indices in } S_Y] \leq 8(\alpha l)^{-\beta k/2} + 2^{-c_1 \delta n}.$$

Set $\delta' = \alpha/2$ and $\epsilon = \sqrt{8(\alpha l)^{-\beta k/2} + 2^{-c_1 \delta n}}$. We will now use Markov's inequality in the following way: Let $A_y = Pr_{\vec{X} \in D}[\vec{X}$ is not δ'-good in $S_y]$. Thus A_Y is a random variable determined by Y. From the above analysis, $E_Y[A_Y] \leq \epsilon^2$. Therefore, by Markov's inequality, $Pr_Y[A_Y \geq \epsilon] \leq \epsilon$. In other words, $Pr_Y[S_Y$ is δ'-informative to within $\epsilon] \geq 1 - \epsilon$. □

Note that in the sources considered in section 4, we know that each block has "many" good bits; thus, since we know the block boundaries, working with the good bits was much easier there. Since we have no idea of the location of good bits in general δ-sources, we have to work much harder here.

Acknowledgments. We thank Avi Wigderson for helpful discussions, and the two referees for their detailed and helpful suggestions.

REFERENCES

[ABI] N. ALON, L. BABAI, AND A. ITAI, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, J. Algorithms, 7 (1986), pp. 567–583.

[AC+] A. E. ANDREEV, A. E. F. CLEMENTI, J. P. D. ROLIM, AND L. TREVISAN, *Weak random sources, hitting sets, and BPP simulations*, in Proc. 38th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 264–272.

[AG+] N. ALON, O. GOLDREICH, J. HÅSTAD, AND R. PERALTA, *Simple constructions of almost k–wise independent random variables*, Random Structures Algorithms, 3 (1992), pp. 289–303.

[BR] M. BELLARE AND J. ROMPEL, *Randomness-efficient oblivious sampling*, in Proc. 35th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 276–287.

[BL] M. BEN-OR AND N. LINIAL, *Collective coin flipping*, in Advances in Computing Research 5: Randomness and Computation, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 91–115.

[Blu] M. BLUM, *Independent unbiased coin flips from a correlated biased source: A finite Markov chain*, Combinatorica, 6 (1986), pp. 97–108.

[CG] B. Chor and O. Goldreich, *Unbiased bits from sources of weak randomness and probabilistic communication complexity*, SIAM J. Comput., 17 (1988), pp. 230–261.

[CG+] B. Chor, O. Goldreich, J. Håstad, J. Friedman, S. Rudich, and R. Smolensky, *The bit extraction problem or t-resilient functions*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1985, pp. 396–407.

[CW] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification, and weak random sources*, in Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 14–19.

[DFK] M. Dyer, A. Frieze, and R. Kannan, *A random polynomial time algorithm for approximating the volume of a convex body*, J. ACM, 38 (1991), pp. 1–17.

[FLW] A. M. Ferrenberg, D. P. Landau, and Y. J. Wong, *Monte Carlo simulations: Hidden errors from "good" random number generators*, Phys. Rev. Lett., 69 (1992), pp. 3382–3384.

[GW] O. Goldreich and A. Wigderson, *Tiny families of functions with random properties: A quality-size trade-off for hashing*, Random Structures Algorithms, 11 (1997), pp. 315–343.

[HRD] T.-S. Hsu, V. Ramachandran, and N. Dean, *Parallel implementation of algorithms for finding connected components in graphs,* in Parallel Algorithms, 3rd DIMACS Implementation Challenge, October 17–19, 1994, DIMACS, Ser. Discrete Math. Theor. Comput. Sci. 30, Sandeep N. Bhatt, ed., AMS, Providence, RI, 1997, pp. 23–41.

[Hsu] T.-S. Hsu, *Graph Augmentation and Related Problems: Theory and Practice*, Ph.D. thesis, Department of Computer Sciences, University of Texas at Austin, Austin, TX, October 1993.

[ILL] R. Impagliazzo, L. Levin, and M. Luby, *Pseudo-random generation from one-way functions*, in Proc. 21st Annual ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 12–24.

[IZ] R. Impagliazzo and D. Zuckerman, *How to recycle random bits*, in Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 248–253.

[KKL] J. Kahn, G. Kalai, and N. Linial, *The influence of variables on Boolean functions*, in Proc. 29th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 68–80.

[LLS] D. Lichtenstein, N. Linial, and M. Saks, *Some extremal problems arising from discrete control processes*, Combinatorica, 9 (1989), pp. 269–287.

[LN] R. Lidl and H. Niederreiter, *Finite Fields*, Addison-Wesley, Reading, MA, 1983.

[Lub] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.

[NN] J. Naor and M. Naor, *Small–bias probability spaces: Efficient constructions and applications*, SIAM J. Comput., 22 (1993), pp. 838–856.

[Nis] N. Nisan, *Extracting randomness: How and why*, in Proc. IEEE Conference on Computational Complexity (formerly Structure in Complexity Theory), IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 44–58.

[NZ] N. Nisan and D. Zuckerman, *Randomness is linear in space*, J. Comput. System Sci., 52 (1996), pp. 43–52.

[San] M. Santha, *On using deterministic functions in probabilistic algorithms*, Inform. Comput., 74 (1987), pp. 241–249.

[SV] M. Santha and U. Vazirani, *Generating quasi-random sequences from slightly random sources*, J. Comput. System Sci., 33 (1986), pp. 75–87.

[Sip] M. Sipser, *Expanders, randomness, or time versus space*, J. Comput. System Sci., 36 (1988), pp. 379–383.

[SSZ] M. Saks, A. Srinivasan, and S. Zhou, *Explicit OR-dispersers with polylogarithmic degree*, J. ACM, 45 (1998), pp. 123–154.

[Ta-S] A. Ta-Shma, *On extracting randomness from weak random sources*, in Proc. 28th Annual ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 276–285.

[Va1] U. Vazirani, *Efficiency considerations in using semi-random sources*, in Proc. 19th Annual ACM Symposium on Theory of Computing, ACM, New York, 1987, pp. 160–168.

[Va2] U. Vazirani, *Randomness, Adversaries and Computation*, Ph.D. thesis, University of California, Berkeley, CA, 1986.

[Va3] U. Vazirani, *Strong communication complexity or generating quasi-random sequences from two communicating semi-random sources*, Combinatorica, 7 (1987), pp. 375–392.

[VV] U. Vazirani and V. Vazirani, *Random polynomial time is equal to slightly-random polynomial time*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1985, pp. 417–428. See also U. Vazirani and V. Vazirani, *Random Polynomial Time is Equal to Semi-random Polynomial Time,* Tech. Report 88-959, Department of Computer Science, Cornell University, Ithaca, NY, 1988.

[WZ] A. Wigderson and D. Zuckerman, *Expanders that beat the eigenvalue bound: Explicit construction and applications*, Combinatorica, to appear. Also see Technical Report CS-TR-95-21, Computer Science Dept., The University of Texas at Austin, Austin, TX. Preliminary version appears in Proc. 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 245–251.

[Zu1] D. Zuckerman, *General weak random sources*, in Proc. 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 534–543.

[Zu2] D. Zuckerman, *Simulating BPP using a general weak random source*, Algorithmica, 16 (1996), pp. 367–391.

[Zu3] D. Zuckerman, *On unapproximable versions of NP-complete problems*, SIAM J. Comput., 25 (1996), pp. 1293–1304.

[Zu4] D. Zuckerman, *Randomness-optimal oblivious sampling*, Random Structures Algorithms, 11 (1997), pp. 345–367.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1460–1509

LOWER BOUNDS IN A PARALLEL MODEL WITHOUT BIT OPERATIONS*

KETAN MULMULEY†

Abstract. We define a natural and realistic model of parallel computation called the *PRAM model without bit operations*. It is like the usual PRAM model, the main difference being that no bit operations are provided. It encompasses virtually all known parallel algorithms for (weighted) combinatorial optimization and algebraic problems. In this model we prove that for some large enough constant b, the mincost-flow problem for graphs with n vertices cannot be solved deterministically (or with randomization) in $\sqrt{n}/b$ (expected) time using $2^{\sqrt{n}/b}$ processors; this is so even if we restrict every cost and capacity to be an integer (nonnegative if it is a capacity) of bitlength at most an for some large enough constant a. A similar lower bound is also proved for the max-flow problem. It follows that these problems cannot be solved in our model deterministically (or with randomization) in $\Omega(N^c)$ (expected) time with $2^{\Omega(N^c)}$ processors, where c is an appropriate positive constant and N is the total bitlength of the input. Since these problems were known to be P-complete, this provides concrete support for the belief that P-completeness implies high parallel complexity and for the $P \neq NC$ conjecture. Our lower bounds also extend to the *PRAM model with limited bit operations*, which provides instructions for parity and left or right shift by one bit.

Our proof is based on basic algebraic geometry. So we investigate if the algebrogeometric approach could also work for the P versus NC problem. Our results support this possibility, and a close analysis of the limitation of our technique in this context suggests that such a proof of $P \neq NC$ should somehow use geometric invariant theory in a deep way.

Key words. computational complexity, parallel algorithms, lower bounds

AMS subject classifications. 68Q05, 68Q15, 68Q22, 68Q25

PII. S0097539794282930

1. Overview.

1.1. Introduction.

One fundamental question in computer science is whether $P = NC$. It is generally believed that the answer is no. We are specifically interested in weighted combinatorial optimization problems in P, such as mincost-flow and max-flow, which have the so-called strongly polynomial time algorithms [17]—these are fast, i.e., polynomial time, sequential algorithms that do not use bit operations and the number of whose basic arithmetic steps depends polynomially only on the number of input parameters, not on the input bitlength. For these problems it is natural to ask if they also have fast parallel algorithms that do not use bit operations. Of course, if we assume that $P \neq NC$, then the answer for the mincost-flow and max-flow problems is no, whether bit operations are available or not, because they are P-complete [15]. In this paper we prove this unconditionally, i.e., without assuming $P \neq NC$, when bit operations are not available. This provides concrete support for the belief that P-completeness implies high parallel complexity, and for the $P \neq NC$ conjecture itself, by proving its implications in a model that is restricted but realistic.

The model. What do we mean by a parallel algorithm that does not use bit operations? Formally, we mean an algorithm in a *PRAM model without bit operations*

*Received by the editors December 1, 1994; accepted for publication (in revised form) April 15, 1998; published electronically April 27, 1999. Extended abstracts of parts of this paper appeared in the Proceedings of the ACM Symposium on the Theory of Computing, 1994 and 1997.
http://www.siam.org/journals/sicomp/28-4/28293.html

†Department of Computer Science, University of Chicago, Chicago, IL 60637 and Indian Institute of Technology, Bombay, India (mulmuley@cs.uchicago.edu, http://www.cs.uchicago.edu/~mulmuley).
The work in this paper was partially supported by a David and Lucille Packard Foundation fellowship.

(section 2). It is like the usual PRAM model, the main difference being that it does not provide instructions for any bit operations such as $\wedge$, $\vee$, or *extract-bit*. The model provides usual arithmetic $(+, -, \times)$, comparison $(=, \leq, <)$, store, indirect reference, and branch operations. Each memory location can hold an integer; a rational number is represented by a pair of integers—its numerator and denominator—both of which can be accessed by the processors separately. The processors communicate as in the usual PRAM model. The model does not provide instructions for truncation or integer division with rounding. The lower bounds are proved in the conservative unit-cost model in which each operation is assigned unit cost regardless of the bitlengths of the operands. The PRAM model does not satisfactorily address the issue of interprocessor communication, so important in practice, but it is ideal for proving lower bounds since a lower bound in this model automatically applies to any other realistic parallel model of computation.

Although our model does not provide bit operations, the issue of bitlengths is not ignored. This means two things. First, the running time of an algorithm (in the unit-cost model) is allowed to depend on the input bitlength. For example, the running time of Neff's parallel algorithm [39] for computing approximate roots of a polynomial—which lies in our model—depends on the input bitlength and the bit precision level desired in the output. Second, our lower bounds are expressible in terms of the total bitlength of the input and are applicable even when this bitlength is small, i.e., polynomial in the number of input parameters. In other words, we prove existence of hard instances of small bitlength. Such a result allows connections to the P versus NC problem by expressing lower bounds in terms of bitlength of the input. In contrast, the lower bounds in the models such as the algebraic computation tree model [46, 53] are in terms of the number of input parameters and require their bitlengths to be extremely large or unbounded to be applicable.

Unlike some earlier models used for proving lower bounds, such as the constant-depth [12, 19, 52] or monotone circuit model [42, 1, 5], the PRAM model without bit operations is realistic and natural. It encompasses virtually all known parallel algorithms for weighted optimization and algebraic problems. These include fast parallel algorithms for solving linear systems [11]; minimum-weight-spanning trees [32]; shortest paths [32]; global mincuts in weighted, undirected graphs [30, 31]; blocking flows and max-flows [13, 45]; approximate computation of roots of polynomials [3, 39]; sorting [32]; and several problems in computational geometry [43].

If in our model we require that the running time of a parallel algorithm depend only on the number of input parameters but not on their bitlengths, the resulting *arithmetic PRAM model* is much weaker but still very interesting; many of the preceding parallel algorithms [11, 13, 30, 31, 32, 45] actually belong to this model. Our lower bounds in the PRAM model without bit operations automatically yield ones in the arithmetic PRAM model too—just ignore the bitlength restrictions in their statements.

Lower bounds. We show that for some large enough constant b, the mincost-flow problem for networks with n nodes cannot be solved in the PRAM model without bit operations deterministically (or with randomization) in $\sqrt{n}/b$ (expected) time using $2^{\sqrt{n}/b}$ processors; this is so even if we restrict every cost and capacity to be an integer (nonnegative if it is a capacity) of bitlength at most an for some large enough constant a. We prove the same lower bound for the max-flow problem, which holds even if we restrict every edge-capacity to be a nonnegative integer of bitlength at most an^2 for some large enough constant a. The restriction on the bitlengths implies that these

problems cannot be solved in our model deterministically (or with randomization) in $\Omega(N^c)$ (expected) time with $2^{\Omega(N^c)}$ processors, where c is an appropriate positive constant and N is the total bitlength of the input. Our lower bound also applies to the weighted s-t-mincut problem (for directed or undirected graphs) since it is the dual of the max-flow problem.

Our lower bounds hold for both the decision and the additive approximation versions of these problems. In the decision version, one is given an additional integer parameter w, called *threshold*, and the problem is to decide if the optimum exceeds w. (Thus for max-flow, the problem is to decide if the max-flow value exceeds w; for mincost-flow, the problem is to decide if there exists a flow with value v, specified in the input, and cost at most w.) In the additive approximation version, the goal is to compute the optimum within a small, say, less than 1/8, *additive* error.

Our lower bounds also imply some separation results of complexity theoretic nature. Let SP be the class of languages in P that have strongly polynomial time algorithms [17]; for example, the mincost-flow and max-flow problems belong to SP. (By a language we mean in this paper a set of integer sequences rather than boolean strings.) Let $C[t(N), p(N)]$ be the class of languages that can be recognized in the PRAM model without bit operations in $t(N)$ time (in the unit cost model) using $p(N)$ processors, where N denotes the input bitlength. The randomized class $RC[t(N), p(N)]$ is defined similarly. Let $C^i = C[O(\log^i(N)), \text{poly}(N)]$ be the analogue of the class NC^i and $C = \cup_i C^i$ the analogue of the class NC. For detailed definitions of these complexity classes see section 2.3.

In our model an analogue of the $P \not\subseteq NC$ conjecture would be that $SP \not\subseteq C$. This is not just an analogue but also an implication, i.e., $P \not\subseteq NC$ implies that $SP \not\subseteq C$ (Proposition 2.2). Since the mincost-flow and max-flow problems belong to SP, our lower bounds imply unconditionally that $SP \not\subseteq C$; in fact, they imply something much stronger:

$$SP \not\subseteq RC[N^c, 2^{N^c}]$$

for a suitable positive constant c.

We also prove that the hierarchies $\{C^i\}$ and $\{RC^i\}$, which are analogues of the hierarchies $\{NC^i\}$ and $\{RNC^i\}$, do not collapse. This follows from the following stronger statement: For some positive constant c,

$$C^i \not\subseteq RC^{\lfloor i/c \rfloor + 1}$$

for all i.

PRAM with limited bit operations. It appears that the $P \neq NC$ conjecture is hard essentially due to the issue of bits. There are actually two issues: (1) the running time of a parallel algorithm can depend on the input bitlength, and (2) the algorithm can access an arbitrary bit of the operand. The first issue is addressed satisfactorily in our model, the second issue is not. The higher order bits of an operand are easy to access in our model—it is the middle and lower order bits that are hard (though not impossible) to access (section 2.2). We can extend our lower bound to the *PRAM model with limited bit operations*, which provides instructions for parity and left or right shift by one bit; in this model even the lower order bits are easy to access. This is as far as one can go. If one allows shift by an arbitrary number of bits, the model would become as powerful as the usual unrestricted PRAM-model, and that is beyond the scope of our techniques for reasons outlined in section 7. Parity

is a nonalgebraic operation, or, if it is considered algebraic, it has high exponential degree. Therefore it is interesting that our techniques can handle it.

P versus NC. Our lower bounds are proved using basic algebraic geometry (Milnor–Thom result [34], Collins's decomposition [10], and transversality techniques [16]) in conjunction with diophontine techniques over integer lattices. We also need lower bounds for the so-called parametric complexity (section 3) of the mincost-flow and max-flow problems; these were proved earlier in some other contexts combinatorially [38, 14, 57, 7] (see also Theorem 3.8). Our proof should be contrasted with the combinatorial proofs of the lower bounds for, say, constant-depth [12, 19, 52] or monotone circuits [42, 1, 5, 56], and also with the proofs of the lower bounds in the algebraic computation tree model [2, 46, 53]—these are for sequential sorting-related problems, and the issue of bitlengths is not addressed.

In view of the effectiveness of the algebrogeometric approach in a restricted yet realistic PRAM model of computation in this paper, we investigate its potential in the unrestricted PRAM model. Toward this end, we formulate a certain conjecture (section 7), which if true would imply that $P \neq NC$. Of course, a proof of $P \neq NC$ may not go via this conjecture; it has been formulated mainly to bring out what is lacking in our current approach and what more is needed. Roughly, the conjecture says that for every positive constant a, linear pullbacks of the well-known algebraic group variety $SL_m(C)$, where m satisfies some dimension constraints, cannot have a certain *separation property* that depends on the constant a. We then prove the conjecture for every large enough constant a. Our technique is insufficient for proving the conjecture in full generality, i.e., for every positive constant a—as expected, since it relies on the restricted nature of the PRAM model without bit operations. But a careful analysis of its limitation reveals what a proof of $P \neq NC$ ought to exploit: it should somehow use properties of varieties admitting good group actions—as investigated in geometric invariant theory [6, 37]—in a deep way. (In the terminology of [44], these techniques would not be "natural," but calling them so would not do justice to their depth and beauty. The framework of [44] is not meaningful in the PRAM model without bit operations in this paper just as it is not in the monotone circuit model.[1])

1.2. The issue of bitlengths. Our lower bounds for mincost-flow and max-flow depend on the fact that the edge-capacities can have polynomial—i.e., $\Omega(n)$ or $\Omega(n^2)$—bitlengths. This is necessary: if the bitlengths were small, these problems can, in fact, be solved fast in parallel in our model. For example, consider the max-flow problem. Suppose all edge-capacities have $O(\log n)$ bitlengths. Then in our model all these bits can be extracted in parallel in logarithmic time using the available instructions (section 2.2). Once all bits in the input are extracted, the model becomes as powerful as the usual unrestricted PRAM model. One can thereafter use an RNC-algorithm for unweighted matching [24, 36] to get a polylogarithmic time algorithm. More generally, if all edge-capacities have polylog(n) bitlengths, the problem can be solved in our model in polylog(n) time using $2^{\text{polylog}(n)}$ processors; by a variation of our lower bound, this is close to optimal. The problem can also be solved in $O(n^2 \log n)$ time with $O(n)$ processors without any restriction on the bitlengths by a parallel algorithm of Shiloach and Vishkin [45] or of Goldberg and Tarjan [13], both

[1]This is because a lower bound in our model is meaningful only for a language that can be recognized by an efficient sequential algorithm that does not use bit operations (section 2.2); for a language in P, this means it should have a strongly polynomial time algorithm. But there is no natural way to define a random language of this kind as would be required in [44] for its largeness criterion to make sense.

of which lie in our model. Our lower bound should be contrasted with these upper bounds.

Recall that a strongly polynomial time sequential algorithm [17] is a polynomial time algorithm with poly(n) basic arithmetic steps, where n is the number of input parameters not the input bitlength. Similarly, let us define a *strongly polylogarithmic* time parallel algorithm to be an algorithm in the PRAM model without bit operations which runs in polylog(n) time. For example, the algorithms of Karger [30] and Karger and Motwani [31] for the global mincut problem for weighted undirected graphs are strongly polylogarithmic. Our lower bound implies that the mincost-flow and max-flow problems, which have strongly polynomial time sequential algorithms, have no analogous strongly polylogarithmic time parallel algorithms; but actually it even rules out in our model algorithms that work in $\Omega(N^c)$ (expected) time with $2^{\Omega(N^c)}$ processors, where c is a small enough positive constant and N is the total bitlength of the input. In other words, for each of these problems it rules out a fast parallel algorithm that has the main characteristic of the ellipsoid algorithm: namely, even if it does not use any bit operations, its running time can depend on the input bitlength—this may happen if the algorithm computes the optimum value or some other numerical quantities with high precision. One example of a parallel algorithm in our model that has this characteristic is Neff's algorithm [39] for computing approximate roots of polynomials.

Our lower bound also rules out in our model a fast parallel algorithm for mincost-flow or max-flow that computes the optimum value approximately with high precision—this is what the interior-point algorithm [22] does—and then rounds it up to get the exact value. This is because our lower bound also applies to the approximate computation of the optimum value within a small, say, less than 1/8, *additive* error. Hence, even though rounding is not available in our model, the final rounding can be ignored: all that matters is that the answer just before this rounding is within a small additive error of the exact answer.

Since truncations are not available in our model, they cannot be used to control the bitsizes as in the ellipsoid or the interior-point algorithm. But our lower bounds are proved in the unit-cost model, so the algorithm need not worry about intermediate bitsizes. The ellipsoid and the interior-point algorithms also perform some basic algebraic operations such as solving linear systems and computing approximate roots of polynomials—these can be simulated fast in parallel in our model using the algorithms in [11] and [3, 39], respectively.

1.3. Statement of the results. Now we shall formally state our main results.

THEOREM 1.1. *The mincost-flow problem for networks with k nodes cannot be solved in the PRAM model without bit operations deterministically (or with randomization) in $\sqrt{k}/b$ (expected) time using $2^{\sqrt{k}/b}$ parallel processors, even assuming that every cost and capacity is an integer with bitlength at most ak for some large enough positive constants a and b.*

Similarly, the max-flow problem for directed or undirected networks with k nodes cannot be solved in the PRAM model without bit operations deterministically (or with randomization) in $\sqrt{k}/b$ (expected) time using $2^{\sqrt{k}/b}$ parallel processors, even assuming that every capacity is a nonnegative integer of bitlength at most ak^2 for some large enough positive constants a and b. The same lower bound also applies to the dual s-t-mincut problem for weighted graphs.

All lower bounds apply for the decision as well as the additive approximation versions of the problems. The lower bounds also apply in the PRAM model with

limited bit operations.

The total bitsize N of a network with k-nodes and $O(k)$ or $O(k^2)$-bit edge-capacities and costs is polynomial in k. This allows us to express the preceding lower bounds in terms of the input bitlength N.

COROLLARY 1.2. *The mincost-flow or the max-flow problem cannot be solved in the PRAM model without bit operations (or with limited bit operations) deterministically (or with randomization) in $\Omega(N^c)$ (expected) time using $2^{\Omega(N^c)}$ processors for some positive constant c.*

Remark. The theorem implies $c = 1/6$ for mincost-flow and $c = 1/8$ for max-flow; for mincost-flow the value of c can be reduced to $1/4$ (see section 3.1.1).

Since the mincost-flow problem belongs to the class SP, we also conclude the following.

COROLLARY 1.3.

$$SP \not\subseteq RC[N^{1/4}/b, 2^{N^{1/4}/b}]$$

for a large enough constant b.

We shall also prove that the $\{C^i\}$ and $\{RC^i\}$ hierarchies do not collapse; these are analogues of the conjectures that the $\{NC^i\}$ and $\{RNC^i\}$ hierarchies do not collapse.

THEOREM 1.4. *$C^i \not\subseteq RC^{i/c}$ for some positive constant c.*

Corollary 1.5 follows.

COROLLARY 1.5. *The $\{C^i\}$ and $\{RC^i\}$ hierarchies do not collapse.*

For the P versus NC problem and related results, see section 7.

1.4. Basic idea of the proof. We shall actually prove a general result (Theorem 3.3), which says that if the so-called parametric complexity of the problem is high, then it is hard to parallelize in our model. This implies our lower bounds for the mincost-flow and max-flow problems because their parametric complexity is exponential, as was shown by Carstensen [7, 8] and Zadeh [57]. Our proof has two steps. In the first step we show that if the problem with high parametric complexity had a fast parallel algorithm in our model, then there would exist a certain low-degree algebraic decomposition of a small subset of the three-dimensional integer lattice. In the second step—the heart of the proof—we show that such algebraic decomposition cannot exist. We shall now elaborate this a bit more.

Let n denote the number of integer parameters in the input and N the total bitlength of the input. Let Z denote the set of integers. Suppose, to the contrary, that there is a (nonuniform) machine M in the PRAM model without bit operations that efficiently solves a given combinatorial optimization problem with high parametric complexity for all inputs with number of parameters n and bitlength at most N. Let $L \subseteq Z^n$ denote the language corresponding to the problem; i.e., L is the set of all points in Z^n for which the answer is yes. For reasons that will become clear soon, we shall parametrize input to this machine using a constant d number of integer parameters. The parameterization is given by an integral linear map $I : Z^d \to Z^n$. Thus any integer point $z \in Z^d$ corresponds to the input $I(z)$ to the machine. For example, if L is the max-flow language, one can choose a suitable flow network whose edge-capacities are integral linear forms in two integer parameters z_1 and z_2, and let z_3 denote the threshold. This specifies a linear map from $z = (z_1, z_2, z_3)$ to the input $I(z)$ consisting of a flow network and a threshold.

Since the machine works correctly only for inputs of bitlength at most N, we shall only be interested in those $z \in Z^d$ such that the bitlength of $I(z)$ is at most N—such z will be called *permissible*. Let us color a permissible point $z \in Z^d$ green if $I(Z)$

belongs to the language L; otherwise color it red. The points that are not permissible are colorless. The first step in our proof (Theorems 4.6 and 5.6) is to show that if M recognizes L efficiently, then the green and the red points in Z^d can be *separated* by a set of algebraic (hyper)surfaces of "small" total degree. Here the total degree depends exponentially on d; so we need d to be as small as possible—in fact, we shall be able to choose $d = 3$ in our parameterization. This first step uses the Milnor–Thom result [34] from algebraic geometry, which has also been used earlier [2, 46] in the algebraic computation tree model.

When the parametric complexity of the problem is high, one can choose a parameterization I so that the green and the red points are distributed badly in some sense. The heart of our proof (Theorem 5.9) lies in showing that when the permissible points are badly colored, the red and the green points cannot be separated by any set of algebraic surfaces of small degree. The algebraic surfaces that arise in our proof can be highly singular and they can intersect badly. Therefore, without disturbing the separation property, first we smooth the surfaces and ensure that their intersections and "silhouettes" are also smooth using transversality techniques from differential geometry [16]. Next we refine the partition formed by these surfaces further by Collins' decomposition method [10] so that all regions in the resulting partition are "cylinders" of simple shape. Then using diophontine techniques, especially the pigeonhole principle, we show that if this partition were to separate the green and the red points, then some region in it must be too warped. However, since all surfaces have small degree, this cannot happen, a contradiction.

It turns out that the parametric complexity of a combinatorial optimization problem is essentially the maximum number of pivot steps the simplex algorithm would need using the so-called Gass–Saaty pivot rule. It was studied in this and related contexts in operations research. Klee and Minty [26] had shown in their classic paper that the simplex algorithm can take exponentially many steps for some simple pivot rules. The same was shown for the Gass–Saaty pivot rule by Goldfarb [14] and Murty [38] for general linear programming, by Zadeh [57] for mincost-flow, and by Carstensen [7, 8] for max-flow. This provides exponential lower bounds for the parametric complexity of these problems. Lower bounds on their parallel complexity in our model then follow from our general lower bound (Theorem 3.3) mentioned above.

1.5. Organization of the paper. In section 2 we describe the PRAM model without bit operations formally. In section 3 we explain the connection between parametric and parallel complexity. In section 4 we first prove our general lower bound in a restricted linear PRAM model without bit operations; this model does not provide general multiplication. The proof here is elementary and shows the basic ideas in a very simple setting. Moreover, we even get a stronger lower bound that is sensitive to processor-time trade-off. In section 5 the proof is extended to the general PRAM model without bit operations. In section 6 we extend the lower bound to randomized algorithms and to the PRAM model with limited bit operations. In section 7 we analyze the limitations of our approach in the context of the general P versus NC problem and indicate what more may be needed.

2. The model. Here we shall formally define the PRAM model without bit operations. This model is like the usual PRAM model [23], the essential difference being that the instruction set of the processors does not contain any bit operations.

Input to the machine is supposed to be a sequence of integers, rather than boolean numbers. In other words, each integer in the input is used as a specification for itself;

each rational number is specified in terms of its numerator and denominator; and the remaining information, such as graph specification, is encoded as an integer sequence of zeroes and ones. The input is supposed to be divided into two parts: nonnumeric data and numeric data. For the max-flow problem, the nonnumeric data would specify the underlying graph and the numeric data would specify the nonnegative edge-capacities. Let n denote the cardinality of the input, i.e., the total number of its integer parameters and N its total bitlength.

2.1. PRAM without bit operations. First, we shall define the model in the nonuniform setting. A nonuniform machine for cardinality n and bitsize N is supposed to work correctly on all inputs of cardinality n and bitlength at most N. Our machine will consist of $p(n, N)$ processors ordered in some fashion, where $p(n, N)$ is some function of n and N. The processors have private memories, and they communicate through a shared global memory. EREW, CREW, and CRCW modes of communication are defined as usual. Each memory location can contain an integer of any size that can arise in the course of execution. Accordingly, instructions can be assigned costs that depend on the bitlengths of the operands. This is important when one proves an upper bound on the running time of an algorithm. However, our lower bounds also work in a stronger sense when one assigns unit cost to each instruction regardless of the bitsizes of the operands; since the actual cost of any instruction is $\Omega(1)$, such lower bounds hold in any cost model. For the purpose of lower bounds, we shall assume this unit-cost model in the paper.

In our machine, a rational number is represented as a pair of its denominator and numerator, which need not be relatively prime, and each of which can be accessed separately by a processor. Rational operations, including division, can then be simulated by integer operations. It is important that the machine can access the numerator and the denominator separately; if we were to treat a rational number as an abstract data type (i.e., a black box) so that only its value is visible to the outside world, then the model would be far weaker. Initially, the shared memory will contain n, the cardinality of the input, in its first location, followed by the nonnumeric data, followed by the numeric data. The remaining memory locations are initialized to zero. At the end of execution, the first location in the shared memory should contain the output value.

In the nonuniform setting, it is best to think that the program of each processor has been unfolded in the form of a computation tree, quite like in the algebraic computation tree model [2, 46]. This means the length of a program can depend on n and N. Each node in this tree has a label and an instruction associated with it. No instructions for bit manipulations are provided. The instructions are of the following kinds:

1. $w = u \circ v$, where w denotes a memory location, u and v denote either memory locations or constants, which in the nonuniform setting can depend on n and N. Finally, $\circ$ denotes a binary operation $+$, $-$, or $\times$.
2. **go to** l, where l is a label of some instruction.
3. **if** $u : 0$ **go to** l, where u denotes a memory location and $:$ denotes either $<$, $\leq$, or $=$.
4. $u := v$: Store contents of the memory location v into the location u.
5. $u :=\uparrow v$ (indirect reference): Treat the value in the memory location v as a pointer. Store into the memory location u the value in the pointed memory location. We assume that the value in v is a valid pointer in a sense specified below.

6. **stop**.

We assume that the pointer involved in an indirect reference is not some numeric argument in the input or a quantity that depends on it. For example, in the max-flow problem the algorithm should not use an edge-capacity as a pointer—which is a reasonable condition. To enforce this restriction, one initially puts an *invalid-pointer* tag on every numeric argument in the input. During the execution of an arithmetic instruction, the same tag is also propagated to the result if any operand has that tag. Trying to use a memory value with invalid-pointer tag results in error.

To allow randomization, we provide an additional instruction: *random-branch* l. When it is encountered, the processor flips a fair coin. If the toss is head, it branches to the instruction labeled l; otherwise, it proceeds to the next instruction. We allow two-sided errors in randomized algorithms.

In this model the higher order bits of an operand can be easily accessed using available instructions, but the lower order bits are hard to access (section 2.2): more precisely, the lth bit from the left can be extracted in $O(l+\log x)$ operations, where x is the value of the operand. Our model does not provide integer division with rounding, because if it did, an arbitrary bit in the input could be extracted fast and the model would become as powerful as the usual PRAM model. Nevertheless, small roundings in which the rounded quotient is guaranteed to have small, say, polylogarithmic, bitsize can be simulated efficiently using available operations. Since general truncations are not provided in our model, one cannot use them to control bitsizes as in the ellipsoid or the interior-point algorithm. This does not matter because our lower bounds are proved in the unit-cost model, and hence the algorithm need not worry about intermediate bitsizes.

The model just described was nonuniform. In its uniform version we assume that there is a Turing machine that, given n and N, generates programs for all $p(n, N)$ processors using space that is logarithmic in their total size. In the uniform version we also assume that the size of each processor's program is constant; in other words, it is not meant to be unfolded in the form of a computation tree. Since we have provided instructions for jump and indirect reference, unfolding is not necessary.

2.1.1. Arithmetic PRAM model. The arithmetic PRAM model is obtained when one restricts the PRAM model without bit operations by requiring the running time $t(n, N)$ and the number of processors $p(n, N)$ to depend only on n but not on the bitlength N. In this model the issue of bitlengths is ignored just as in Yao's algebraic computation tree model for integer inputs [53]. Nevertheless it is interesting because many parallel algorithms that have been designed for (weighted) combinatorial optimization or algebraic problems lie in this model. If one is interested only in showing that a given problem does not have a strongly polylogarithmic time parallel algorithm, as defined in section 1.2, then this is the model to be used. All our lower bounds automatically hold in this weaker model, just ignore the bitsize restrictions in their statements.

2.1.2. Linear PRAM model. The linear PRAM model results if we restrict the PRAM model without bit operations by requiring that at least one operand of the $\times$ instruction be a constant. If in addition we require that the running time $t(n, N)$ and the number of processors $p(n, N)$ depend only on n but not on the bitlength N, then the resulting model is called *the arithmetic linear PRAM model*.

2.1.3. Examples. We now list various versions of our models in the increasing order of power. For each model we give examples of algorithms which lie in that

model but not in the weaker model preceding it.

Arithmetic linear PRAM model. Contains parallel algorithms for many weighted combinatorial optimization problems, such as shortest paths [32], minimum-weight spanning trees [32], max-flows and blocking flows [45, 13], global mincuts in undirected weighted graphs [31], and many problems in computational geometry [43].

Arithmetic PRAM model. Also contains parallel algorithms for solving linear systems over rationals and other problems in linear algebra, due to Csanky [11] and others (e.g., [21]), and for evaluating arithmetic circuits (straight line codes) over rationals [55]. In Csanky's algorithm [11], as it is, the bitsizes of intermediate operands can become superpolynomial in the bitlength of the input. This is fine since we are using the unit-cost model. (Of course, one can keep the bitsizes under control, but this requires integer division with rounding, an operation not available in our model.)

PRAM model without bit operations. Also contains Neff's algorithm for computing approximate roots of polynomials [39], the earlier algorithm of Ben-Or et al. [3] for a restricted class of polynomials, the numerical algorithm of Pan and Reif [40] for solving linear systems iteratively, and numerical parallel algorithms for several other problems (see the book [4] for a survey). These algorithms use truncations, which are not available in our model, to keep intermediate bitsizes under control, but, again, this is not necessary in our unit-cost model, as we already pointed out.

We do not know a natural example that is in the linear PRAM model but not in the arithmetic linear PRAM model—in this paper the linear PRAM model is used more as a medium for showing basic ideas of the proof in an elementary setting.

As one nontrivial example in weighted combinatorial optimization, let us explain why Karger and Motwani's NC-algorithm [31] for computing global mincuts in undirected weighted graphs falls in our model; in fact, it lies in the arithmetic PRAM model. It begins by normalizing weights in the graph so that they have polynomial magnitudes, i.e., logarithmic bitsizes. This requires rounding a ratio of two integers to the nearest integer, where the ratio is guaranteed to have logarithmic bitsize. Such small roundings can be carried out efficiently in our model in logarithmic time using the available operations. After this, Karger and Motwani give an NC-algorithm to compute all approximate global mincuts in the resulting normalized graph. But since all bits of the edge-weights in the normalized graph are now known to our algorithm, it can simulate any algorithm in the unrestricted PRAM model hereafter. Thus the subsequent NC-computation on the normalized graph can be simulated in our model without any problem. Once all approximate global mincuts in the normalized graph are computed—which, by Karger's result [30], are polynomial in number—Karger and Motwani's algorithm computes the weights of these cuts in the original graph and selects one with minimum weight. This too poses no problem in our model.

2.1.4. Algebraic operations. Since the parallel algorithms in [3, 39] for computing of approximate roots of polynomials lie in the PRAM model without bit operations, this operation can be simulated efficiently. Thus all our lower bounds also hold with appropriate modifications to take into account the cost of simulation, even if we were to allow this approximate algebraic operation in the model. In the sequential setting it plays a crucial role in the ellipsoid and interior-point algorithms.

2.1.5. PRAM with limited bit operations. The PRAM with limited bit operations is obtained by extending the PRAM model without bit operations by adding to the instruction set operations for left or right shift by one bit and parity, or equivalently an assignment operation $u := \lfloor u/2 \rfloor$, or, more generally, an assignment

$u := \lfloor u/p \rfloor$ for any constant p. If we were to allow shift by arbitrary number of bits, the model would become as strong as the unrestricted PRAM model.

2.2. Bit extraction. In the arithmetic PRAM model the running time of an algorithm is required to depend only on the cardinality n but not on the total bitlength N, so extracting arbitrary bit of the operand is, in general, impossible. The situation here is like in the monotone circuit model [42], where the *not* operation is impossible, or like in Yao's algebraic computation tree model for integer inputs [53], where bit extraction is impossible.

In the PRAM model without bit operations, bits can be extracted using the available instructions but this can be hard, although not impossible. More precisely, all bits of an operand x with bitlength l can be extracted in $O(l)$ time sequentially as follows. First estimate the bitlength l in $O(\log l)$ operations, starting with 2 and squaring repeatedly until the value of x is exceeded. After this the bits of x can be extracted one by one, starting at the most significant bit, using available arithmetic and comparison operations. More generally, all bits can be extracted in $O(\sqrt{l})$ time using $2^{\sqrt{l}}$ processors as follows. Conceptually divide x into $\sqrt{l}$ blocks of equal length, and extract the blocks one by one, starting with the most significant block. For example, to extract the most significant block, one can guess in parallel all of its possible values. Whether a guessed value is the correct one can be verified easily using the available operations.

However, if l is large, bit extraction is hard. In fact, it is provably so (Proposition 2.1). Of course, a lower bound for bit extraction in our model does not have much meaning as a lower bound statement; saying that bit extraction is hard in the PRAM model without bit operations is analogous to saying that the *not* operation is impossible in the monotone circuit model or that bit extraction is impossible in the algebraic computation tree model or in the arithmetic PRAM model. Just as a lower bound in the monotone circuit model is interesting only if the problem is monotone, a lower bound in the PRAM model without bit operations is interesting only if the problem has an efficient sequential algorithm that does not use bit operations: for a problem in P, this means it should have a strongly polynomial time algorithm [17] (see also section 2.3 for more discussion of this issue). This is true of all combinatorial optimization problems we consider. One trivial example of a problem that does not have a strongly polynomial time algorithm is bit extraction itself (because such an algorithm for bit extraction must work within $O(1)$ arithmetic and comparison operations, which is impossible).

PROPOSITION 2.1. *The lowest order bit of an n-bit operand cannot be extracted in the PRAM model without bit operations in $\sqrt{n}/a$ time using $2^{\sqrt{n}/a}$ processors for some large enough constant a.*

Proof. Suppose, to the contrary, that there is a parallel machine M in the model that outputs the value of the lowermost bit in $t(n) = \sqrt{n}/a$ time using $p(n) = 2^{t(n)} = 2^{\sqrt{n}/a}$ processors; here a is a large enough constant to be chosen later. Given a nonnegative integer t, call two inputs x and $\bar{x}$ t-equivalent if the behavior of the machine M up to time t on the input x is the same as that on the input $\bar{x}$; more formally, the branches taken by all processors in M up to time t are the same for both x and $\bar{x}$. We say that x and $\bar{x}$ are equivalent if they are $t(n)$-equivalent.

Now assume that the input x to the machine is guaranteed to be in a fixed t-equivalence class σ. This fixes all branches taken by every processor in M up to time t; then each memory location is a certain fixed polynomial in the input x, and the degree of this polynomial is at most 2^t. Now consider all comparisons, at most $p(n)$ in

number, that M would execute at time $t+1$. Since the value of each memory location just before time $t+1$ is a predetermined polynomial in x, once we are told that $x \in \sigma$, each such comparison amounts to evaluating the sign of some polynomial, which is $+$, $-$, or 0. Let $f_1(x), \ldots f_p(x)$, where $p \leq p(n)$, be these polynomials. Each $f_i(x)$ has degree at most 2^t and hence at most 2^t real roots. Consider the set of all roots of all polynomials $f_i(x)$, $i \leq p$. They divide the real line into at most $2^t p(n) + 1$ intervals. If $x_1, x_2 \in \sigma$ lie in the same open interval, or if they are equal and coincide with some root, then the signs of $f_i(x_1)$ and $f_i(x_2)$ coincide for each i, and consequently all processors in M would take the same branching path at time $t + 1$. This means that any t-equivalence class σ is split into at most

$$2 \cdot 2^t p(n) + 1 \leq 2 \cdot 4^{t(n)} + 1$$

$(t + 1)$-equivalence classes. By induction, it follows that the total number of equivalence classes, i.e., $t(n)$-equivalence classes, is at most $(2 \cdot 4^{t(n)} + 1)^{t(n)} < 5^{t(n)^2}$.

Now consider a fixed equivalence class ζ. Once ζ is fixed, the branches taken by all processors during the entire execution of M are fixed and the comparison carried out at every such branch corresponds to evaluating the sign of a fixed polynomial in x of degree at most $2^{t(n)}$. Let $P(\zeta)$ denote the set of all these polynomials that correspond to all branches of all processors during the entire execution. The size of $P(\zeta)$ is at most $t(n)p(n)$. Let P be the union of all $P(\zeta)$, as ζ ranges over all equivalence classes, at most $5^{t(n)^2}$ in number. The size of P is at most $t(n)p(n)5^{t(n)^2}$, and each polynomial in P has degree at most $2^{t(n)}$. Consider the set R of real roots of all polynomials in P; its size is at most $t(n)p(n)5^{t(n)^2}2^{t(n)}$. These roots divide the real line into at most

$$t(n)p(n)5^{t(n)^2}2^{t(n)} + 1 < 6^{t(n)^2} = 6^{n/a^2}$$

intervals. If two inputs x_1 and x_2 lie in the same open interval, then the behavior of all processors in M on the input x_1 is the same as that on x_2; thus the output would be the same, too—in other words, the lowermost bits of x_1 and x_2 must be the same. However, we have already seen that the number of intervals is at most $6^{n/a^2}$. If a is large enough, this number is much less than $2^n/2$. But then, by the pigeonhole principle, at least one such open interval would contain two integers x_1 and x_2 with the same lowermost bit, a contradiction. □

More generally, the same method also proves that extracting the ith bit from the left is hard for every "large" i. In the PRAM model with limited bit operations, one can prove similarly that the middle bits are hard to extract.

The essential weakness of a machine in our model lies in accessing bits. But if the language is boolean, i.e., if all its input parameters are either 0 or 1, then all bits of the input are available to our machine right at the outset. In this case, our model is as powerful as the unrestricted PRAM model. Thus proving a lower bound for a boolean language in our model is equivalent to proving it in the unrestricted PRAM model. On the other hand, using our model for proving a lower bound for a boolean language would not make much sense.

2.3. Complexity classes. We shall now formally define, both in the sequential and the parallel setting, complexity classes of problems having efficient algorithms that do not use bit operations; however, their running times may or may not depend on the input bitlengths and we need to distinguish between these two cases carefully.

First consider the sequential setting. If we allow the running time to depend on the bitlength, then we need to be very careful. One cannot simply define a sequential

algorithm that does not use bit operations to be an algorithm in the restricted RAM model without bit operations. This is because all bits in the input of bitlength N can be extracted in this model sequentially with $O(n)$ arithmetic and comparison operations. After this, any algorithm in the unrestricted RAM model can be simulated in the restricted RAM model without bit operations. This means a polynomial time algorithm in the restricted RAM model without bit operations is as powerful as a polynomial time algorithm in the unrestricted RAM model. Thus it is somewhat tricky to define a fast sequential algorithm whose running time can depend on the input bitlength but which does not use bit operations in a "real sense." Intuitively, such an algorithm should be like the ellipsoid or the interior-point algorithm for linear programming [22, 25] (the truncations that occur in these algorithms for controlling bitlengths are not bit operations in a real sense). This can be made precise and formal, but we shall not do so here.

If we do not allow the running time to depend on the input bitlength, the matter becomes much simpler. An efficient sequential algorithm that does not use bit operations and whose running time (in the unit-cost model) also does not depend on the input bitlength is nothing but a *strongly polynomial time* algorithm [17]. By this we mean that (1) the running time of the algorithm in the unit-cost model is polynomial in the number of integer parameters in the input, and (2) the bitsize of any intermediate operand that arises in the course of execution is bounded by a polynomial in the total bitlength of the input.

Let $SP \subseteq P$ (strongly polynomial) be the class of languages in P which can be recognized by strongly polynomial time algorithms. (Recall that a language in this paper means a set of integer sequences rather than boolean strings.) The class SP is quite rich. It contains decision problems associated with solving linear systems, network flow problems such as min-cost flow and max-flow, minimum-weight matching, *combinatorial* linear programming [47], ϵ-approximate knapsack problem [20], 3/2-approximate traveling salesman problem [9], and large-enough-constant-factor approximation of weighted MAX-SNP problems [41].

Next, let us turn to the parallel setting. Here an analogue of the class SP is the class SNC (strong-NC). It is the class of problems that have *strongly polylogarithmic time algorithms*, i.e., the algorithms in the PRAM model without bit operations that work in polylog(n) time using poly(n) processors, where n denotes the number of integer parameters in the input. Since we are following the unit-cost model, we are not requiring the bitlengths of intermediate operands to remain polynomial in the input bitlength. Therefore strictly speaking SNC need not be a subclass of NC; whether this is indeed so is open. The class SNC is also quite rich. Some problems in this class are solving linear systems over rationals [11], minimum-weight-spanning trees [32], shortest paths [32], global mincuts in undirected graphs [31], and several problems in computational geometry such as constructing convex hulls [43].

Now let us allow the running time of a parallel algorithm to depend on the input bitlength; however, it should not use any bit operations. Formally, let C be the class of problems that can be solved in the PRAM model without bit operations in polylog(N) time using poly(N) processors, where N denotes the input bitlength. Again, there is no restriction on the bitlengths of intermediate operands, so strictly speaking C need not be a subclass of NC or even P; the formal inclusion question remains open. An important problem that is in the class C but not in SNC is that of computing approximate roots of real polynomials up to a specified bit-precision level [3, 39]. Neff's algorithm [39] uses truncations, which are not available in our model, to

keep bitsizes of intermediate operands under control, but they can be ignored. Then bitsizes of intermediate operands can become superpolynomial, which is fine since we are following the unit-cost model. This also explains why we did not impose any restriction on the bitsizes of intermediate operands.

More generally, let $C[t(n,N),p(n,N)]$ be the class of languages that can be recognized in the PRAM model without bit operations in $t(n,N)$ time in the unit-cost model using $p(n,N)$ processors, where n is the number of parameters in the input and N is the total bitlength of the input. Let $C^i = C[O(\log^i(N)),\mathrm{poly}(N)]$ be the analogue of the class NC^i. One can define the randomized class RC, or more generally, $RC[t(n,N),p(n,N)]$ similarly. The class RC is the analogue of the class RNC.

In our model, an analogue of the $P \not\subseteq NC$ conjecture would be that $SP \not\subseteq SNC$ or, more strongly, that $SP \not\subseteq C$. This is not just an analogue but also an implication.

PROPOSITION 2.2. *$P \not\subseteq NC$ implies $SP \not\subseteq C$.*

Proof. Suppose $P \not\subseteq NC$. Consider the restricted linear programming problem in which all integer parameters in the input have $O(1)$ bitsizes. It belongs to SP [47]. It is also known to be P-complete, so it cannot belong to NC. If it belonged to C it would also belong to NC, because when all integer parameters in the input have $O(1)$ bitsizes the PRAM model without bit operations is as powerful as the unrestricted PRAM model (section 2.2). □

Our results in this paper imply, unconditionally, that $SP \not\subseteq C$ and much more (Corollary 1.3). They also imply that the hierarchies $\{C^i\}$ and $\{RC^i\}$ do not collapse.

3. Parametric versus parallel complexity. All our lower bounds follow from a general lower bound (Theorem 3.3) on the parallel complexity of a general optimization problem in our model in terms of its parametric complexity. In this section we explore the relationship between parametric and parallel complexity in detail. In section 3.1 we state our general lower bound and derive from it our main results stated in section 1.3. In section 3.2 we give a simpler proof of Carstensen's exponential lower bound [8] for the parametric complexity of the max-flow and the dual weighted s-t-mincut problem. In section 3.3 we prove a polynomial upper bound on the parametric complexity of the weighted global mincut problem for undirected graphs. This should explain why the weighted s-t-mincut problem for undirected graphs has no fast parallel algorithm in our model, whereas the global mincut problem has, e.g., the one in [30] or [31].

3.1. A general lower bound. First, let us define parametric complexity. For the max-flow problem it is defined as follows. Consider a graph with n vertices whose edge-capacities are linear functions of a parameter λ with rational coefficients of bitsize at most $\beta(n)$ for some function $\beta(n)$. We also require that all capacities remain nonnegative as λ varies over reals in some interval $[p,q]$ of definition. We use real capacities only for the sake of defining parametric complexity; the actual capacities in the input will always be nonnegative integers. Let $f(\lambda)$ be the max-flow value for a particular $\lambda \in [p,q]$. The function $f(\lambda)$ is piecewise linear. Let $\rho(f)$ be the number of breakpoints, i.e., slope changes in the function graph of f in the interval $[p,q]$. Parametric complexity $\phi(n,\beta(n))$ of the max-flow problem for cardinality n and bitsize $\beta(n)$ is defined to be the maximum possible value of $\rho(f)$ over all valid edge-capacity parameterizations, as above, of graphs with n vertices.

One can similarly define parametric complexity of an arbitrary homogeneous optimization problem. Consider a general combinatorial optimization problem, where the input consists of two parts: nonnumeric data and numeric data. Given an input I, the problem is to compute the optimum value of some quantity; we shall

denote this optimum value for I by $F(I)$. For example, in the max-flow problem the nonnumeric data would specify the network and the numeric data would specify the edge-capacities. Given an input I, $F(I)$ would be the max-flow value. In the decision version of the optimization problem one is given an additional integer parameter w, called *threshold*, and the problem is to decide if $w \leq F(I)$.

We will mainly be interested in the integral version of the optimization problem, wherein numeric parameters are integers. However, for the sake of defining parametric complexity we assume that the quantity $F(I)$ is well defined even when the numeric parameters are reals, possibly with some constraints; for example, we can require them to be nonnegative. We also assume that the problem is homogeneous; this means if we multiply all numeric parameters in I by a positive number α, then the optimum value $F(I)$ also gets multiplied by α. For example, the max-flow problem is well defined even when all capacities are nonnegative reals, and it is homogeneous, too.

DEFINITION 3.1. *By a linear parameterization for cardinality n, we mean a map $\mathcal{P}$ that associates with each real number λ in some interval $[p, q]$ an input $I(\lambda)$ such that the following hold:*

1. *The nonnumeric part of the input $I(\lambda)$ remains the same for every $\lambda \in [p, q]$.*
2. *The number of numeric parameters in $I(\lambda)$ is n for every $\lambda \in [p, q]$.*
3. *Each numeric parameter is a linear function of λ with rational coefficients and it satisfies any required constraints in the problem (such as nonnegativity) for every $\lambda \in [p, q]$.*
4. *The function graph of $F(\lambda) = F(I(\lambda))$ as λ varies over $[p, q]$ is piecewise linear and convex; we shall denote it by $G(\mathcal{P})$. (The function graph is convex for a maximization problem. For a minimization problem it would be concave.)*

The interval $[p, q]$ is called the interval of definition. The bitsize of $\mathcal{P}$ is defined to be the maximum among the bitsizes of the coordinates of the vertices of $G(\mathcal{P})$ and of the coefficients of the linear functions in $\mathcal{P}$ specifying numeric parameters. These coordinates and coefficients are in general rational. We define the bitsize of a rational to be the sum of the bitsizes of its denominator and numerator. We shall denote the bitsize of $\mathcal{P}$ by $\beta(\mathcal{P})$. The number of bounded linear segments in $G(\mathcal{P})$ is called the *complexity* of $\mathcal{P}$; it will be denoted by $\rho(\mathcal{P})$.

For example, in the max-flow problem, we can fix a graph with n edges and let the capacities of its edges be linear functions of λ so that as λ varies over some interval $[p, q]$ all these capacities remain nonnegative; this specifies a linear parameterization. It is well known that the optimum function of such parameterization is piecewise linear and convex.

DEFINITION 3.2. *Fix a homogeneous optimization problem. Its parametric complexity for cardinality n is defined to the maximum of $\rho(\mathcal{P})$ as $\mathcal{P}$ ranges over all linear parameterizations for cardinality n; we denote it by $\phi(n)$.*

More generally, parametric complexity for cardinality n and bitsize $\beta(n)$ is defined to the maximum of $\rho(\mathcal{P})$ as $\mathcal{P}$ ranges over all linear parameterizations for cardinality n with bitsize $\beta(n)$; we denote it by $\phi(n, \beta(n))$.

So far, n has denoted the number of numeric parameters in the input. We shall abuse the notation on some occasions and let n stand for something different but similar. For example, in the max-flow problem we shall let n denote the number of nodes in the flow network. The precise meaning of n in the given context should be clear.

We shall show that if the parametric complexity of a problem is high, then it is hard to parallelize in our model.

THEOREM 3.3. *Fix a homogeneous optimization problem. Let $\phi(n, \beta(n))$ be its parametric complexity for input cardinality n and bitsize $\beta(n)$. Then there exists a large enough constant b such that the decision version of the problem cannot be solved in the PRAM model without bit operations in $\sqrt{\log[\phi(n,\beta(n))]}/b$ time using $2^{\sqrt{\log[\phi(n,\beta(n))]}/b}$ processors; this is so even if we restrict every numeric parameter in the input to be an integer with bitlength at most $a\beta(n)$ for a large enough constant a.*

We shall prove this result in sections 4 and 5. It will be extended to the randomized setting in section 6.

COROLLARY 3.4. *The lower bound in Theorem* 3.3 *also applies to the additive approximation version of the problem in which the goal is to compute the optimum within a small, say, less than* 1/8, additive *error. (The constant* 1/8 *here is replaceable by any $\epsilon < 1/2$.)*

Proof. Suppose there is a fast parallel algorithm in our model for the additive approximation version. This will yield a fast parallel algorithm for the decision version as follows. First we compute the approximate optimum v with additive error less than, say, 1/8. Since we are assuming that the input is integral, we know that the exact optimum is integral. Thus we compare v with the integral threshold w in the input; w is less than or equal to the exact optimum iff $w \leq v + 1/8$. □

THEOREM 3.5.

1. *For combinatorial linear programming, $\phi(n, O(n)) = 2^{\Omega(n)}$, where n denotes the total number of variables and constraints* (see *Murty* [38]).
2. *For the mincost-flow problem, $\phi(n, O(n)) = 2^{\Omega(n)}$, where n denotes the number of nodes in the flow network* (see *Zadeh* [57]).
3. *For the max-flow problem for directed or undirected networks, $\phi(n, O(n^2)) = 2^{\Omega(n)}$, where n denotes the number of nodes in the flow network* (see *Carstensen* [8] *and Theorem* 3.8).

By combinatorial linear programming we mean restricted linear programming problem of the form "maximize $\{cx \mid Ax \leq b\}$," where all entries of A have logarithmic bitlengths; it has a strongly polynomial time algorithm [47]. The result of Murty [38] is subsumed by the later results [57, 7] since the max-flow and mincost-flow are instances of linear programming; however, Murty's proof is much simpler than these later ones.

3.1.1. Applications. We shall now derive the main results stated in section 1.3 from Theorem 3.3.

In the deterministic setting, our lower bounds for the mincost-flow and max-flow (Theorem 1.1) follow directly from Theorem 3.3, Corollary 3.4, and Theorem 3.5. For randomized algorithms and the PRAM model with limited bit operations, see section 6. (For the mincost-flow problem the improvement in the constant c mentioned after the statement of Corollary 1.2 follows from the fact that the total bitsize of Zadeh's [57] parametrized flow network with k nodes is only $O(k^2)$, even though some of its edges may have $O(k)$-bit costs.)

We now proceed to prove that $C^i \not\subseteq RC^{i/c}$ for some positive constant c (Theorem 1.4). Fix a positive integer j. From the usual max-flow problem, we shall first construct a *padded* language L that belongs to the class C. A valid input instance of L for cardinality n has the following form: It contains a flow network G on nodes $1, \ldots, \lfloor \log^j n \rfloor$ with edge-capacities of bitlength at most $a \log^{2j} n$ for a suitable constant a to be specified later. The remaining nodes are dummy and isolated. The input also specifies the threshold w.

The input is accepted iff the max-flow value for G is less than or equal to w. Let us show that $L \in C^{bj}$ for a suitable constant b. It is easy to check fast in parallel if

the input is valid, i.e., if the nodes $\lfloor \log^j n \rfloor + 1, \ldots, n$ are indeed dummy and isolated, and if all edge-capacities have bitlengths at most $a \log^{2j} n$. If the input is not valid it is rejected right away. Otherwise dummy nodes are discarded. After this, one is left with a small network G with $\lfloor \log^j n \rfloor$ nodes. The max-flow problem for G can be solved by just one processor using, say, a version of Dinic's strongly polynomial time algorithm [48] in $O(\log^{3j} n)$ time. If one uses the parallel algorithm of Shiloach and Vishkin [45] or Goldberg and Tarjan [13], it can be solved in $O(\log^{2j} n \log\log n)$ time using $O(\log^j n)$ processors. The total bitlength N of a valid input instance in L is $O(n + \log^{4j} n)$. Hence $L \in C^{bj}$ for a suitable constant b.

Now we shall get a lower bound on the parametric complexity of the padded problem. For this we use Carstensen's [8] or our (section 3.2) parametric construction with padding. This means we form a padded graph with n nodes in which all but $k = \log^j n$ nodes are dummy. The edge-capacities of the network G on the remaining k nodes are then linear functions of the parameter λ with coefficients of $O(k)$ bitsize as in [8] or section 3.2. It follows from Theorem 3.5 (third statement) that the parametric complexity $\phi(n, O(k^2))$ of the padded max-flow problem is $2^{\Omega(k)}$, where $k = \log^j n$. Therefore, by Theorem 3.3, it cannot be solved in the PRAM model without bit operations in $\log^{j/2} n/a'$ time using $2^{\log^{j/2} n/a'}$ processors for a large enough constant a', even assuming that every edge-capacity in the input has at most $a \log^{2j} n$ bitlength for a suitable large enough constant a. Thus L does not belong to $RC^{j/2-1}$. But we have already seen that $L \in C^{bj}$ for a suitable positive constant b. Thus it follows that $C^i \not\subseteq RC^{i/c}$ for a suitable positive constant c.

The role of the max-flow problem in this proof can be played by any problem in SP whose parametric complexity has a good exponential lower bound. For example, we could as well have used the mincost-flow problem or the general combinatorial linear programming problem. Then we have to use Zadeh's [57] or Murty's [38] parametric construction.

3.2. s-t-mincuts in weighted undirected graphs. Carstensen [8] proved that the parametric complexity of the s-t-mincut problem for weighted undirected graphs with n vertices is $2^{\Omega(n)}$. The same lower bound also applies to the dual max-flow problem for undirected graphs. Carstensen's proof is very intricate, however. In this section we give a much simpler proof using similar construction.

Let G be a weighted undirected graph with two distinguished vertices s and t. Each edge has a nonnegative weight. By an s-t-cut (U, V), we shall mean a disjoint partition (U, V) of the vertex set of G such that U contains s and V contains t. The weight $w(U, V)$ of this cut is the total weight of all edges going from U to V. By an *s-t-*mincut in G, we mean an s-t-cut (U, V) with minimum weight. We say that an s-t-cut $J = (U, V)$ contains a vertex u if $u \in U$.

For each positive integer n, we shall construct a weighted undirected graph G_n with $2n + 2$ vertices whose weights are linear functions of a parameter λ such that the following hold:

1. In the interval $(-T, T)$, where $T = 2^{n+1}$, all edge-weights are nonnegative.
2. Let $g(\lambda)$ denote the weight of a mincut in G for a particular λ. Then the function graph of $g(\lambda)$ has at least $2^n - 1$ breakpoints, i.e., slope changes in the interval $(-T, T)$.
3. The coefficients of the vertices of this function graph are rationals with $O(n^2)$ bitlengths.

The graph G_n is defined as follows. The vertices of G_n consist of s, t, the vertices labeled $1, \ldots, n$, and the vertices labeled $\bar{1}, \ldots, \bar{n}$. There are edges connecting s to

every vertex i and $\bar{i}$ and edges connecting every vertex i and $\bar{i}$ to t. For all $i \neq j$, there are also edges connecting the vertices i and $\bar{i}$ to the vertices j and $\bar{j}$. Let $w(u,v) = w(v,u)$ denote the weight of the edge between u and v. Let w_i for $1 \leq i \leq n$ be positive integers such that $w_n = 1$ and for $i < n$

$$w_i > 2T \sum_{i<l\leq n} w_l. \tag{3.1}$$

The choice $w_i = (2nT)^{n-i}$ for $i \leq n$ would suffice. Let $t_j = 2^{n-j+1}$. The edge-weights are then defined as follows:

1. For all i, $w(s,i) = w(s,\bar{i}) = w_i T$.
2. For all i, $w(i,t) = w_i(T+\lambda)$ and $w(\bar{i},t) = w_i(T-\lambda)$
3. For all $i > j$, $w(i,j) = w(\bar{i},\bar{j}) = w_i(T-t_j)$ and $w(i,\bar{j}) = w(\bar{i},j) = w_i T$.

Note that all edge-weights are positive in the interval $(-T,T)$ of interest to us.

THEOREM 3.6. *The graph of the mincut cost function $g(\lambda)$ for G_n has at least $2^n - 1$ breakpoints in the interval $(-T,T)$.*

Before we prove the theorem, we shall prove a lemma. Let part(0) denote the interval $(-T,T)$. For $i \geq 1$, define part(i) to be the partition obtained by splitting each interval of part($i-1$) at its midpoint. Thus part(i) contains 2^i intervals of length 2^{n+2-i}. Consider a complete binary tree of depth n whose nodes at depth i from the root are labeled with the intervals of part(i). Any sequence σ of $+1$'s and -1's with length $i \leq n$ denotes a unique node of this tree. Let $J(\sigma)$ denote the corresponding interval; it is the unique interval of part(i) containing the point $p(\sigma)$ with coefficient $t(\sigma) = \sum_{j\leq i} \sigma(j) t_j$, where $\sigma(j)$ denotes the jth symbol of σ. By convention $t(\sigma) = 0$ and $J(\sigma)$ denotes the interval $(-T,T)$ if σ is empty. Let $I(\sigma)$ be the interval of positive length $2^{n+2-i} - 2$ obtained by removing from both ends of $J(\sigma)$ intervals of length 1.

LEMMA 3.7. *Fix any $i \leq n$ and a sequence σ of length i. Fix any $\lambda \in I(\sigma)$. Let $C(\lambda) = (U(\lambda), V(\lambda))$ be a mincut in G_n at this λ; it need not be unique. Then, for every $j \leq i$,*

1. *$U(\lambda)$ contains j iff $\sigma(j) = -1$, and*
2. *$U(\lambda)$ contains $\bar{j}$ iff $\sigma(j) = +1$.*

Proof. Notice that the weighted graph G_n remains invariant if we switch all pairs $(i,\bar{i})$ of vertices and replace λ by $-\lambda$. Hence, the first statement is equivalent to the second so we shall prove only the former. The proof proceeds by induction on i.

Basis case, $i = 1$. Consider the case when $\sigma(1) = +1$. Suppose, to the contrary, that $U(\lambda)$ contains the vertex 1. If we move 1 from $U(\lambda)$ to $V(\lambda)$, the value of the cut would decrease by

$$\begin{aligned}
&w(1,t) + \textstyle\sum_{j>1:j\in V(\lambda)} w(j,1) + \sum_{j>1:\bar{j}\in V(\lambda)} w(\bar{j},1) \\
&\quad - w(s,1) - \textstyle\sum_{j>1:j\in U(\lambda)} w(j,1) - \sum_{j>1:\bar{j}\in U(\lambda)} w(\bar{j},1) \\
&= w_1(T+\lambda) + \textstyle\sum_{j>1:j\in V(\lambda)} w_j(T-t_1) + \sum_{j>1:\bar{j}\in V(\lambda)} w_j T \\
&\quad - w_1 T - \textstyle\sum_{j>1:j\in U(\lambda)} w_j(T-t_1) - \sum_{j>1:\bar{j}\in U(\lambda)} w_j T \\
&= w_1(\lambda) + h(\lambda,1),
\end{aligned}$$

where $h(\lambda, 1)$ is the lower order term

$$\sum_{j>1:j\in V(\lambda)} w_j(T-t_1) + \sum_{j>1:\bar{j}\in V(\lambda)} w_j T - \sum_{j>1:j\in U(\lambda)} w_j(T-t_1) - \sum_{j>1:\bar{j}\in U(\lambda)} w_j T;$$

it is bounded in absolute value by $2T\sum_{j>1} w_j$, which by (3.1) is strictly less than w_1. It follows that in the interval $I(\sigma) = [1, T-1]$ the decrease in the weight of the mincut is positive. This contradicts the fact that J is a mincut. Thus $U(\lambda)$ cannot contain 1. It follows similarly that if $\sigma(1) = -1$ and $V(\lambda)$ contained the vertex 1, then moving it from $V(\lambda)$ to $U(\lambda)$ would decrease the weight of the cut by

$$-w_1\lambda - h(\lambda, 1),$$

which is positive if $\lambda \in I(\sigma) = (-T, -1)$. Thus $V(\lambda)$ cannot contain 1 if $\sigma(1) = -1$. It follows that $U(\lambda)$ contains 1 iff $\sigma(1) = -1$.

Now let us turn to the general case. In general, let $\hat{\sigma}$ be the string obtained from σ by removing its last, i.e., the ith, symbol. Clearly, $I(\sigma) \subset I(\hat{\sigma})$. Hence, by the inductive hypothesis applied to $\hat{\sigma}$, $U(\lambda)$ at any $\lambda \in I(\sigma)$ must contain for every $j < i$ the vertex j iff $\sigma(j) = -1$ and the vertex $\bar{j}$ iff $\sigma(j) = 1$. It remains to show that $U(\lambda)$ contains i iff $\sigma(i) = -1$.

First, consider the case when $\sigma(i) = 1$. Suppose, to the contrary, that $U(\lambda)$ contains i. Then moving i from $U(\lambda)$ to $V(\lambda)$ would decrease the weight of the cut by

$$\begin{aligned} &w(i,t) - w(s,i) + \textstyle\sum_{j\neq i:j,\bar{j}\in V(\lambda)} w(i,j) + w(i,\bar{j}) \\ &\quad - \textstyle\sum_{j\neq i:j,\bar{j}\in U(\lambda)} [w(j,i) - w(\bar{j},i)] \\ &= w_i\lambda + \textstyle\sum_{j<i:j,\bar{j}\in V(\lambda)} w(i,j) + w(i,\bar{j}) \\ &\quad - \textstyle\sum_{j<i:j,\bar{j}\in U(\lambda)} [w(i,j) - w(i,\bar{j})] + h(\lambda, i), \end{aligned} \tag{3.2}$$

where $h(\lambda, i)$ is the lower order term

$$\sum_{j>i:j,\bar{j}\in V(\lambda)} w(j,i) + w(\bar{j},i) - \sum_{j>i:j,\bar{j}\in U(\lambda)} [w(j,i) - w(\bar{j},i)];$$

it is bounded in absolute value by $2T\sum_{j>i} w_j$, which by (3.1) is strictly less than w_i. By the inductive hypothesis, the decrease given by (3.2) is equal to

$$\begin{aligned} &w_i\lambda + \textstyle\sum_{j<i:\sigma(j)=1} w(i,j) + \sum_{j<i:\sigma(j)=-1} w(i,\bar{j}) \\ &\quad - \textstyle\sum_{j<i:\sigma(j)=-1} w(i,j) - \sum_{j<i:\sigma(j)=1} w(i,\bar{j}) + h(\lambda, i) \\ &= w_i\lambda + \textstyle\sum_{j<i:\sigma(j)=1} w_i(T-t_j) + \sum_{j<i:\sigma(j)=-1} w_i T \\ &\quad - \textstyle\sum_{j<i:\sigma(j)=-1} w_i(T-t_j) - \sum_{j<i:\sigma(j)=1} w_i T + h(\lambda, i) \\ &= w_i\lambda - \textstyle\sum_{j<i:\sigma(j)=1} w_i t_j + \sum_{j<i:\sigma(j)=-1} w_i t_j + h(\lambda, i) \\ &= w_i\lambda - w_i(\textstyle\sum_{j<i} \sigma(j) t_j) + h(\lambda, i) \\ &= w_i(\lambda - t(\hat{\sigma})) + h(\lambda, i), \end{aligned}$$

where $t(\hat{\sigma}) = \sum_{j<i} \hat{\sigma}(j)t_j = \sum_{j<i} \sigma(j)t_j$. If $\lambda \in I(\sigma)$ and $\sigma(i) = 1$, then $\lambda - t(\hat{\sigma}) \geq 1$. Since the absolute value of $h(\lambda, i)$ is less than w_i, this decrease is positive in that case. Hence moving i from $U(\lambda)$ to $V(\lambda)$ would strictly decrease the weight of the cut, which contradicts the fact the $C(\lambda)$ is a mincut. Thus $U(\lambda)$ cannot contain i if $\sigma(i) = 1$.

Now consider the case when $\sigma(i) = -1$. Suppose to the contrary that $V(\lambda)$ contains i. Then it follows similarly that moving i from $V(\lambda)$ to $U(\lambda)$ would decrease the weight of the cut by

$$w_i(t(\hat{\sigma}) - \lambda) - h(\lambda, i).$$

If $\lambda \in I(\sigma)$ and $\sigma(i) = -1$, then $t(\hat{\sigma}) - \lambda \geq 1$. Hence this decrease is strictly positive, which contradicts the fact that $C(\lambda)$ is a mincut. It follows that $V(\lambda)$ cannot contain i if $\sigma(i) = -1$.

Thus $U(\lambda)$ contains i iff $\sigma(i) = -1$. □

If σ is any sequence of 1 and -1 of length n, then it follows from the preceding lemma that throughout the interval $I(\sigma)$ the mincut remains the same; let us call it $C(\sigma) = (U(\sigma), V(\sigma))$. It also follows that $U(\sigma)$ contains i iff $\sigma(i) = -1$ and $\bar{i}$ iff $\sigma(i) = 1$. The weight of $C(\sigma)$ at any $\lambda \in I(\sigma)$ is of the form $A(\sigma)\lambda + B(\lambda)$, where $A(\sigma) = -\sum \sigma(i)w_i$. Thus the slopes of the weight functions of all 2^n cuts $C(\sigma)$ as σ ranges over all strings of 1 and -1 of length n are distinct. This proves Theorem 3.6. □

Bitlengths of the edge-capacities in our construction remain $O(n^2)$ in the interval $[-T, T]$ of interest. Hence Theorem 3.8 follows.

THEOREM 3.8. *Parametric complexity $\phi(n, O(n^2))$ of the s-t-mincut problem for weighted undirected graphs of cardinality n is $2^{\Omega(n)}$. This also holds for the dual max-flow problem.*

3.3. Global mincuts in weighted undirected graphs. In this section we show that the parametric complexity of the global mincut problem for weighted undirected graphs is at most polynomial in the number of vertices; this should explain why this problem has fast parallel algorithms in our model [30, 31]. By a global mincut we mean a nontrivial cut; there is no restriction that some distinguished vertices s and t be separated by the cut.

We shall make use of the following combinatorial result and its proof in [30].

LEMMA 3.9. *In any positively weighted undirected graph, the number of cuts within a constant multiplicative factor of a global mincut is polynomial in the number of vertices.*

We shall also use a result by Chandrasekharan and Gusfield [18] that the parametric complexity of the maximum-weight spanning tree problem is $O(n^4)$; this is in fact true for any matroidal optimization problem having a greedy algorithm.

Let G be an undirected graph with n vertices whose edge-weights are linear functions of a parameter λ. It is furthermore guaranteed that in some interval $[p, q]$ all edge-weights remain positive. Let $g(\lambda)$ denote the cost of a mincut in G at a given λ.

THEOREM 3.10. *The number of breakpoints in the function graph of $g(\lambda)$ in the interval $[p, q]$ is at most polynomial in n.*

Proof. The coefficients of the linear weight functions of the edges in G may be arbitrary reals. But one can show that without of loss of generality these coefficients can be assumed to be rationals with bitsizes at most polynomial in n. This follows from the fact that the parametric complexity of the mincut problem is the same as the maximum size of a two-dimensional projection of the corresponding combinatorial

polytope, which is the convex hull of the incidence vectors of all cuts in G. Since the coefficients of the linear weight functions have polynomial size bitlengths, all vertices of the function graph of $g(\lambda)$ have rational coefficients with polynomial bitlengths. For this reason one can also assume that p and q have polynomial bitlengths.

The parametric complexity of the maximum-weight spanning tree problem is $O(n^4)$ [18]. Thus by dividing $[p, q]$ into at most $O(n^4)$ subintervals if necessary, one can assume that a maximum-weight spanning tree in G remains the same throughout $[p, q]$, i.e., as λ varies in the interval $[p, q]$. This tree has n edges since G can be assumed to be connected. Thus subdividing $[p, q]$ further into at most n subintervals, one can also assume without loss of generality that the edge of minimum weight in this maximum-weight spanning tree remains the same throughout $[p, q]$. Let this edge be e and let its weight function be $w_e(\lambda)$. By subdividing the interval $[p, q]$ further into at most n subintervals if necessary, one can also assume that the set of edges whose weights exceed n^2 times the weight of e also remains the same throughout $[p, q]$. As in Karger [30], we can contract these edges in G without changing the mincut. Let us assume that all edges in G have weights at most n^2 times that of e throughout $[p, q]$. This means the weight of any cut at any $\lambda \in [p, q]$ is at most n^4 times the weight $w_e(\lambda)$ of e throughout $[p, q]$. It also follows, as in [30], that the weight of any cut at any $\lambda \in [p, q]$ is at least $w_e(\lambda)$.

By our assumption, both $w_e(p)$ and $w_e(q)$ must be positive. Assume without loss of generality that $w_e(p) < w_e(q)$, the other case being similar. Since the coefficients of the linear weight functions, and also p and q, have polynomial size bitlengths, $\log(w_e(q)) - \log(w_e(p))$ is polynomial in n. Thus by subdividing $[p, q]$ further into polynomially many subintervals, if necessary we can assume that $\log(w_e(q)) - \log(w_e(p)) \leq 1/n$, which means that $w_e(q) \leq 2w_e(p)$.

For any cut J, let $w_J(\lambda)$ denote its weight function. Then our interval $[p, q]$ has the following properties. First,

$$w_e(p) \leq w_e(q) \leq 2w_e(p). \tag{3.3}$$

Second, for any cut J and any $\lambda \in [p, q]$,

$$w_e(\lambda) \leq w_J(\lambda) \leq n^4 w_e(\lambda). \tag{3.4}$$

Divide $[p, q]$ further into, say, n^5 subintervals of equal length. Let $[a, b]$ denote any such subinterval.

Claim. For any $\lambda \in [a, b]$ and any cut J, $|w_J(\lambda) - w_J(b)| \leq 2w_e(b)/n$.

Proof of the claim. By (3.4), $w_J(q) \leq n^4 w_e(q)$, and similarly $w_J(p) \leq n^4 w_e(p) \leq n^4 w_e(q)$ by (3.4) and (3.3). Hence, the slope of the weight function $w_J(\lambda)$ is at most $n^4 w_e(q)/(q - p)$. Therefore, for any $\lambda \in [a, b]$, $|w_J(\lambda) - w_J(b)|$ is at most

$$\frac{n^4 w_e(q)(b - \lambda)}{(q - p)} \leq \frac{w_e(q) n^4}{n^5} = \frac{w_e(q)}{n} \leq \frac{2w_e(p)}{n} \leq \frac{2w_e(b)}{n}.$$

This proves the claim.

Let K denote a mincut at b.

Claim. A cut I whose weight becomes minimum at some $\lambda \in [a, b]$ has its weight at b at most twice that of K at b.

Proof of the claim. First, note that $w_I(b) - w_K(b)$ is equal to

$$(w_I(b) - w_I(\lambda)) + (w_I(\lambda) - w_K(\lambda)) + (w_K(\lambda) - w_K(b)).$$

Since K is a mincut at b, $w_I(b) - w_K(b)$ is nonnegative, and similarly $w_K(\lambda) - w_I(\lambda)$ is nonnegative. Therefore, $w_I(b) - w_K(b)$ is at most

$$(w_I(b) - w_I(\lambda)) + (w_K(\lambda) - w_K(b)) \leq 4w_e(b)/n$$

by the previous claim. But $w_K(b)$ is at least $w_e(b)$ by (3.4). Thus, the last quantity is at most $4w_K(b)/n$. Now the claim follows.

By Lemma 3.9 there are only polynomially many cuts whose weights at b are at most four times the weight of the mincut K at b. Let Γ_b denote the set of such cuts. Now the preceding claim implies that any cut that can become minimum during $[a, b]$ must belong to the set Γ_b whose size $|\Gamma_b|$ is polynomial in n. In other words, in the interval $[a, b]$ the mincut weight function is given by

$$g(\lambda) = \min\{w_K(\lambda) \mid K \in \Gamma_b\}$$

and so can have at most $|\Gamma_b|$ breakpoints within the interval $[a, b]$. This is true for each of the polynomially many subintervals into which $[p, q]$ was divided. Thus the theorem follows. □

4. Linear PRAM. In this section we prove our general lower bound (Theorem 3.3) for the linear PRAM model. Actually we get a stronger lower bound that takes into account processor-time trade-off.

THEOREM 4.1. *Fix any homogeneous optimization problem. Let $\phi(n, \beta(n))$ be its parametric complexity for input cardinality n and bitsize $\beta(n)$. Then there exists a large enough constant b such that the decision version of the problem cannot be solved in the linear PRAM model in $\sqrt{\log[\phi(n, \beta(n))]}/b$ time using $2^{\sqrt{\log[\phi(n,\beta(n))]}/b}$ processors or, more generally, in $\log[\phi(n, \beta(n))]/(b \log p)$ time using p processors; this is so even if we restrict every numeric parameter in the input to be an integer with bitlength at most $a\beta(n)$ for a large enough constant a.*

This also holds for approximate computation of the optimum within a small, say, less than $1/8$, additive error.

In conjunction with Theorem 3.5 this leads to a somewhat stronger lower bound for the linear PRAM than in Theorem 1.1.

THEOREM 4.2. *The mincost-flow problem for networks with k nodes cannot be solved in the linear PRAM model deterministically (or with randomization) in $k/(b \log p)$ (expected) time using p processors, even assuming that every cost and capacity is an integer with bitlength at most ak, for some large enough positive constants a and b; in particular, it cannot be solved in $o(k/\log k)$ time using polynomial number of processors. The same lower bound also applies for the max-flow problem with the bitlength restriction ak^2 instead of ak.*

Compare with the upper bound provided by the parallel algorithms of Shiloach and Vishkin [45] and Goldberg and Tarjan [13] (which lie in the linear PRAM model); these work in $O(k^2 \log k)$ time with $O(k)$ processors without any restriction on the bitlengths.

Here we shall consider only deterministic algorithms. For randomized algorithms, see section 6.1.

4.1. The number of possible branchings. In this section we describe a general technique for bounding the number of possible ways in which the processors can branch in the linear PRAM model. The technique is language independent. Thus we shall describe it in a general setting.

Let $\Sigma = \Sigma(x_1, \ldots, x_n)$, $x_i \in Z$, be the language for which we want to prove a lower bound. Here $x_1, \ldots, x_n$ denote the integer parameters in the input. We shall denote the total bitsize of the input by N. We shall denote the integer tuple $(x_1, \ldots, x_n)$ by x. Consider any nonuniform machine for n and N. It is required to work correctly only on inputs with n parameters of total bitlength at most N. Hence, we shall confine ourselves to only those values of x_i for which the total bitsize of x is bounded by N. Let $p(n, N)$ be the number of processors in the machine. Let $t(n, N)$ be the maximum time taken by the machine on any input of bitlength at most N. Let $x, x' \in Z^n$ be any two n-tuples of bitsize at most N. For any positive integer t, we say that x and x' are *t-equivalent* if on the input x the instruction executed by any processor in our machine at any time less than or equal to t is the same as the instruction that would have been executed if the input were x' instead. We say that x and x' are *equivalent* if they are t-equivalent for all $t \leq t(n, N)$. By a t-equivalence class, we mean the set of all t-equivalent k-tuples of bitsize at most N. An equivalence class is defined similarly. Let $\phi(t)$ be the number of t-equivalence classes and ϕ be the number of equivalence classes.

Ideally, we would like to get good bounds on $\phi(t)$ and ϕ. Unfortunately, this seems difficult. Therefore, we shall get around this problem by specializing the possible inputs to our machine; this corresponds to restricting one's attention to inputs that lie within some affine subspace. We shall then obtain a good bound on the number of equivalence classes among specialized inputs.

Specifically, let us imagine running our machine on the set of possible inputs defined by $x_i = l_i(z) = l_i(z_1, \ldots, z_d)$, where for each $i \leq n$, l_i is a certain integral linear form (possibly nonhomogeneous) in d variables and z_j ranges over all integer values of bitlength bounded by some integer r_j. When x_i is a nonnumeric parameter, we shall assume that $l_i(z)$ is constant. In other words, we assume that the nonnumeric parameters do not change as we change the parameters z_i. The forms l_i will depend on the problem under consideration. We shall turn to this issue in section 4.2. Here we need only to assume that the integers r_j and the forms l_i are such that the total bitsize of the tuple $x = (x_1, \ldots, x_n)$ remains bounded by N as we let z_i range over all permissible values. This implies that the machine must work correctly on this parametrized set of inputs within a given time bound $t(n, N)$. In what follows, we shall assume that every d-tuple $z = (z_1, \ldots, z_d)$ under consideration is *permissible*. This means each z_i is an integer of bitlength at most r_i. In our applications d will be constant.

Let $\Sigma(z_1, \ldots, z_d)$ denote the set of permissible d-tuples accepted by the machine. We say that z is accepted if $x = l(z) = (\ldots, l_i(z), \ldots)$ is accepted. Two permissible d-tuples z, z' are said to be t-equivalent if $l(z)$ and $l(z')$ are t-equivalent. The notion of equivalence is defined similarly. Let $\sigma(t)$ denote the number of t-equivalence classes among the permissible values of z. Let $\sigma = \sigma(t(n, N))$ denote the number of equivalence classes.

THEOREM 4.3. *For all t, $\sigma(t)$ is bounded by $[2p(n, N)]^{dt}$.*

Proof. Let C denote a fixed t-equivalence class. Let z denote a (permissible) integral value in this equivalence class. In what follows, we pretend that z is a generic variable. We are told that it belongs to C. But we are not told its actual value. We imagine running our machine on the input $x = l(z)$. We have to specify what is meant by executing an arithmetic operation in this setting. Naturally, we cannot expect the operands to be integers, unless they happen to be constants. Rather they will be functions of z. Since there is no general multiplication in the linear PRAM model,

these functions will be linear. By an arithmetic operation + or −, we simply mean the corresponding operation on the linear forms (functions). We are also allowed multiplication by constants. This amounts to multiplying the corresponding linear form by a constant. The nonnumeric parameters are fixed in our parameterization. Since in our model the memory pointers are allowed to depend only on the nonnumeric parameters, indirect memory references are no problem. In other words, whenever an instruction involving a memory pointer is encountered during the course of execution, the value of the pointer would be completely determined at that time, because it cannot depend on the generic parameter z. This in turn completely determines which memory locations are to be fetched (or stored into) during the execution of that instruction. In case several processors are planning to write into the same memory location, it is also determined which processor's writing is going to prevail (in the CRCW mode of communication). We assume that when a processor is executing a comparison it is simply told the result of that comparison, which, at any time less than or equal t, is completely determined by the class C; i.e., it is the same for all permissible values in C.

LEMMA 4.4. *Let C be a fixed t-equivalence class. For every memory location j, there exists a linear form $g_j[C,t](z)$ that gives the content of that location at time t for every $z \in C$.*

Proof. Once C is fixed, the execution path of each processor up to time t is fixed. Thus the lemma follows easily by induction on t. □

Let us now see what happens at time t given that the generic variable z belongs to a fixed t-equivalence class C as above. If a processor is to execute an arithmetic operation at time t, then the result of that operation will be a certain uniquely determined linear form to be stored in a certain uniquely determined location in the memory. If the operation is a branch, the branching path will be determined by a comparison between the contents of two uniquely determined memory locations, which are certain fixed linear forms in z. In other words, each branch is determined by testing a certain linear equality or inequality in z of the form $g(z) : 0$, where the comparison : is $>$, $\geq$, or $=$. The number of such linear constraints is at most $p(n,N)$, one per processor. Let $g_1(z), g_2(z), \ldots$ be the nonzero linear forms involved in these constraints. Consider the arrangement in R^d formed by the hyperplanes $g_i(z) = 0$. The hyperplanes naturally partition R^d into a disjoint union of (open) j-faces, where each j-face of the arrangement is a j-dimensional convex polytope. When d is fixed, the total number of faces in the arrangement (of all dimensions) is $O(m^d)$, or at most $(2m)^d$ to be more precise, where m is the number of hyperplanes, i.e., the number of linear forms g_i. There are at most $p(n,N)$ forms, one per processor. Hence the number of faces in the arrangement is at most $(2p(n,N))^d$.

Now we make the following crucial observation: Any two $z, z' \in C$ belong to the same $(t+1)$-equivalence class if they belong to the same face of this partition. This is because the sign of each linear function $g_i(z)$ is the same at any point of the face (the sign is either +, −, or 0). It immediately follows that

$$\sigma(t+1) \leq (2p(n,N))^d \sigma(t).$$

Induction on t proves Theorem 4.3.

The proof of the theorem also yields the following lemma.

LEMMA 4.5. *For every t-equivalence class C, there exists a set $\Phi(C,t)$ of at most $p(n,N)t$ linear constraints (equalities or inequalities) in d variables such that $z \in C$ iff all these constraints are satisfied.*

Proof. The constraints correspond to the branches before time t, at most t per processor, which are all fixed by the class C. □

By Lemma 4.5, each equivalence class D of permissible z-values is characterized by a set $\Phi(D) = \Phi(D, t(n, N))$ of linear constraints. Let $\Phi = \cup_D \Phi(D)$, where D ranges over all equivalence classes. By Theorem 4.3 and Lemma 4.5, it follows that Φ contains at most $[2p(n, N)]^{dt(n,N)} p(n, N)t(n, N)$ linear forms. Consider the arrangement in R^d formed by the hyperplanes defined by these linear forms. It is easy to see that each face of this arrangement can be labeled *yes* or *no* in such a way that a permissible z is accepted by the machine iff it lies in a face labeled *yes*. Thus we have proved the following.

THEOREM 4.6. *Assume that the language $\Sigma(z_1, \ldots, z_d)$ is accepted in the linear PRAM model using $p(n, N)$ processors in $t(n, N)$ time. Then R^d can be partitioned by a set of at most*

$$[2p(n, N)]^{dt(n,N)} p(n, N)t(n, N)$$

hyperplanes and each face of the arrangement can be labeled yes *or* no *so that a permissible $z \in Z^d$ belongs to $\Sigma(z_1, \ldots, z_d)$ iff it lies in a face labeled* yes.

4.2. Parameterization. The preceding theorem suggests the following strategy for reducing a global lower bound problem to a local problem in small dimensions. Parametrize the input to the machine using a small (constant) number of parameters. Then for this parameterization prove that a partition having the properties mentioned in Theorem 4.6 cannot exist. As a byproduct, we shall actually end up proving a stronger lower bound (Theorem 4.7) that is nonuniform with respect to parameterization. This means for any parameterization $\mathcal{P}$ we get a lower bound that applies even to nonuniform machines whose programs can depend on $\mathcal{P}$ arbitrarily. Theorem 4.1 would be a consequence of this stronger lower bound.

Fix a general optimization problem as in section 3.1 and any parameterization $\mathcal{P}$ for cardinality n and bitsize $\beta(n)$. Let the complexity of $\mathcal{P}$ be $\rho(n)$. For a given rational parameter λ, each numeric parameter in the input $\mathcal{P}(\lambda)$ is a linear function of the form $u\lambda + v$; we can assume that u and v are integers because if they are not, we can clear all denominators by multiplying all numeric parameters by a large enough integer, and since the problem is homogeneous, the optimum value gets multiplied by the same integer. When λ is rational, $u\lambda + v$ is in general rational, too. This is not allowed, since in our optimization problem every numeric parameter has to be integral. However, with every such parameterization $\mathcal{P}$, one can associate a homogeneous integral parameterization $\tilde{\mathcal{P}}$ with two integer parameters z_1 and z_2—which can be thought of as the denominator and numerator of the parameter λ—by replacing each linear form $u\lambda + v$ in the definition of $\mathcal{P}$ by the corresponding integral form $uz_2 + vz_1$. For any integer pair (z_1, z_2), each numeric parameter of the input $\tilde{\mathcal{P}}(z_1, z_2)$ is now an integer as required. The bitsize $\beta(n)$ or the complexity $\rho(n)$ of $\tilde{\mathcal{P}}$ is defined to be the same as that of $\mathcal{P}$.

Remark. Integral nature of our optimization problem—i.e., the assumption that $F(I)$ and each numeric parameter in I is an integer—is important; it is crucial in proving the lower bound for the approximate version (Corollary 3.4) from the one for the decision version (Theorem 3.3). Even if we were to allow rational values of numeric parameters, the situation would not be any easier or different, since the machine in our model can access the numerator and the denominator of a rational number separately.

Now given a large enough positive integer a, called the *permissibility constant*, let $\mathcal{I}$ be the set of inputs of the form (z_3, I) in the decision version of our optimization problem, where the following holds:

1. The integer $w = z_3$ denotes the threshold and its bitsize is at most $a\beta(n)$.
2. I is of the form $\tilde{\mathcal{P}}(z_1, z_2)$ for some integers z_1 and z_2 of bitsize at most $a\beta(n)$.

We shall prove the following theorem.

THEOREM 4.7. *Assume that the permissibility constant a is large enough. Then no machine in the linear PRAM model can decide whether $w \le F(I)$ correctly for every $I \in \mathcal{I}$ within $\sqrt{\log \rho(n)}/a'$ parallel time using $2^{\sqrt{[\log \rho(n)/a'}}$ processors for some large enough constant a' that does not depend on the permissibility constant a.*

More generally, no machine in the linear PRAM model can decide whether $w \le F(I)$ correctly for every $I \in \mathcal{I}$ within $\log \rho(n)/(a' \log p)$ parallel time using p processors for some large enough constant a' that does not depend on the permissibility constant a.

In the deterministic setting, Theorem 4.1 follows from this theorem by applying it to a parameterization $\mathcal{P}$ for cardinality n and bitsize $\beta(n)$ whose complexity is equal to $\phi(n, \beta(n))$, the maximum possible value. The randomized setting is considered in section 6.1. In what follows, we shall denote $\rho(n)$ and $\beta(n)$ by simply ρ and β, respectively.

Let us apply Theorem 4.6 with $d = 3$ to the linear parameterization of the input as in Theorem 4.7. It is defined by the linear map

$$(z_1, z_2, z_3) \to I(z_1, z_2, z_3) = (z_3, \tilde{\mathcal{P}}(z_1, z_2)).$$

Thus in Theorem 4.6 $\Sigma(z_1, z_2, z_3)$ consists of those tuples (z_1, z_2, z_3) such that $z_3 \le \tilde{\mathcal{P}}(z_1, z_2)$. This gives the following theorem.

THEOREM 4.8. *Let a' be any positive constant. If a linear PRAM machine works correctly on the input $I(z_1, z_2, z_3)$ for every permissible z_1, z_2, and z_3 in $t = \sqrt{\log \rho}/a'$ time using 2^t processors, then R^3 can be partitioned by at most 2^{5t^2} planes and each face of the resulting arrangement can be labeled* yes *or* no *so that a permissible integer point (z_1, z_2, z_3) lies in a face labeled* yes *iff $I(z_1, z_2, z_3)$ is feasible. (The constant 5 is just chosen to be large enough.)*

More generally, if a linear PRAM machine works correctly on the input $I(z_1, z_2, z_3)$ for every permissible z_1, z_2, and z_3 in $\log \rho/(a' \log p)$ time using p processors, then R^3 can be partitioned by at most $2^{(5 \log \rho)/a'}$ planes and each face of the resulting arrangement can be labeled yes *or* no *so that a permissible integer point (z_1, z_2, z_3) lies in a face labeled* yes *iff $I(z_1, z_2, z_3)$ is feasible.*

4.3. A lattice problem. We are thus led to the problem of showing that an arrangement as in Theorem 4.8 cannot exist for large enough constants a and a'; this will prove Theorem 4.7. Let $F(\lambda)$ be the optimum function associated with $\mathcal{P}$ (as in section 3.1) and $G = G(\mathcal{P})$ its function graph. Since our parameterization is assumed to be homogeneous, $I(z_1, z_2, z_3)$ is feasible iff $z_3 \le \tilde{\mathcal{P}}(z_1, z_2)$, i.e., $z_3/z_1 \le F(z_2/z_1)$. (Here and in what follows, we assume that a permissible z_1 is always positive.) This suggests that we think of G as lying in the affine plane $z_1 = 1$ in R^3, where R^3 is now considered as the two-dimensional projective space. We let z_2, z_3 play the role of coordinates in the affine plane. Then the point (z_1, z_2, z_3) in R^3 is naturally projected onto the point in the affine plane that lies on the ray emanating from the origin and passing through (z_1, z_2, z_3). In what follows, by the *projection* onto the affine plane we shall always mean this natural projection. Given any set ϕ of points

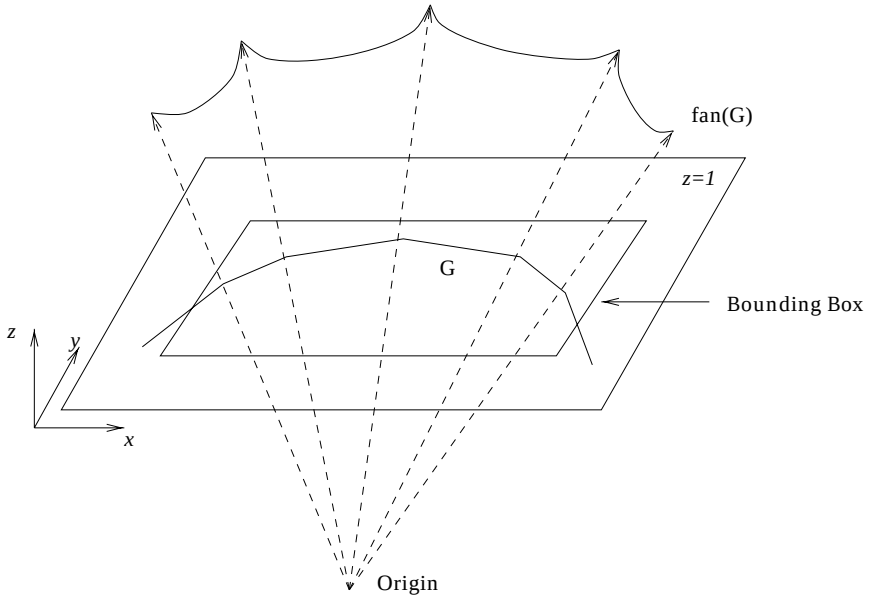


FIG. 4.1. *The graph G and its fan.*

in the affine plane, let fan(ϕ), the fan through ϕ, denote the points in R^3 that project onto the points in ϕ. Then $I(z_1, z_2, z_3)$ is feasible iff (z_1, z_2, z_3) lies below fan(G) in the z_3-direction; this also includes the case when (z_1, z_2, z_3) lies on fan(G).

For the sake of simplicity, let us rename the coordinates z_1, z_2, z_3 as z, x, and y, respectively. Thus $z = 1$ now serves as the affine plane in R^3. The preceding discussion implies that in the partition of Theorem 4.8, a permissible integer point in Z^3 is labeled *yes* iff it lies below fan(G) in the y-direction (Figure 4.1).

Since $\mathcal{P}$ has bitsize $\beta = \beta(n)$ and complexity $\rho = \rho(n)$ it follows that (1) the graph G is piecewise linear and convex with ρ bounded segments, and (2) all vertices of G have rational coordinates that can be expressed in terms of numerators and denominators of absolute value at most $\mu = 2^\beta$, or in other words, of bitsize at most $\beta = \log \mu$. Trivially $\rho \leq 4\mu^2$. Moreover, all vertices of G lie in the interior of the *bounding box* defined by $|x| \leq 2\mu$ and $|y| \leq 2\mu$ (Figure 4.1). Consider two horizontal planes, parallel to the affine plane, defined by $z = \bar{\mu}$ and $z = 2\bar{\mu}$, where $\bar{\mu}$ would be chosen to be much larger than μ. Let B denote the region (slab) between these two planes that is enclosed by the fan through the boundary of the bounding box in the affine plane. The coordinates of all integer points in this slab have bitsizes at most twice of $\bar{\beta} = \log \bar{\mu}$. The following theorem would imply that the partition as in Theorem 4.8 cannot exist.

THEOREM 4.9. *Let S be any set of δ planes in R^3. Assume that $\bar{\beta} = \log \bar{\mu}$ is greater than a large enough constant multiple of $\beta = \log \mu$ and that ρ is greater than a large enough constant multiple of δ^4. (Since $\rho \leq 4\mu^2$, this also means that $\log \bar{\mu}$ is greater than a sufficiently large constant multiple of $\log \delta$.) Then at least one face in the arrangement (partition) in B formed by the planes in S contains an integer point lying below* fan(G) *and also an integer point strictly above this fan.*

Remark. If ρ were less than δ, then one can let S be the set of planes that contain

the fans through the edges of G. No face in the resulting partition can contain an integer point below $\mathrm{fan}(G)$ and also an integer point strictly above $\mathrm{fan}(G)$ (in its interior). What the theorem says is that one cannot do much better than this trivial scheme.

Remark. It is possible to strengthen the theorem slightly so that ρ is only required to be greater than a large enough constant multiple of δ^3 instead of δ^4.

In the arithmetic linear PRAM model the issue of bitlengths is ignored; i.e., the running time of the algorithm cannot depend on the bitlengths of the input parameters. In that case we only need to prove the theorem when the slab B is unbounded, i.e., when its upper bounding plane goes to infinity. This is quite easy (as the reader can verify). In fact, then S must contain every plane containing the origin and an edge of G, so δ has to be at least ρ; this completes our lower bound proof for the arithmetic linear PRAM model.

Proof of Theorem 4.7 *from Theorem* 4.9. Before we prove the theorem, let us see why it implies that the partition as in Theorem 4.8 cannot exist. This will prove Theorem 4.7. Let $t = \sqrt{\log \rho}/a'$, as in Theorem 4.8. Let S be the set of planes mentioned in its statement. Their total number δ is at most 2^{5t^2}, so $\log \delta \leq 5t^2 = 5 \log \rho / a'^2$. Choose the constant a' large enough so that $\log \rho$ is sufficiently larger than $\log \delta$ as required in Theorem 4.9. Next fix $\bar{\mu}$ so that $\log \bar{\mu}$ is a sufficiently large multiple of $\log \mu$, also as required in Theorem 4.9. If we choose the permissibility constant a (cf. the statement of Theorems 4.1 and 4.7) large enough, then all integer points in the slab B will be permissible. Now Theorem 4.9 implies that the arrangement as in the first part of Theorem 4.8 cannot exist for such constants a and a'. The proof of the second part is similar. □

Proof of Theorem 4.9. Assume, to the contrary, that no face in the partition of B contains an integer point below $\mathrm{fan}(G)$ and also an integer point strictly above $\mathrm{fan}(G)$. In what follows, this property of the partition will be called the *separation property*. Without loss of generality, we can assume that no integer point in the slab B lies on any plane in S; otherwise, one can ensure this without losing the separation property by replacing each plane with two planes obtained by infinitesimal translation in the positive and negative normal directions. The regions of the partition of B are convex polytopes that can have many facets in general. Therefore we shall refine this partition, using a well-known idea in computational geometry (e.g., see [35]) so that each region in the resulting decomposition is a "cylinder" with at most six facets. Then using the pigeonhole principle we shall show that some cylinder in this decomposition must violate the separation property.

Decomposition. Let Q be the set that contains the planes in S and the planes bounding the slab B. Project onto the affine plane the intersection between every pair of planes in Q (restricted to the slab B). This yields an arrangement of lines within the bounding box in the affine plane (Figure 4.2). We refine this arrangement further by passing vertical lines (parallel to the y-axis) through all intersections among these lines and also intersections with the bounding box (Figure 4.2). Let us denote the resulting partition of the bounding box by $A(Q)$.

The affine partition $A(Q)$ can be lifted to the slab B in a natural fashion: We pass fans through all projected lines in the affine plane; these are the same as the fans through pairwise intersections of the planes in Q. We also pass fans through all auxiliary vertical lines that we added in the affine plane. These fans along with the planes in Q yield a decomposition of B; we shall denote it by $D(Q)$. The total number $d(Q)$ of cells (three-dimensional regions) in $D(Q)$ is $O(\delta^4)$, and each cell is a convex

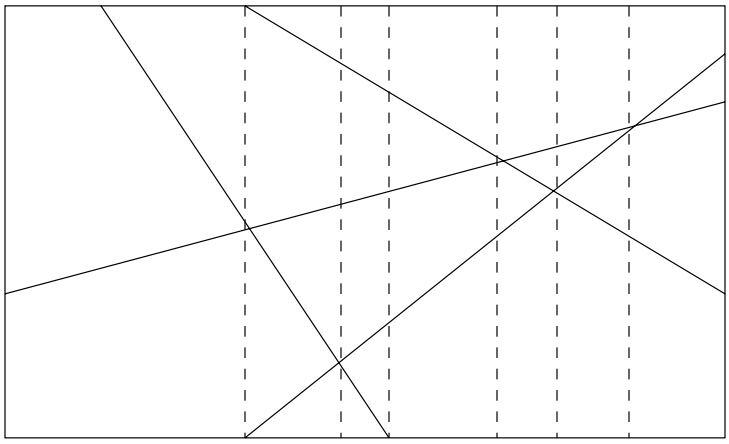


FIG. 4.2. *Decomposition $A(Q)$ in the affine plane.*

cylinder with at most six facets. By a cylinder we mean a convex polytope which has a unique top facet, called the *roof*, and a bottom facet, called the *floor*, as seen from the origin; by the floor we mean the facet that is hit first by any ray coming from the origin toward the cell, and the roof is defined similarly.

Sample points. Next, we choose rational sample points on all (bounded) edges of the graph G; in what follows we just ignore the two unbounded edges of G. Let c be a large enough positive integer constant to be specified later. Divide each edge of the graph into intervals of equal spans along the x-direction by placing δ^c points in its interior. These points will be called *sample points*. The x-coordinates of any two sample points differ by at least $1/(\mu\delta^c)$, because the coordinates of all vertices in G are rationals that can be expressed in terms of denominators and numerators of absolute value at most μ.

Given a sample point p on e, we say that a cell R in $D(Q)$ is *good* for p if its interior contains an integer (lattice) point lying on the ray through p (coming from the origin). We say that R is *good* for an edge e in the graph G if it is good for $[1/d(Q)]$th fraction of the sample points on e.

LEMMA 4.10. *$D(Q)$ contains a cell that is good for $[1/d(Q)]$th fraction of the edges in G.*

Proof. The number of cells in $D(Q)$ is at most $d(Q)$. Hence, by the pigeonhole principle, it suffices to show that for every edge e in the set there is at least one good cell in $D(Q)$. Applying the pigeonhole principle once again, it suffices to show that for every sample point p on e, there is at least one good cell in $D(Q)$. The ray through p is split into at most $\delta + 1$ intervals within B by the planes in Q. Thus the vertical span of at least one such interval—call it e_p—must be at least $\bar{\mu}/(\delta+1)$. The interior of e_p must contain an integer lattice point. This follows from the last property and the following:

1. The coordinates of the sample point p can be expressed as rationals with numerators and denominators of bitlength at most $\log \mu$, ignoring a constant factor.
2. By our assumption, $\log \bar{\mu}$ is greater than sufficiently large constant multiples

of $\log \mu$ and $\log \delta$.
Thus the cell in $D(Q)$ containing e_p is good for p. □

Given an integer point lying on the fan through an edge in G, let us define its neighbor to be the unique integer point with the same x and z coordinates but whose y coordinate is greater by one. Integer points on the fan through G are all feasible, whereas their neighbors are infeasible. This mean if a cell in $D(Q)$ contains an integer point on $\text{fan}(G)$ it cannot contain its neighbor.

The graph G contains ρ edges. By the preceding lemma, there is a cell C in $D(Q)$ that is good for at least $\bar{\rho} = \rho/d(Q)$ such edges. Their number is quite large since $d(Q) = O(\delta^4)$, and ρ is larger than a sufficiently large multiple of δ^4. This means there exist a large number of edges in the graph G such that C contains a large number of integer points on the fan through each of these many edges. But it cannot contain the neighbors of these integer points. Since C is a convex cylinder with at most six facets, this is impossible (as the reader can verify).

This proves Theorem 4.9.

5. PRAM without bit operations. In this section we shall prove our general lower bound for the PRAM model without bit operations (Theorem 3.3) by extending the proof in section 4. As there, we shall prove a stronger nonuniform lower bound (Theorem 5.7) from which Theorem 3.3 will follow. The main difference from the linear PRAM setting is the presence of general multiplication in the instruction set. This causes some algebraic geometry to enter the proof in an essential way.

5.1. The number of possible branchings. First, we extend the general technique for bounding the number of possible ways in which the processors can branch (section 4.1). We shall describe the technique in a language-independent setting, following the same notation as in section 4.1.

Let $\Sigma = \Sigma(x) = \Sigma(x_1, \ldots, x_n)$, $x_i \in Z$, be the language for which we want to prove a lower bound. Here $x_1, \ldots, x_n$ denote the integer parameters in the input. Fix a nonuniform machine in the PRAM model without bit operations with $p(n, N)$ processors, which works correctly on inputs with n parameters with total bitlength at most N in $t(n, N)$ time. We define the t-equivalence classes with respect to this machine just as in section 4.1, the main difference being that the processors can now multiply. The number of t-equivalence classes, in general, does not have a good upper bound. We get around this problem, as in section 4.1, by parametrizing the inputs to the machine and then bounding the number of t-equivalence classes among the parametrized inputs.

Let us imagine running our machine on the set of possible inputs defined by $x_i = l_i(z) = l_i(z_1, \ldots, z_d)$, where for each $i \leq n$ the parametrizing linear form l_i is exactly as in section 4.1. As there, let $\Sigma(z_1, \ldots, z_d)$ denote the set of permissible d-tuples accepted by the machine. We say that z is accepted if $x = l(z) = (\ldots, l_i(z), \ldots)$ is accepted. Two permissible d-tuples z, z' are said to be t-equivalent if $l(z)$ and $l(z')$ are t-equivalent. The notion of equivalence is defined similarly. Let $\sigma(t)$ denote the number of t-equivalence classes among the permissible values of z. Let $\sigma = \sigma(t(n, N))$ denote the number of equivalence classes. The following theorem generalizes Theorem 4.3.

THEOREM 5.1. *For all t, $\sigma(t)$ is bounded by $[2 + 2 \cdot 2^t p(n, N)]^{dt}$.*

Proof. Let C denote a fixed t-equivalence class. Let z denote a (permissible) integral value in this equivalence class. In what follows, we pretend that z is a generic variable. We are told that it belongs to C. But we are not told its actual value. We imagine running our machine on the input $x = l(z)$. We have to specify what is

meant by executing an arithmetic operation in this setting. Now the operand would be polynomials in z, rather than linear forms as in the linear PRAM setting. By an arithmetic operation $\times$, $+$, or $-$, we simply mean the corresponding polynomial operation. Since nonnumeric parameters are fixed by our parameterization, indirect memory references are no problem. We assume that when a processor is executing a comparison it is simply told the result of that comparison, which, at any time less than or equal t, is fixed by the class C. The following lemma generalizes Lemma 4.4.

LEMMA 5.2. *Let C be a fixed t-equivalence class. Then for every memory location j, there exists a polynomial $g_j[C,t](z)$ that gives the content of that location at time t for every $z \in C$. Moreover, the degree of g_j is at most 2^t.*

Proof. Once C is fixed, the execution path of each processor up to time t is fixed. Therefore, each memory location at time $\leq t$ is a completely determined function of $z_1, \ldots, z_d$, in fact, a polynomial function. At time $t = 0$, each memory location is either a constant or is equal to some $l_i(z)$; so it is a polynomial of degree at most one. Unlike in the linear PRAM setting, now we have general multiplication. This corresponds to multiplying two polynomials in $(z_1, \ldots, z_d)$; the degree of their product is the sum of their degrees. The lemma follows by induction on t. ∎

Let us now see what happens at time t, given that the generic variable z belongs to a fixed t-equivalence class C as above. If a processor is to execute an arithmetic operation at time t, then the result of that operation is a certain uniquely determined polynomial to be stored in a certain uniquely determined location in the memory. If the operation is a branch, the branching path is determined by comparing contents of two uniquely determined memory locations, which are certain fixed polynomials in z. In other words, each branch is determined by testing a certain polynomial equality or inequality in z of the form $g(z) : 0$, where the comparison : is $>$, $\geq$, or $=$. The number of such polynomial constraints is at most $p(n, N)$, one per processor. Let $g_1(z), g_2(z), \ldots$ be the nonzero polynomials involved in these constraints. Consider the partition (stratification) of R^d formed by the hypersurfaces $g_i(z) = 0$. A sign-invariant component of this stratification is defined to be the maximal set of points such that the sign of any g_i remains the same over the whole set; it need not be connected or even simply connected. The sign of a function at a given point is defined to $+$, $-$, or 0 depending on whether its value at the point is positive, negative, or zero. We say that a sign-invariant component is nonempty if it contains a permissible integer point.

THEOREM 5.3 (Milnor–Thom). *The number of nonempty sign-invariant components in the preceding stratification of R^d is bounded by $[2 + 2\sum \deg(g_i)]^d$.*

Proof. If no hypersurface under consideration contains any permissible integer point, then this follows directly from the Milnor–Thom bound [34, 49] on the number of connected components of the set $\bigcup_i \{g_i(z)^2 > 0\}$. Otherwise, we can replace each hypersurface $g_i(z) = 0$ by two hypersurfaces $g_i(z) = \epsilon$ and $g_i(z) = -\epsilon$, where ϵ is an infinitesimal positive real (Figures 5.1 and 5.2). Since the number of permissible integer points is finite, we can choose ϵ small enough so that no new hypersurface contains any permissible integer point and the partition of the permissible integer points induced by the sign-invariant components of the old stratification coincides with the partition induced by the sign-invariant components of the new stratification. Now we can apply the Milnor–Thom bound as before. ∎

Because the number of polynomials g_i is at most $p(n, N)$, one per processor, and $\deg(g_i)$ is at most 2^t, it follows from the preceding result that the total number of sign-invariant components in the preceding partition of R^d is at most $[2+2p(n, N)2^t]^d$.

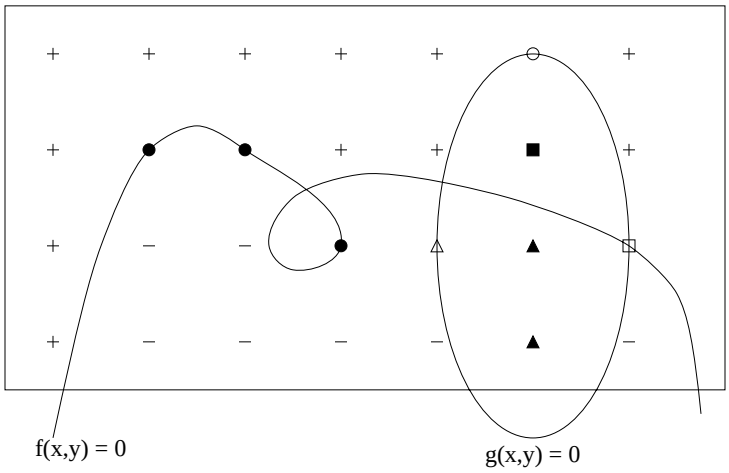


FIG. 5.1. *Before perturbation: integer points in different sign-invariant components are depicted differently.*

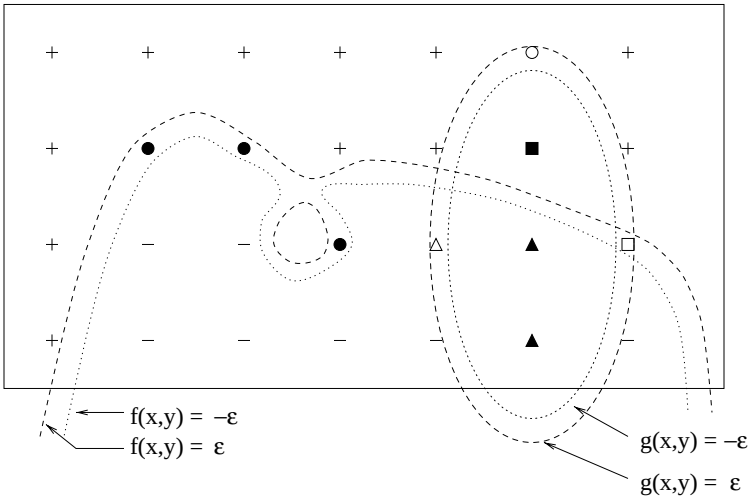


FIG. 5.2. *After perturbation: the sign-invariant partition of the integer points is unchanged.*

Now we make the crucial observation.

OBSERVATION 5.4. *Any two $z, z' \in C$ belong to the same $(t+1)$-equivalence class if they belong to the same sign-invariant component of this partition.*

It immediately follows that

$$\sigma(t+1) \leq [2 + 2p(n,N)2^t]^d \sigma(t).$$

Induction on t proves Theorem 5.1. □

The proof of Theorem 5.1 also yields the following lemma.

LEMMA 5.5. *For any t-equivalence class C, there exists a set $\Phi(C,t)$ of at most*

$p(n, N)t$ polynomial constraints (equalities or inequalities) in d variables such that $z \in C$ iff all these constraints are satisfied. Moreover, the degree of every polynomial occurring in $\Phi(C, t)$ is at most 2^t.

Proof. The constraints correspond to the branches taken by all processors before time t, which are all fixed by the class C. □

By Lemma 5.5, each equivalence class D of permissible z-values is characterized by a set $\Phi(D) = \Phi(D, t(n, N))$ of polynomial constraints. Let $\Phi = \cup_D \Phi(D)$, where D ranges over all equivalence classes. By Theorem 5.1 and Lemma 5.5, it follows that Φ contains at most

$$[2 + 2p(n, N)2^{t(n,N)}]^{dt(n,N)} p(n, N)t(n, N)$$

polynomials of degree at most $2^{t(n,N)}$. Consider the partition of R^d formed by the hypersurfaces defined by these polynomials. Each sign-invariant component of this partition can be labeled *yes* or *no* in such a way that a permissible z is accepted by the machine iff it lies in a sign-invariant component labeled *yes*. Thus we have proved the following.

THEOREM 5.6. *Assume that the language $\Sigma(z_1, \ldots, z_d)$ is accepted in the PRAM model without bit operations using $p(n, N)$ processors in $t(n, N)$ time. Then R^d can be partitioned by a set of at most*

$$[2 + 2p(n, N)2^{t(n,N)}]^{dt(n,N)} p(n, N)t(n, N)$$

polynomials of degree at most $2^{t(n,N)}$ and each sign-invariant component can be labeled yes *or* no *so that a permissible $z \in Z^d$ belongs to $\Sigma(z_1, \ldots, z_d)$ iff it lies in a sign-invariant component labeled* yes.

In the proof of this theorem, performance of the machine on N-bit inputs that do not lie in the parameterized family $\Sigma(z_1, \ldots, z_d)$ does not matter. In other words, the theorem holds even if $t(n, N)$ is defined to be the maximum time taken on any N-bit input in $\Sigma(z_1, \ldots, z_d)$.

5.2. Parameterization. We shall now prove a general lower bound in the PRAM model without bit operations that is nonuniform with respect to parameterization. It generalizes Theorem 4.7, and we follow the same notation as in section 4.2.

THEOREM 5.7. *Assume that the permissibility constant a is large enough. Then no machine in the PRAM model without bit operations can decide whether $w \leq F(I)$ correctly for every $I \in \mathcal{I}$ in $\sqrt{\log \rho(n)}/a'$ time using $2^{\sqrt{\log \rho(n)}/a'}$ processors for some large enough constant a' that does not depend on the permissibility constant a.*

In the deterministic setting, Theorem 3.3 follows from this theorem by applying it to a parameterization $\mathcal{P}$ for cardinality n with bitsize $\beta(n)$ whose complexity is equal to $\phi(n, \beta(n))$, the maximum possible value. Randomized setting is considered in section 6.1. In what follows, we shall denote $\rho(n)$ and $\beta(n)$ by simply ρ and β, respectively.

Let us apply Theorem 5.6 with $d = 3$ to the linear parameterization of the input as in Theorem 5.7. It is defined (as in section 4.2) by the linear map

$$(z_1, z_2, z_3) \to I(z_1, z_2, z_3) = (z_3, \tilde{\mathcal{P}}(z_1, z_2)).$$

Thus in Theorem 5.6 $\Sigma(z_1, z_2, z_3)$ consists of those tuples (z_1, z_2, z_3) such that $z_3 \leq \tilde{\mathcal{P}}(z_1, z_2)$. This gives the following theorem.

THEOREM 5.8. *Let a' be any positive constant. If a machine in the PRAM model without bit operations works correctly on the input $I(z_1, z_2, z_3)$ for every permissible*

z_1, z_2, and z_3, in $t = \sqrt{\log \rho}/a'$ time using 2^t processors, then R^3 can be partitioned by at most 2^{20t^2} algebraic surfaces of degree at most 2^t and each sign-invariant component of this partition can be labeled yes *or* no *so that a permissible integer point (z_1, z_2, z_3) lies in a sign-invariant component labeled* yes *iff $I(z_1, z_2, z_3)$ is feasible.*

5.3. A lattice problem. We shall prove Theorem 5.7 by showing that an algebraic partition as in Theorem 5.8 cannot exist for large enough constants a and a'. We proved this in the linear setting in section 4.3. We wish to generalize that proof to the algebraic setting.

Let G be the graph corresponding to the optimum function $\mathcal{P}$ as in section 4.3 (Figure 4.1). Let $\text{fan}(G)$ be its fan. Let us also rename the coordinates z_1, z_2, z_3 as z, x, and y, respectively. Thus $z = 1$ now serves as the affine plane in R^3. Then in the partition of Theorem 5.8, a permissible integer point in Z^3 is labeled *yes* iff it lies below $\text{fan}(G)$ in the y-direction.

Recall that (1) the graph G is piecewise linear and convex with ρ segments, and (2) all vertices of G have rational coordinates that can be expressed in terms of numerators and denominators of absolute value at most $\mu = 2^\beta$ or, in other words, of bitsize at most $\log \mu$. All vertices of G lie in the interior of the bounding box defined by $|x| \le 2\mu$ and $|y| \le 2\mu$. Consider two horizontal planes, parallel to the affine plane, defined by $z = \bar{\mu}$ and $z = 2\bar{\mu}$, where $\bar{\mu}$ would be chosen to be much larger than μ. Let B denote the region (slab) between the two planes that is enclosed by the fan through the boundary of the bounding box in the affine plane. The coordinates of all integer points in this slab have bitsizes at most twice $\bar{\beta} = \log \bar{\mu}$. The following theorem would imply that the partition as in Theorem 5.8 cannot exist.

THEOREM 5.9. *Let S be any set of surfaces in R^3 with total degree δ. Assume that $\bar{\beta} = \log \bar{\mu}$ is greater than a large enough constant multiple of $\beta = \log \mu$ and that $\log \rho$ is greater than a large enough constant multiple of $\log \delta$. Then at least one sign-invariant component in the partition of B formed by the surfaces in S contains an integer point lying below $\text{fan}(G)$ and also an integer point strictly above this fan.*

Remark. Since $\rho \le 4\mu^2$ trivially, the assumption in the theorem also implies that $\log \bar{\mu}$ is much greater than $\log \delta$.

Before we prove the theorem, let us see why it implies that the partition as in Theorem 5.8 cannot exist; this will prove Theorem 5.7. Let S be the set of surfaces that occur in Theorem 5.8. Their total degree δ is at most $2^t 2^{20t^2} = 2^{21t^2}$, where $t = \log \rho/a'$. Then choose the constant a' and the permissibility constant a large enough and apply Theorem 5.9 to this set S exactly as we applied Theorem 4.9 to the set of planes occurring in Theorem 4.8.

In the rest of this section we shall prove Theorem 5.9. Assume, to the contrary, that no sign-invariant component in the partition of B contains an integer point below $\text{fan}(G)$ and also an integer point strictly above $\text{fan}(G)$. In what follows, this property of the partition will be called the *separation property.* We shall first smooth and refine the partition of B so that the regions in the resulting partition have a simple shape. Then by the pigeonhole principle we shall show that some region in this partition must violate the separation property.

Transversality. The surfaces S in Theorem 5.9 depend on the nonuniform machine under consideration. In general, they can be highly singular; this cannot be helped, since we have no control over the nonuniform machine. Singularities can cause problems in our proof, but fortunately we can get rid of them as follows.

First, we can assume that no integer point in the slab B lies on any surface in S.

Otherwise, one can ensure this without losing the separation property by replacing each surface with two surfaces obtained by perturbing the constant term in its defining equation infinitesimally; since there are only finitely many integer points in B, we can choose the perturbation small enough so that the underlying partition of the integer points in B according to sign-invariant components does not change (see Figures 5.1 and 5.2). We can also add infinitesimal reals freely to the coefficients of the polynomials defining the surfaces in S without losing the preceding or the separation property. In other words, we can assume that the surfaces in S are in *general position*—this just means we have the freedom to perturb them by adding infinitesimal reals to the coefficients of their defining equations. The general position assumption lets one handle the technical problems mentioned above in a clean fashion, because then the transversality techniques from differential geometry (see [16, Chapter 2]) become fully applicable. For example, by perturbing the surfaces in S infinitesimally we can now assume that they are smooth (Sard's theorem), that they intersect transversally, that their *silhouettes* (to be defined soon) are smooth, and so forth.

Collins's decomposition. Let Q be the set that contains (1) the surfaces in S, (2) the planes bounding the slab B, and (3) 6δ horizontal *dividing* planes that subdivide the slab B into horizontal slabs of equal height. By horizontal, we mean parallel to the affine plane.

Consider the partition of B formed by the surfaces in Q. The regions in this partition can have very complicated shapes. Therefore, we shall refine the partition further so that the regions in the resulting partition have simple shapes following a variant of the method due to Collins [10]. Before we define the partition formally, let us make a definition.

Let s be a surface in S. We say that a point p on s belongs to the *silhouette* of s, as seen from the origin, if the line joining p and the origin is tangent to s at p. This happens iff

$$f(x,y,z) = 0 \text{ and } x\frac{\partial f}{\partial x} + y\frac{\partial f}{\partial y} + z\frac{\partial f}{\partial z} = 0, \tag{5.1}$$

where $f(x,y,z) = 0$ is the polynomial equation defining s. The silhouette of s, if nonempty, is a smooth space curve if s is in general position (see [16, Chapter 2]).

Let us project onto the affine plane (1) the silhouette of every surface in S (restricted to the slab B), and (2) the intersection between every pair of surfaces in Q; by our general position assumption, this intersection is a smooth space curve.

The preceding projection yields an arrangement of curves within the bounding box in the affine plane (Figure 5.3). We refine this arrangement further by passing vertical lines (parallel to the y-axis) through all intersections among the curves, singular[2] points on the curves, and also critical points on the curves where the tangents become parallel to the y-axis. Let us denote the resulting partition of the bounding box by $A(Q)$. It is a two-dimensional Collins' decomposition. Every region (cell) in this partition is monotone with respect to the y direction. In other words, its intersection with any line parallel to the y-axis is connected, if nonempty.

The affine partition $A(Q)$ can be lifted to the slab B in a natural fashion: We pass fans through all projected curves in the affine plane; these are the same as the fans through pairwise intersections and silhouettes of the surfaces in Q. We also pass fans through all auxiliary vertical lines that we added in the affine plane. These fans

[2]Transversality conditions cannot remove such singular points on projections.

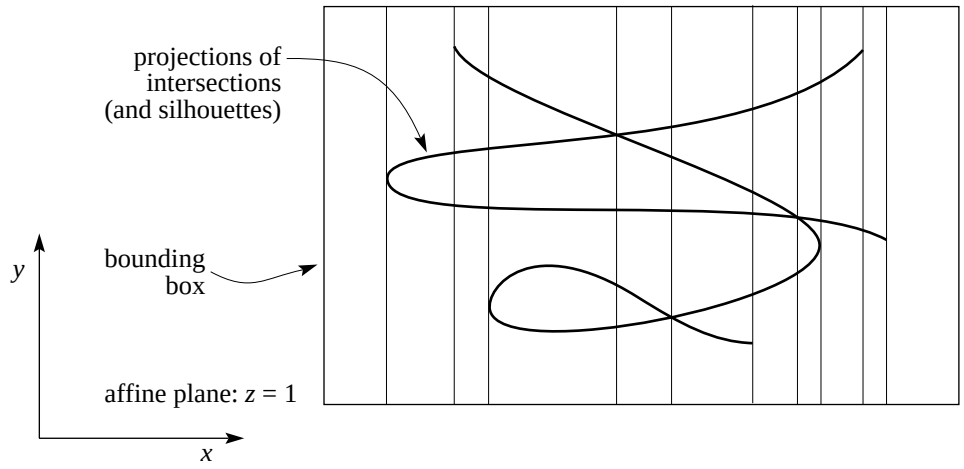


FIG. 5.3. *Decomposition $A(Q)$ in the affine plane.*

along with the surfaces in Q yield a decomposition of B. We shall denote it by $D(Q)$. It has the following properties [10]:

1. Each region, i.e., a three-dimensional cell in $D(Q)$, has at most six sides. It is monotone when viewed from the origin. In other words, its intersection with any ray is connected, if nonempty. By a ray, we mean here and in what follows a ray coming from the origin. (In Collins's original construction, the regions of $D(Q)$ are monotone with respect to the z-direction.) Monotonicity allows us to define the top side, the *roof*, and the bottom side, the *floor*, of each region unambiguously. Formally, the floor consists of all points on the boundary of the region that are hit first by the rays coming from the origin; the roof is defined similarly.
2. The floor of each region is contained in just one surface in Q; the same is true of the *roof*.
3. The remaining sides of the region are contained in fan surfaces.
4. The projection of each cell in $D(Q)$ is a cell in the affine partition $A(Q)$.
5. The total number $d(Q)$ of cells in $D(Q)$ is $O(\delta^{O(1)})$. This follows from the Milnor–Thom result [34, 49] because the total degree of the surfaces in Q, along with the fan surfaces, is $O(\delta^{O(1)})$.

Sample points. Next, we choose rational sample points on the edges of the graph G exactly as in section 4.3. The x-coordinates of any two sample points differ by at least $1/(\mu\delta^c)$.

Let us call a cell R in $D(Q)$ *flat* if its roof and also the floor are contained in the dividing horizontal planes. Given a sample point p on e, we say that R is *good* for p if its interior contains an integer (lattice) point lying on the ray through p. We say that R is *good* for a bounded edge e in the graph G if it is good for $[1/d(Q)]$th fraction of the sample points on e.

LEMMA 5.10. *$D(Q)$ contains a flat cell that is good for $[1/d(Q)]$th fraction of the edges in G.*

Proof. The number of flat cells in $D(Q)$ is at most $d(Q)$. Hence, by the pigeonhole

principle, it suffices to show that for every bounded edge e in G there is at least one good flat cell in $D(Q)$. Applying the pigeonhole principle once again, it suffices to show that for every sample point p on e, there is at least one good flat cell in $D(Q)$. The ray through p is split into $6\delta+1$ intervals by the 6δ dividing planes. At most δ of these intervals are intersected by the surfaces in S. To see this, note that the surfaces in S are in general position and the sample points do not depend on S. Hence, the ray through p is intersected by the surfaces in S in a finite number of points, and this number is at most δ, the total degree of the surfaces in S, by the simplest special case of the Milnor–Thom result (which is essentially Bezout's result). It follows that there is at least one such interval in the interior of B that is not intersected by any surface in S. Denote this interval by e_p. Let P_1 and P_2 be the consecutive dividing planes adjacent to this interval. Note the following:

1. The interval e_p is contained in some cell of $D(Q)$.
2. The endpoints of e_p lie on P_1 and P_2. But they do not lie on any surface in S if the surfaces in S are in general position. The latter fact follows because the rays through the sample points on the graph G as well as their intersections with the dividing planes are finite in number and they do not depend on S. Hence, we can perturb the surfaces in S infinitesimally before the construction of the decomposition $D(Q)$ begins and ensure that no surface in S contains any such intersection.
3. The preceding property, in conjunction with the properties of $D(Q)$, implies that the roof and the floor of the cell in $D(Q)$ containing e_p must be contained in the planes P_1 and P_2, respectively. Hence, this cell must be flat.
4. The vertical span of e_p is $\bar{\mu}/(6\delta+1)$.

The interior of e_p must contain an integer lattice point. This follows from the last property just as in the proof of Lemma 4.10. Thus the flat cell containing e_p is good for p. □

The graph G contains ρ bounded edges of distinct slopes. By the preceding lemma, there is a flat cell C in $D(Q)$ that is good for at least $\bar{\rho} = \rho/d(Q)$ such edges. Let these $\bar{\rho}$ edges be $e_1, e_2, \ldots$ ordered from left to right in the affine plane. Such ordering is possible because the graph G, being a function graph, is monotone with respect to the y direction. Let T be the projection of C onto the affine plane. It belongs to the affine partition $A(Q)$. Hence, it is monotone in the y direction. Let g denote the curve containing the upper side of T. By the definition of $A(Q)$, the upper side of T is monotone with respect to the y direction and contains no singularities. Hence, it defines a certain smooth function $y = \bar{g}(x)$ within the span of T along the x-axis.

LEMMA 5.11. *The second derivative of the function $\bar{g}(x)$ has at least $\bar{\rho}-1$ extrema within the span of T along the x-axis.*

Proof. The basic idea is to show that the curve g gets very close to several sample points on each segment e_i. Since these segments are not too far apart, and their slopes are sufficiently different, g must change its direction several times (Figure 5.4).

Fix any e_i. The cell C is good for e_i. Hence, it is good for at least $[1/d(Q)]$th fraction of the sample points on e_i. Since the number of sample points on e_i is δ^c and $d(Q)$ is $O(\delta^{O(1)})$, there are at least eight such sample points in the interior of each e_i, assuming that the constant c is large enough. In what follows, these eight sample points will be the only active sample points on e_i; all other sample points are discarded. Order these remaining sample points from left to right. Fix any such sample point p. It must lie within T because C contains an integer lattice point—call

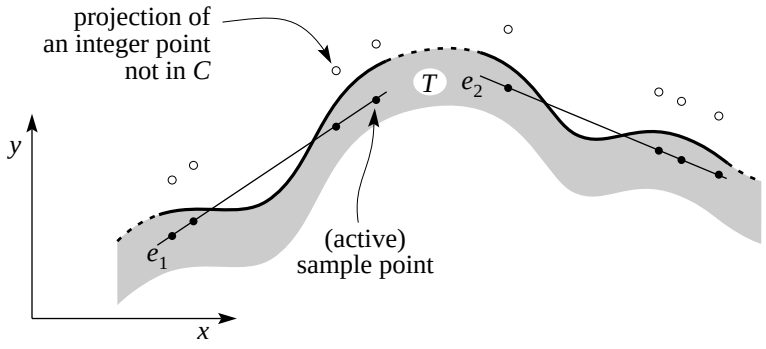


FIG. 5.4. *Upper side of T.*

it $\bar{p}$—on the ray through p.

Claim. The difference between the y-coordinate of p and the value of the function $\bar{g}(x)$ at the x coordinate of p is at most $1/\bar{\mu}$.

Proof of the claim. Consider the integer point $\hat{p}$ with the same x and z coordinates as the integer point $\bar{p}$ but whose y coordinate is one plus the y-coordinate of $\bar{p}$. This point certainly lies in the slab B since the bounded edge of G containing p is well within the interior of the bounding box in the affine plane. The point $\hat{p}$ does not lie on any ray through the graph G because p lies on G and G is monotone in the y direction. Since our partition has the separation property, the cell C cannot contain $\hat{p}$ because it already contains $\bar{p}$. Let q denote the projection of $\hat{p}$ onto the affine plane. It lies above p and has the same x-coordinate as p. Moreover, its y-coordinate differs from that of p by at most $1/\bar{\mu}$. This follows because the z-coordinate of $\hat{p}$—in fact, of every point in the slab B—is at least $\bar{\mu}$. Since the cell C does not contain $\hat{p}$ and T is monotone in the y direction, the upper side of T must separate p and q (Figure 5.4). This proves the claim.

Let $a_1, a_2, \ldots$ be the slopes of $e_1, e_2, \ldots$. They are all different. The difference between any two slopes, ignoring a constant factor, is at least $1/\mu^2$ because the coordinates of all vertices in G can be expressed as rationals with numerators and denominators of absolute value at most μ. The x-coordinates of any two sample points differ by at least $1/(\mu\delta^c)$, as we remarked earlier. These facts, in conjunction with the previous claim and Roelle's mean value theorem in elementary calculus, imply the following:

1. The derivative $d\bar{g}(x)/dx$ attains a value close to a_i between the x-coordinates of the jth and $(j+1)$st sample points on e_i for all $1 \le j < 8$. By close we mean that the error term is $O(\mu\delta^c/\bar{\mu})$.
2. This implies further that the second derivative attains a value close to zero between the x-coordinates of, say, the third and the sixth sample points on e_i. Similarly, somewhere between the x-coordinates of the seventh sample point on e_i and the second sample point on e_{i+1}, the absolute value of the second derivative is $\Omega(|a_{i+1} - a_i|/\mu)$, ignoring an additive $O(\mu^2\delta^{2c}/\bar{\mu})$ error term; here one also needs to use the fact that the x-coordinates of any two sample points differ by at most μ.
3. The preceding fact implies that every pair (e_i, e_{i+1}) contributes at least one

distinct extremum of the second derivative because $\log \bar{\mu}$ is greater than sufficiently large constant multiples of $\log \mu$ and $\log \delta$.

The lemma follows from the last fact. □

Claim. The second derivative of the function $\bar{g}(x)$ has at most $O(\delta^{O(1)})$ extrema in the span of T along the x-axis.

Proof. Let $g(x, y) = 0$ be the polynomial equation satisfied by the upper side of T. Its degree is at most $O(\delta^{O(1)})$ because the curve g containing the upper side results by projecting a silhouette or an intersection of two surfaces in S. Differentiating this equation three times in a row, we get four implicit equations satisfied by x, y and the three derivatives $y^{(1)}, y^{(2)}$, and $y^{(3)}$:

$$g(x, y) = 0,\ g_1(x, y, y^{(1)}) = 0, \ldots,$$

where $y^{(k)}$ is a formal variable that denotes the kth order derivative. In addition, the extrema satisfy $y^{(3)} = 0$. Because the surfaces in S are in a general position, these five equations in five variables have a finite number of real solutions. This number is $O(\delta^{O(1)})$, by the Milnor–Thom result.[3] Hence, the claim follows. (Formally, finiteness of the number solutions used above follows from the jet-transversality results; cf. [16, Chapter 2]. Alternatively, one can forget about transversality and just bound the number of connected components of the solution set using the Milnor–Thom result. The rest of the proof then needs to be modified a bit.)

The claim contradicts the preceding lemma because $\bar{\rho} = \rho/d(Q)$, $d(Q) = \delta^{O(1)}$, and $\log \rho$ is larger than a sufficiently large multiple of $\log \delta$.

This proves Theorem 5.9. □

6. Extensions.

6.1. Randomized algorithms.

All our lower bounds extend to randomized algorithms in the PRAM model without bit operations. The following result generalizes Theorem 3.3.

THEOREM 6.1. *Fix any homogeneous optimization problem. Let $\phi(n, \beta(n))$ be its parametric complexity for input cardinality n and bitsize $\beta(n)$. Then there exists a large enough constant b such that the decision version of the problem cannot be solved in the PRAM model without bit operations using randomization in $\sqrt{\log[\phi(n, \beta(n))]}/b$ expected time using $2^{\sqrt{\log[\phi(n,\beta(n))]}/b}$ processors; this is so even if we restrict every numeric parameter in the input to be an integer with bitlength at most $a\beta(n)$ for a large enough constant a.*

This also applies to the additive approximation version of the problem.

To prove it, we need only to extend Theorem 5.7. Its analogue in the randomized setting, following the same notation, is the next theorem.

THEOREM 6.2. *Assume that the permissibility constant a is large enough. Then no randomized machine in the PRAM model without bit operations can decide whether $w \leq F(I)$ correctly for every $I \in \mathcal{I}$ in $T = \sqrt{\log \rho(n)}/a'$ expected time using $p = 2^{\sqrt{[\log \rho(n)}/a'}$ processors with at most ϵ two-sided-error probability for some large enough constant a' that does not depend on the permissibility constant a.*

One can similarly extend Theorem 4.7 for the linear PRAM model; we leave this to the reader.

[3] Or one can also apply Bezout's theorem, but one has to be a bit careful. Bezout's theorem is applicable only if the number of complex solutions, not just the real solutions, is finite. This technicality can be taken care of as in Lemma 1 of Milnor [34].

Now we shall prove Theorem 6.2. In what follows, we shall denote $\rho(n)$ and $\beta(n)$ by simply ρ and β, respectively. Let $I(z_1, z_2, z_3)$ denote the total parametrized input, which includes the threshold, as in section 5.2. Let Φ be any suitably chosen subset of the permissible integer points in Z^3. Consider a probability distribution on Z^3 that is zero outside Φ and uniform on Φ. By Yao's lemma [54], it suffices to show, ignoring factors of two, that there is no deterministic machine M in the PRAM model without bit operations containing p processors that runs in expected time T, possibly erring on at most ϵ fraction of the points in Φ; here the expectation is meant to be with respect to the probability distribution on the input parameters. We can assume that ϵ is a small enough constant, say, $1/100$, by increasing T suitably.

Fix another positive constant $\bar{\epsilon} = 1/100$. Let us call a permissible integer point z in Φ *satisfactory* with respect to Φ, ϵ, and $\bar{\epsilon}$ if M stops on the corresponding input $I(z)$ in $t = T/\bar{\epsilon}$ time without any error. The proof of Theorem 5.8 can be extended *verbatim* to prove the following.

THEOREM 6.3. *Suppose a deterministic machine M as described above exists in the PRAM model without bit operations. Then R^3 can be partitioned by at most 2^{20t^2} algebraic surfaces of degree at most 2^t and each sign-invariant component of this partition can be labeled* yes *or* no *so that a satisfactory permissible integer point (z_1, z_2, z_3) lies in a sign-invariant component labeled* yes *iff $I(z_1, z_2, z_3)$ is feasible.*

Now we can generalize the proof of Theorem 5.7 to the randomized setting. First, let us specify the set Φ that fixes the input distribution. Let G be the graph as in section 5.3. Recall that the ray through each sample point on the graph G is split into exactly $6\delta+1$ intervals by the horizontal dividing planes. Moreover, each such interval contains a permissible integer point (cf. the proof of Lemma 5.10). Fix any such integer point on every such interval; in what follows, it will be called a *distinguished* point. The *neighbor* of this distinguished point is defined to be the integer point in the slab B with the same x and z coordinates as the distinguished point, but with the y coordinate greater by plus one. The graph G contains ρ (bounded) edges with distinct slopes. We shall let Φ be the set of distinguished points, along with their neighbors, on all rays through the sample points on these edges. Observe that the neighbors of all distinguished points are disjoint; this follows because G is monotone in the y-direction. Hence, exactly half of the points in Φ are distinguished. Let us call a distinguished point *satisfactory* if it is satisfactory and so is its neighbor. The number of unsatisfactory distinguished points is at most $2(\epsilon + \bar{\epsilon})$th fraction of the number of points in Φ. Let us call a sample point on an edge of G *satisfactory* if at least half of the distinguished points on the ray through it are satisfactory. Since the rays through all sample points have exactly the same number of distinguished points, the number of such unsatisfactory sample points is at most $4(\epsilon + \bar{\epsilon})$th fraction of the total number of sample points on the edges of G. Let us call an edge of G *satisfactory* if at least half of the sample points on that edge are satisfactory. Since all edges of G have the same number of sample points, the number of unsatisfactory edges is at most $8(\epsilon+\bar{\epsilon})\rho \leq 8\rho/50$. Discard all unsatisfactory edges and all unsatisfactory sample points on the satisfactory edges. We are still left with a large number of satisfactory quantities.

Now we can translate the proof of Theorem 5.7 with the use of Theorem 5.8 substituted with that of Theorem 6.3, making sure that everything involved is satisfactory. Thus, we say that a flat cell R in $D(Q)$ is good for a (satisfactory) sample point p; we mean that R contains a (satisfactory, distinguished) integer point on the ray through p. The proof of Lemma 5.10 can be translated easily since the number

of unsatisfactory intervals, i.e., the intervals containing unsatisfactory distinguished points, on any sample ray is at most $3\delta + 1$, and they can be thrown away. Similarly, Lemma 5.11 holds in this extended setting because the only integer points involved in its proof can be assumed to be satisfactory distinguished points and their satisfactory neighbors.

This proves Theorem 6.2.

6.2. PRAM with limited bit operations. We now indicate how our lower bounds can be extended to the PRAM model with limited bit operations.

THEOREM 6.4. *The lower bounds in Theorems* 3.3 *and* 6.1 *also hold in the PRAM model with limited bit operations.*

For the sake of simplicity, we shall consider only the deterministic setting.

We indicate only the changes that need to be made to the proof for the PRAM model without bit operations. Suppose there were to exist a machine M in this model for the problem under consideration that works in $t(n, N)$ time using $p(n, N)$ processors. Let $z_1, \ldots, z_d$ be the integer parameters that parametrize the input to this machine as in section 5.1. Let $\sigma(t)$ denote an equivalence class of parametrized inputs at time t, where two inputs belong to the same $\sigma(t)$ iff all processors in the machine follow the same branching path up to time t when the machine is started on either of the inputs.

Claim. Fix an equivalence class $\sigma(t)$ and the value of the parametrized input $z = (z_1, \ldots, z_d)$ modulo $2^{t(n,N)}$; i.e., z_i modulo $2^{t(n,N)}$ is fixed for every i. Then for all inputs z in this equivalence class having the same value modulo $2^{t(n,N)}$, and any time t, the value of any memory location modulo $2^{t(n,N)-t}$ at that time is the same.

Proof. Proof is by induction on t. The base case is clear. Consider $t > 1$. Once $\sigma(t)$ is fixed, so is the corresponding $\sigma(t-1)$. By the induction hypothesis, the values of all memory locations modulo $2^{t(n,N)-t+1}$ are fixed. Consider any memory location u. At time t, either u remains unchanged or is changed in accordance with an assignment of type:

1. $u := v\{+, -, \text{or}, *\}w$, where v and w are memory locations or constants; in this case the value of u at time t modulo $2^{t(n,N)-t}$, in fact, modulo $2^{t(n,N)-t+1}$ as well, is fixed by the values of v and w at time $t-1$ modulo $2^{t(n,N)-t+1}$; or
2. $u := \lfloor u/2 \rfloor$; in this case the value of u at time t modulo $2^{t(n,N)-t}$ is fixed by its value at time $t-1$ modulo $2^{t(n,N)-t+1}$. □

It follows that once we fix the input z modulo $2^{t(n,N)}$, and an equivalence class $\sigma(t)$, $t \leq t(n, N)$, then each assignment of type $u := \lfloor u/2 \rfloor$ that occurs on the computation path of any processor up to time t can be replaced unambiguously by a rational assignment $u := u/2$ or $u := u/2 - 1$. Now it can be checked that the proof of Theorem 5.6 goes through verbatim provided we confine ourselves to z-inputs having the same value modulo $2^{t(n,N)}$. For the sake of simplicity, we shall confine ourselves with z-inputs that are zero modulo $2^{t(n,N)}$; we call them *green* inputs. We are thus led to the following extension of Theorem 5.6 in the extended model.

THEOREM 6.5. *Assume that the language $\Sigma(z_1, \ldots, z_d)$ is accepted in the PRAM model with limited bit operations using $p(n, N)$ processors in $t(n, N)$ time. Then R^d can be partitioned by a set of at most*

$$[2 + 2p(n, N)2^{t(n,N)}]^{dt(n,N)} p(n, N)t(n, N)$$

polynomials of degree at most $2^{t(n,N)}$ and each sign-invariant component can be labeled yes *or* no *so that a* green *permissible $z \in Z^d$ belongs to $\Sigma(z_1, \ldots, z_d)$ iff it lies in a sign-invariant component labeled* yes.

Now we are ready to extend the proof of Theorem 5.7 to the extended model. For the rest of the proof, we simply confine ourselves to the green inputs. Note that the green z points form a sublattice of the original integer lattice. If we were to change our coordinate system so as to scale the green lattice by a factor of $1/2^{t(n,N)}$, then it would look like our original lattice. What is the effect of this scaling on the rest of the proof of Theorem 5.7? Well, the quantities μ and $\bar{\mu}$ that occur in section 5.3 have to be scaled by $1/2^{t(n,N)}$, but the remaining quantities such as ρ and δ are unaffected. If we choose the constant a' there large enough, $\log \mu$ and $\log \bar{\mu}$ are scaled down by at most the same constant factor, say, 2. But it can be checked that the rest of the proof in section 5.3 is robust with respect to a constant factor scaling of $\log \mu$ and $\log \bar{\mu}$. This proves Theorem 5.7 in the extended PRAM model with limited bit operations.

6.3. The number of iterations in sequential algorithms. Our lower bound for the mincost-flow and max-flow problems, which are instances of (combinatorial) linear programming, implies an interesting lower bound on the number of iterations in the sequential iterative algorithms for linear programming such as the ellipsoidal algorithm or the interior-point algorithm. These algorithms consist of a sequence of iterations, and each iteration can be accomplished—at least in theory—in $O(\log^2 n)$ parallel steps using a polynomial number of processors; this can be done using fast parallel algorithms for linear algebra [11] and approximate root extraction [3, 39]. It follows from Theorem 5.7 that given any parametrized family $\mathcal{I} = I(z)$ of inputs, these algorithms must require $\Omega(\sqrt{\log \rho(n)}/\log^2 n)$ iterations for some instance in the family; here $\rho(n)$ is defined for the family $\mathcal{I}$ as in Theorem 5.7. In the worst case—e.g., when $\mathcal{I}$ is a parametric family that attains maximum parametric complexity for the mincost-flow problem—this lower bound is $\Omega(n^{1/4}/\log^2 n)$. A similar lower bound applies to any iterative algebraic algorithm, deterministic or randomized. By an iterative algorithm, we mean an algorithm whose each iteration admits efficient parallelization in a theoretical sense. In practice, such an iteration may not be actually carried out in parallel, because parallel algorithms that are efficient in theory are not always practical. Rather, we think that parallelizability of a step indicates that it may be amenable to fast special-purpose algorithms. For example, fast algorithms for solving sparse linear systems seem to be useful in speeding up the iteration in the interior-point method. Hence, the number of iterations is an important criterion in practice.

Now suppose we are given an iterative algebraic algorithm, deterministic or randomized, whose each step can be efficiently parallelized in $\tau(n)$ parallel steps. It follows similarly that given any family $I(z)$ of inputs, this algorithm must require $\Omega(\sqrt{\log \rho(n)}/\tau(n))$ iterations for some instance in the family.

6.4. Weighted MAX-SNP problems. Once one has shown that one problem is hard to parallelize in the PRAM model without bit operations, it follows, as in the NP-completeness theory, that any problem that can be reduced to it fast in parallel in this model is also hard to parallelize. In this fashion one can show that several weighted MAX-SNP problems [41] are also hard to parallelize in the PRAM model without bit operations. For reasons stated in section 2.2 we consider only problems that have efficient sequential algorithms which do not manipulate bits (here this means strongly exponential time algorithms).

THEOREM 6.6. *The following problems cannot be solved in the PRAM model without bit operations (or with limited bit operations) deterministically (or with randomization) in $O(N^a)$ (expected) time using 2^{N^a} processors, where N denotes the input bitlength and a is a small enough positive constant. In all cases, the weights are*

meant to be nonnegative integers. (For the definitions of these problems, see [41].*)*

1. *Weighted* 3*-SAT-B, weighted* 3*-SAT-V, and weighted* 2*-SAT.*
2. *Weighted independent set-B.*
3. *Weighted vertex cover-B.*
4. *Weighted clique.*
5. *Weighted three-dimensional matching.*
6. *Weighted Steiner trees in graphs.*
7. *Weighted not-all-equal* 3*-SAT.*
8. *Weighted max-cut.*
9. *Traveling salesman.*
10. *Traveling salesman (metric).*

Proof. We have already shown that the weighted s-t-mincut problem is hard to parallelize in our model. It is easy to reduce one of the problems on the list, say, the weighted 3-SAT-B, to this problem, and then one can use standard reductions among weighted MAX-SNP-complete problems [41]. One has only to check that these reductions do not manipulate bits, i.e., they work fast in parallel in the PRAM model without bit operations. □

6.5. Higher order parametric complexity. The notion of parametric complexity as defined in section 3 can be readily generalized to allow any fixed number of parameters. Thus instead of a linear parameterization $\mathcal{P}$ in one rational parameter λ, as defined in section 3.1, one can consider a parameterization with any constant c number of rational parameters $\lambda_1, \ldots, \lambda_c$. The corresponding optimum function graph $G(\mathcal{P})$ should now be a convex c-dimensional simplicial complex. We define the complexity $\rho(n)$ of $\mathcal{P}$ to be the number of (bounded) facets of this complex. The bitsize of $\mathcal{P}$ is defined to be the maximum among the bitsizes of the coordinates of the vertices of $G(\mathcal{P})$ and of the coefficients of the linear functions in $\mathcal{P}$ specifying numeric parameters. We associate with $\mathcal{P}$ a $(c+1)$-dimensional homogeneous integral parameterization $\tilde{\mathcal{P}}$ very much as in section 3.1. Now given a large enough *permissibility constant* a, we can let $\mathcal{I}$ be the set of inputs of the form (z_{c+2}, I) in the decision version of our general optimization problem, where the following hold:

1. The integer z_{c+2} denotes the threshold and its bitsize is at most $a\beta(n)$.
2. I is of the form $\tilde{\mathcal{P}}(z_1, \ldots, z_{c+1})$ for some integers $z_1, \ldots, z_{c+1}$ of bitsize at most $a\beta(n)$.

Then the natural generalization of Theorem 5.7 to such parametric input families $\mathcal{I}$ holds. The proof generalizes because Collins' decomposition, the Milnor–Thom result, and the other techniques used in section 5.3 work in higher dimensions as well.

7. The P versus NC problem. Valiant [51] conjectured that over a field of characteristics other than two, the permanent of an $n \times n$ matrix cannot be expressed as a specialization—projection in his terminology—of the determinant of a matrix of poly(n) size. An evidence in support of this conjecture is #P-completeness of the permanent. Analogously we shall formulate for the mincost-flow problem a conjecture, which if true would imply that $P \neq NC$. We shall also prove its special case (Theorem 7.4). Evidence in support of the conjecture will be provided by this special case, our lower bound for the mincost-flow problem (Theorem 1.1) and the general belief that the perfect-matching problem for unweighted graphs does not belong to NC^1—it is not even known to belong to NC. The mincost-flow problem is chosen only for concreteness (see the discussion after the conjecture). In contrast to the setting in Valiant [51], which is purely algebraic, the issue of bitlengths is of paramount importance here.

Let C denote the set of complex numbers.

DEFINITION 7.1. *Given a hypersurface $H \in C^l$ and a linear map $F : C^k \to C^l$, $k \leq l$, let $F^{-1}(H)$ be the pullback (inverse image) of H under F.*

Here F can be nonhomogeneous, i.e., it can have constant additive terms in its defining forms.

We are mainly interested in the case when $l = m^2$ for some m and H is the determinant hypersurface $SL_m(C)$ in C^{m^2}. It is defined as follows. Let y_{ij}, $i, j \leq m$, denote the coordinates of C^{m^2}. Let Y denote the $m \times m$ matrix whose coefficients are the coordinates y_{ij}. Let $\det(Y)$ denote its determinant. Then $SL_m(C)$ is defined by the equation $\det(Y) = 1$; it is known to be a smooth hypersurface. In this case $F^{-1}(H)$ is defined by the equation $\det(F(x_1, \ldots, x_k)) = 1$, where $x_1, \ldots, x_k$ denote the coordinates of C^k; the image $F(x_1, \ldots, x_k)$ is an $m \times m$ matrix whose coefficients are linear forms in $x_1, \ldots, x_k$, possibly nonhomogeneous.

Now consider the decision version of the mincost-flow problem. Here we are given an n-vertex network whose each arc has integer cost and nonnegative integer capacity. Given an integer flow value v and an integer threshold w, the problem is to decide if there exists a flow with value v and cost at most w. The input can be identified with an integer tuple $I \in Z^k$ with $k = 2\binom{n}{2} + 2$ consisting of the costs, capacities, flow value, and threshold, where Z denotes the set of integers. Let $L(n) \subseteq Z^k$ denote the mincost-flow language, i.e., the set of tuples for which the answer is yes. For any positive constant a, let $B(a, n)$ denote the set of points in Z^k whose each coordinate has bitsize at most an, i.e., the integer points within the box defined by $|x_i| \leq 2^{an}$ for all $i \leq k$.

DEFINITION 7.2. *Given a hypersurface $H \subseteq C^l$ and a linear function $F : C^k \to C^l$, $l \geq k$, we say that the pullback $F^{-1}(H)$ separates $L(n)$ within $B(a, n)$ if the points in $B(a, n)$ that lie on it are precisely the ones in $L(n) \cap B(a, n)$.*

One can then make the following conjecture.

CONJECTURE. *Consider the hypersurface $H = SL_m(C)$, where $m \leq 2^{n/d}$ and $d > 0$ is a large enough constant ($d = 1$ may suffice). For any positive constant a and any one-to-one linear function $F : C^k \to C^{m^2}$, $k = 2\binom{n}{2} + 2$, the pullback $F^{-1}(H)$ cannot separate $L(n)$ within $B(a, n)$, assuming that n tends to infinity.*

The linear function F in the conjecture is over complex numbers. This is desirable because the methods of algebraic geometry work best over algebraically closed fields. We feel that if the conjecture is true when F ranges over some special integral functions—akin to Valiant's projections [51]—then it should also be true when F ranges over all complex linear functions. This will be borne out by Theorem 7.4, which holds over complex numbers.

We chose the mincost-flow problem in our formulation only for concreteness. One can choose any other problem that is not believed to be in NC, as long as it is suitable for an algebraic approach. If the problem is in P, it should have a strongly polynomial time algorithm. But it need not always be in P. For example, we can choose the permanent problem. This yields a conjecture, stronger than Valiant's [51], which if true would imply that the permanent does not belong to NC.

PROPOSITION 7.3. *If the conjecture is true for some positive $a < 1/(2d)$, then $P \neq NC$.*

Proof. The total bitlength of any tuple I in $B(a, n)$ is at most $kan = \text{poly}(n)$. Suppose, to the contrary, that $P = NC$. Then the mincost-flow problem belongs to NC. Hence there is a small arithmetic formula τ in the bits of I of size $2^{\text{polylog}(n)}$ such that the formula evaluates to one on such I in $B(a, n)$ iff it belongs to $L(n) \cap B(a, n)$.

(Here and in what follows we think of a bit as just an integer that is zero or one.) Each coordinate of I has bitsize at most an. Using Lagrange interpolation, one can construct for its every bit a formula in the coordinate of size $O(2^{2an})$: this formula evaluates to one iff that particular bit of the coordinate is one. Feeding these formulae for bits into the preceding formula τ, we get a formula of size $O(2^{2an+\mathrm{polylog}(n)})$ in the coordinates of I; this formula evaluates to one on $L(n) \cap B(a,n)$ but on no other point in $B(a,n)$. Using Valiant's reduction [51], one can now construct a matrix of almost the same size whose entries are linear forms in the coordinates of I and whose determinant is equal to the formula. This gives a one-to-one linear map $F : C^k \to C^{m^2}$, where $m = O(2^{2an+\mathrm{polylog}(n)})$ such that $F^{-1}[SL_m(C)]$ separates $L(n)$ within $B(a,n)$; moreover if $a < 1/(2d)$, this $m < 2^{n/d}$. This contradicts the conjecture. □

Here is the intuition behind the conjecture.

Our lower bound for the mincost-flow problem (Theorem 1.1) is undoubtedly far from optimal and holds only in the restricted PRAM model without bit operations. But for this problem our model is fairly realistic and the lower bound quite strong; it also holds when the bitsizes of the costs and capacities are restricted to be linear in n, the number of vertices. This suggests that in the presence of this restriction one probably cannot do much better than the following well-known parallel method, which we shall call the *explode-and-match* method.

Let n be the number of vertices in the flow network G, and let l be the maximum bitlength of a cost or capacity. The method first reduces the problem, by the well-known method [27] that "explodes" costs and capacities, to the maximum cardinality matching problem for an unweighted graph $\bar{G}$ of size $\bar{n} = O(2^l \mathrm{poly}(n))$. For the latter problem one can use a fast parallel algorithm in [24] or [36]. When $l = O(n)$, this method is better than the parallel algorithm of Shiloach and Vishkin [45] or of Goldberg and Tarjan [13] as far as the parallel time is concerned, though the processor count is very bad in comparison.

We are interested in the case when $l = an$ and a is the constant in the conjecture. Suppose the perfect-matching problem (and hence the maximum cardinality matching problem) for unweighted graphs belongs to NC^i for some i; so far this is open (it is known to be in RNC^2 [36]). Apply an NC^i-algorithm to the exploded graph $\bar{G}$. This gives a formula in the bits of the encoding of $\bar{G}$ which evaluates to one iff $\bar{G}$ has a matching of required cardinality; the size of this formula is $2^{c \log^i \bar{n}}$ for some constant c. In our case, $\bar{n} = O(2^l \mathrm{poly}(n)) = O(2^{an} \mathrm{poly}(n))$ with $l = an$. Hence the explode-and-match method gives us a formula in the bits of the edge-capacities and costs of G of size $2^{c(an)^i + O(\log^i n)}$ which evaluates to one iff the mincost-flow problem for the original graph G is feasible. Now one can feed into this formula the $O(4^{an})$-size formulae based on Lagrange interpolation for the bits of the costs and capacities (as in the proof of Proposition 7.3). This yields a formula of size

$$O(2^{c(an)^i + O(\log^i n)} 4^{an}) = O(2^{b(an)^i}),$$

where $b = c + 3$. Apply Valiant's reduction [51] to it. This gives a linear function $F : C^k \to C^{m^2}$, where $m = O(2^{b(an)^i})$. The pullback $F^{-1}[SL_m(C)]$ under this linear function separates $L(n)$ within $B(a,n)$.

If $i = 1$, i.e., if the perfect-matching problem belongs to NC^1, then one can choose a small enough so that the preceding $m < 2^{n/d}$. Then the conjecture, in fact, would be false for such small a. However, it is widely believed that the perfect-matching problem does not belong to NC^1; it is not even known to belong to NC.

Let us assume that $i > 1$. It need not be an integer. In other words, we are

even allowing a rather unlikely possibility that the perfect matching problem belongs to NC^i, where $i > 1$ is a real number arbitrarily close to one. In this case the quantity $2^{b(an)^i}$ above grows far faster than $2^{n/d}$ since i occurs in the exponent of the exponent. If we believe that when the costs and capacities have $O(n)$ bitlengths there is no parallel algorithm for the mincost-flow problem that is much better than the explode-and-match method—this is what our lower bound suggests—then, in turn, we are led to believe that one cannot come up with a separating linear function $F : C^k \to C^{m^2}$ with m that is much smaller than the $O(2^{b(an)^i})$ bound in the preceding scheme. But this bound grows much faster than $2^{n/d}$. That, roughly speaking, is the intuition behind the conjecture.

Remark. The belief that the perfect-matching problem does not belong to NC^1 is, in turn, tied to the belief that the determinant problem does not belong to NC^1 (it belongs to NC^2 [11]). This is because the parallel algorithm of Mulmuley, Vazirani, and Vazirani [36] reduces the matching problem to a calculation of determinants.

Another piece of evidence in support of the conjecture is the following.

THEOREM 7.4. *The conjecture is true for every large enough constant a (for the mincost-flow problem).*

Proof. This can be proved using Theorem 5.9.

Let a be a large enough positive constant to be chosen later. Assume that the conjecture is false. Then for some linear function $F : C^k \to C^{m^2}$, where $m \leq 2^{n/d}$, the inverse image $F^{-1}(SL_m(C))$ separates $L(n)$ within $B(a, n)$. Let $SL_m^+(C)$ and $SL_m^-(C)$ be hypersurfaces infinitesimally close to $SL_m(C)$ defined by the equations $\det(Y) = 1+\epsilon$ and $\det(Y) = 1-\epsilon$, respectively, where ϵ is infinitesimally small. Their inverse images—call them H_1 and H_2—partition $B(a, n)$ so that all points of $L(n)$ within it are contained in the same sign-invariant component which does not contain any other point of $B(a, n)$.

Now we shall confine our attention to only those integer points of $B(a, n)$ that lie within a three-dimensional affine subspace chosen as follows. We know that the parametric complexity $\phi(n, \beta(n))$ of the mincost-flow language is $2^{\Omega(n)}$ for $\beta(n) = O(n)$ (Theorem 3.5). Let $\mathcal{P}$ be a parameterization corresponding to the maximum value of the parametric complexity. Then the corresponding graph $G = G(\mathcal{P})$ (section 3.1) has $\rho = 2^{\Omega(n)}$ breakpoints. Let $\tilde{\mathcal{P}}$ be the corresponding homogeneous integral parameterization (section 5.2). Consider the integral parameterization of the input as defined before Theorem 5.8 by the map

$$E : (z_1, z_2, z_3) \to I(z_1, z_2, z_3) = (z_3, \tilde{\mathcal{P}}(z_1, z_2)).$$

We shall confine our attention to only those integer points in $B(a, n)$ that lie in the image of E, which is a three-dimensional integer lattice.

Let B be the region defined as in Theorem 5.9 by letting $\bar{\beta} = \log \bar{\mu} = an/2$; then the coordinates of all integer points in B have bitsizes at most an, which means $B \subseteq B(a, n)$. Let S be the set of surfaces $E^{-1}(H_1)$ and $E^{-1}(H_2)$. Their total degree δ is at most $2 \cdot 2^{n/d}$. Since $B \subseteq B(a, n)$, the partition of B formed by S has the separation property as defined in section 5.3. Choose a large enough so that $\bar{\beta} = \log \bar{\mu} = an/2$ is much larger than $\beta(n) = O(n)$ and d large enough so that $\log \rho = \Omega(n)$ is much larger than $\log \delta$, which is at most $n/d + \log 2$. Then it follows from Theorem 5.9 that this partition of B cannot have the separation property, a contradiction. □

Our technique cannot be used to prove this theorem for the permanent problem. It also cannot be used for proving the conjecture for all positive constants a. This is because all algebrogeometric results we used depend only on the low degree of the

surfaces that arise (actually the transversality results do not even depend on that). In particular, the same technique ends up proving Theorem 7.4 for any hypersurface $H \subseteq C^{m^2}$ of degree at most m (in place of $SL_m(C)$). This apparent strength of the technique is, in fact, its weakness, because of the following proposition.

PROPOSITION 7.5. *There exist hypersurfaces $H \subseteq C^{m^2}$, $m = 2^{n/d}$, of degree at most m for which the conjecture is false for $a < 1/(2d)$.*

Proof. Given an integer point $I \in B(a, n)$, let $\hat{I}$ denote its boolean encoding, which has length at most kan. Certainly there exists a (complicated) boolean formula of degree at most kan which evaluates to zero over $\hat{I}$ iff $I \in L(n)$. For any integer z of bitlength at most an, one can write down the Lagrange interpolation formula of degree at most 2^{2an} for each of its bits. Feeding such formulae into the preceding boolean formula, one gets a polynomial g which evaluates to zero on $I \in B(a, n)$ iff $I \in L(n)$. Its degree is at most $kan2^{2an} < 2^{n/d}$ if a is sufficiently smaller than $1/(2d)$. Let F be the trivial embedding of C^k in C^{m^2}, and let H be defined by the polynomial equation $g = 0$. □

This means if we wish to decrease the value of the constant a in Theorem 7.4 to something less than $1/(2d)$, as required in the conjecture, we must somehow use algebraic geometry of the *specific* hypersurface $SL_m(C)$ in the conjecture, or a similar one, in a deep way. What makes such hypersurfaces special is that they have a natural SL_m action. Such algebraic varieties have been investigated heavily in algebraic geometry and, more specifically, in geometric invariant theory [37] for over a century. One source of great difficulty in our context is that the natural SL_m action on such varieties need not carry over to their linear pullbacks. Therefore an important question here is, What is the algebraic geometry of linear pullbacks of such varieties? A good understanding of this, we feel, will greatly help in the P versus NC problem.

8. Conclusion. Our work shows that parametric complexity of a weighted optimization problem sets a good lower bound for its parallel complexity in the PRAM model without bit operations. In this way it may be possible to investigate parallel complexity of several other weighted optimization problems. This is especially important for problems that are neither known to be P-complete nor have fast parallel algorithms. A prime example is the minimum-weight perfect-matching problem. If the weights are in unary, it has fast parallel algorithms [24, 36]. In general, its parallel complexity is open. We conjecture that the parametric complexity of this problem is $2^{\Omega(n^\epsilon)}$ for some small enough positive ϵ. It would then follow from our general lower bound that it cannot be solved in our model in $o(N^{\epsilon'})$ time using $2^{o(N^{\epsilon'})}$ processors, where N is the input bitlength and ϵ' is a small enough positive constant. Several other problems may be investigated in this way, e.g., matroid intersection problems [28], matroid parity problems [28], construction of blocking flows [13], and problems in computational geometry.

Can one prove a good lower bound in our model for computing the permanent of an integer matrix? Valiant's result [50] leads one to expect that this problem should be even harder than the problems considered in this paper. And yet, paradoxically, it seems much harder to prove a good lower bound for the permanent. Csanky [11] shows that the determinant can be computed in our model in $O(\log^2 n)$ time using a polynomial number of processors. Can one show that it cannot be computed in our model in $O(\log n)$ time using a polynomial number of processors?

Most importantly, the role of algebrogeometric methods in computational complexity deserves to be studied further. Such methods have been useful in proving several combinatorial results; e.g., see [29, 33]. Can the same happen for a separation

question such as $P \neq NC$? Our work supports this possibility.

Acknowledgments. I wish to thank Allan Borodin, Steve Cook, Ajit Diwan, Charlie Rackoff, Donald Goldfarb, Laci Lovász, Katta Murty, Tushar Samant, L. V. Satyanarayana, and Milind Sohoni for illuminating discussions. I am also grateful to the referee for useful criticisms.

REFERENCES

[1] N. Alon and R. Boppana, *The monotone circuit complexity of boolean functions*, Combinatorica, 7 (1987), pp. 1–22.
[2] M. Ben-Or, *Lower bounds for algebraic computation trees*, in Proc. 15th ACM Symposium on Theory of Computing, 1983, pp. 80–86.
[3] M. Ben-Or, E. Feig, D. Kozen, and P. Tiwari, *A fast parallel algorithm for determining all roots of a polynomial with real roots*, SIAM J. Comput., 17 (1988), pp. 1081–1092.
[4] D. Bini and V. Pan, *Polynomial and Matrix Computations*, Birkhäuser Boston, Cambridge, MA, 1994.
[5] R. Boppana and M. Sipser, *The complexity of finite functions*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., North–Holland, Amsterdam, 1990, pp. 757–804.
[6] W. Bruns and U. Vetter, *Determinantal Rings*, Lecture Notes in Math., Springer, New York, 1988.
[7] P. Carstensen, *Complexity of some parametric integer and network programming problems*, Math. Programming, 26 (1983), pp. 64–75.
[8] P. Carstensen, *The Complexity of Some Problems in Parametric Linear and Combinatorial Programming*, Ph.D. dissertation, The University of Michigan, Ann Arbor, MI, 1982.
[9] N. Christofides, *Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem*, Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1976.
[10] G. Collins, *Quantifier elimination for real closed fields by cylindrical algebraic decomposition*, in Proc. 2nd GI Conf. on Automata Theory and Formal Languages, Lecture Notes in Comput. Sci. 33, Springer, Berlin, 1975, pp. 134–183.
[11] L. Csanky, *Fast parallel matrix inversion algorithms*, SIAM J. Comput., 5 (1976), pp. 618–623.
[12] M. Furst, J. Saxe, and M. Sipser, *Parity, circuits, and the polynomial time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.
[13] A. Goldberg and R. Tarjan, *A new approach to the maximum-flow problem*, J. ACM, 35 (1988), pp. 921–940.
[14] D. Goldfarb, *Worst-Case Complexity of the Shadow Vertex Simplex Algorithm*, Report, Department of Industrial Engineering and Operations Research, Columbia University, New York, 1983.
[15] L. Goldschlager, R. Shaw, and J. Staples, *The maximum flow problem is logspace complete for P*, Theoret. Comput. Sci., 21 (1982), pp. 105–111.
[16] M. Golubitsky and V. Guillemin, *Stable Mappings and Their Singularities*, Springer-Verlag, Berlin, 1973.
[17] M. Grotschel, L. Lovász, and A. Schriver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.
[18] D. Gusfield, *Sensitivity Analysis for Combinatorial Optimization*, Memorandum UCB/ERL M80/22, Electronics Research Laboratory, Berkeley, CA, 1980.
[19] J. Håstad, *Almost optimal lower bounds for small depth circuits*, in Advances in Computing Research 5: Randomness and Computation, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 143–170.
[20] O. Ibarra and C. Kim, *Approximation algorithms for certain scheduling problems*, Math. Oper. Res., 3 (1978), pp. 197–204.
[21] O. Ibarra, S. Moran, and L. Rosier, *A note on parallel complexity of computing the rank of order n matrices*, Inform. Process. Lett., 11 (1980), p. 162.
[22] N. Karmarkar, *A new polynomial-time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.
[23] R. Karp and V. Ramachandran, *A survey of parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., Elsevier Science Publishers, New York, 1990, pp. 870–941.
[24] R. Karp, E. Upfal, and A. Wigderson, *Constructing a perfect matching is in random NC*, Combinatorica, 6 (1986), pp. 35–48.

[25] L. Khachian, *Polynomial algorithms in linear programming*, U.S.S.R. Computational Mathematics and Mathematical Physics, 20 (1980), pp. 53–72 (English translation).
[26] V. Klee and G. Minty, *How good is the simplex algorithm?*, in Inequalities, III, O. Shisha, ed., Academic Press, New York, 1972, pp. 159–175.
[27] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.
[28] L. Lovász and M. Plummer, *Matching Theory*, Akadémiai Kiadó, Budapest, 1986.
[29] A. Lubotzky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica, 8 (1988), pp. 261–277.
[30] D. Karger, *Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm*, in Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms, 1993, pp. 21–30.
[31] D. Karger and R. Motwani, *Derandomization through approximation: An NC algorithm for minimum cuts*, in Proc. 26th ACM Symposium on Theory of Computing, 1994, pp. 497–506.
[32] T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kauffman, 1992.
[33] G. Margulis, *Arithmetic groups and graphs without short cycles*, in Proc. 6th International Symposium on Inform. Theory, Tashkent 1984, Abstracts, Vol. I, pp. 123–125.
[34] J. Milnor, *On the betti numbers of real varieties*, in Proc. American Mathematical Society 15, 1964, pp. 275–280.
[35] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice–Hall, New York, 1993.
[36] K. Mulmuley, U. Vazirani, and V. Vazirani, *Matching is as easy as matrix inversion*, Combinatorica, 7 (1987), pp. 105–113.
[37] D. Mumford and F. Fogarty, *Geometric Invariant Theory*, 2nd ed., Springer, New York, Berlin, Heidelberg, 1982.
[38] K. Murty, *Computational complexity of parametric linear programming*, Math. Programming, 19 (1980), pp. 213–219.
[39] C. Neff, *Specified precision polynomial root isolation is in NC*, in Proc. 31st IEEE Symposium on Foundations of Computer Science, 1990, pp. 152–162.
[40] V. Pan and J. Reif, *Efficient parallel solution of linear systems*, in Proc. 17th ACM Symposium on Theory of Computing, 1985, pp. 143–152.
[41] C. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, JCSS, 43 (1991), pp. 425–440.
[42] A. Razborov, *Lower bounds on the monotone complexity of some boolean functions*, Dokl. Akad. Nauk, (1985), pp. 798–801.
[43] J. Reif, ed., *Synthesis of Parallel Algorithms*, Morgan Kaufmann, 1993.
[44] A. Razborov and S. Rudich, *Natural proofs*, in Proc. 26th ACM Symposium on Theory of Computing, 1994, pp. 204–213.
[45] Y. Shiloach and U. Vishkin, *An $O(n^2 \log n)$ parallel max-flow algorithm*, J. Algorithms, 3 (1982), pp. 128–146.
[46] J. Steele and A. Yao, *Lower bounds for algebraic decision trees*, J. Algorithms, 3 (1982), pp. 1–8.
[47] E. Tardos, *A strongly polynomial algorithm to solve combinatorial linear programs*, Oper. Res., 34 (1986), pp. 250–256.
[48] R. Tarjan, *Algorithms for maximum network flow*, Mathematical Programming Study, 26 (1986), pp. 1–11.
[49] R. Thom, *Sur l'homologie des varietes algebriques reelles*, Differential and Combinatorial Topology, S. Cairns, ed., Princeton University Press, Princeton, NJ, 1965.
[50] L. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.
[51] L. Valiant, *Completeness classes in algebra*, in Proc. ACM Symposium on Theory of Computing, 1979, pp. 249–261.
[52] A. Yao, *Separating the polynomial-time hierarchy by oracles*, in Proc. 26th IEEE Symposium on Foundations of Computer Science, 1985, pp. 1–10.
[53] A. Yao, *Lower bounds for algebraic computation trees with integer inputs*, in Proc. 30th IEEE Symposium on Foundations of Computer Science, 1989, pp. 1–10.
[54] A. Yao, *Probabilistic computation: Toward a unified measure of complexity*, in Proc. 18th IEEE Symposium on Foundations of Computer Science, 1977, pp. 222–227.

[55] L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, *Fast parallel computation of polynomials using few processors*, SIAM J. Comput., 12 (1983), pp. 641-644.
[56] A. Wigderson, *The fusion method for lower bounds in circuit complexity*, Bolyai Soc. Math. Stud., (1993), pp. 453–467.
[57] N. Zadeh, *A bad network problem for the simplex method and other minimum cost flow algorithms*, Math. Programming, 5 (1973), pp. 255–266.

SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1510–1539

STACK AND QUEUE LAYOUTS OF DIRECTED ACYCLIC GRAPHS: PART I*

LENWOOD S. HEATH†, SRIRAM V. PEMMARAJU‡, AND ANN N. TRENK§

Abstract. Stack layouts and queue layouts of undirected graphs have been used to model problems in fault-tolerant computing and in parallel process scheduling. However, problems in parallel process scheduling are more accurately modeled by stack and queue layouts of *directed acyclic graphs* (dags). A *stack layout* of a dag is similar to a stack layout of an undirected graph, with the additional requirement that the nodes of the dag be in some topological order. A *queue layout* is defined in an analogous manner. The *stacknumber* (*queuenumber*) of a dag is the smallest number of stacks (queues) required for its stack layout (queue layout). In this paper, bounds are established on the stacknumber and queuenumber of two classes of dags: tree dags and unicyclic dags. In particular, any tree dag can be laid out in 1 stack and in at most 2 queues; and any unicyclic dag can be laid out in at most 2 stacks and in at most 2 queues. Forbidden subgraph characterizations of 1-queue tree dags and 1-queue cycle dags are also presented. Part II of this paper presents algorithmic results—in particular, linear time algorithms for recognizing 1-stack dags and 1-queue dags and proof of NP-completeness for the problem of recognizing a 4-queue dag and the problem of recognizing a 9-stack dag.

Key words. stack layout, queue layout, book embedding, graph embedding, directed acyclic graphs, dags, forbidden subgraph

AMS subject classifications. 05C99, 68Q15, 68Q25, 68R10, 94C15

PII. S0097539795280287

Stack layouts and queue layouts of undirected graphs have appeared in a variety of contexts such as VLSI design, fault-tolerant processing, parallel process scheduling, sorting networks, and parallel matrix computations [3, 4, 7, 8]. Bernhart and Kainen [1] introduce the concept of stack layouts under the name *book embeddings*. Motivated by problems in fault-tolerant processing, Chung, Leighton, and Rosenberg [3] examine stack layouts of undirected graphs and construct optimal stack layouts for a variety of classes of graphs. Motivated by problems in parallel process scheduling, Heath, Leighton, and Rosenberg [4, 8] develop the notion of queue layouts and provide optimal queue layouts for many classes of undirected graphs. However, problems in parallel process scheduling are more accurately modeled by stack and queue layouts of directed acyclic graphs (dags). Bhatt et al. [2] provide an example of a control-memory tradeoff in parallel process scheduling, obtained by examining queue layouts of binary trees. Stack and queue layouts of dags are also closely related to stack and queue layouts of partially ordered sets (posets). Nowakowski and Parker [9] and Syslo [10] initiate the study of stack layouts of posets; Heath and Pemmaraju [6] extend the study to queue layouts of posets.

*Received by the editors January 23, 1995; accepted for publication (in revised form) May 7, 1997; published electronically April 27, 1999.

http://www.siam.org/journals/sicomp/28-4/28028.html

†Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061-0106 (heath@cs.vt.edu). This research was supported in part by National Science Foundation grant CCR-9009953.

‡Department of Computer Science, University of Iowa, Iowa City, IA 52242-1316 (sriram@cs.uiowa.edu). This research was supported in part by National Science Foundation grant CCR-9009953.

§Department of Mathematics, Wellesley College, Wellesley, MA 02181 (atrenk@wellesley.edu). This research was supported by an Eliezer Naddor Postdoctoral Fellowship in Mathematical Sciences from The Johns Hopkins University during the year 1991–1992 while in residence at Dartmouth College.

In this paper, we define stack and queue layouts of dags and develop combinatorial results for stack and queue layouts of some special classes of dags. Each of these classes arises from some property of the undirected graph underlying a dag. In particular, we consider the property of the underlying undirected graph being a path, a cycle, a tree, or a unicyclic graph. We also give forbidden subgraph characterizations of 1-queue tree dags and 1-queue cycle dags. In the companion paper [5], we develop algorithmic results for stack and queue layouts of dags. In particular, we show that 1-stack and 1-queue dags can be recognized in linear time, while the problems of recognizing 9-stack dags and 4-queue dags are both NP-complete.

The organization of this paper is as follows. Section 1 contains definitions, notations, and an initial discussion of 1-queue dags. In section 2, we examine stack layouts of tree dags and unicyclic dags. In section 3, we examine the queue layouts of tree dags and unicyclic dags. In section 4, we present a forbidden graph characterization of 1-queue tree dags and 1-queue cycle dags. Section 5 contains concluding remarks and a conjecture.

1. Preliminaries. Following a common distinction, we use the terminology *edge* for undirected graphs, and the terminology *arc* for directed graphs; we use the terminology *node* for both undirected and directed graphs. The definitions relevant to stack and queue layouts of undirected graphs are found in Heath and Rosenberg [8] and are reproduced here. Throughout these definitions, the concept of a total order σ on a set V of nodes is central; the notation $u <_\sigma v$, where $u, v \in V$, is used consistently to emphasize the particular total order currently under consideration.

Let $G = (V, E)$ be an undirected graph without multiple edges or loops. A *k-stack layout* of G consists of a total order σ on V along with an assignment of each edge in E to one of k stacks, $s_1, s_2, \ldots, s_k$. Each stack s_j operates as follows. The nodes of V are scanned in left-to-right (ascending) order according to σ. When a node v is encountered, any edges assigned to s_j that have v as their right endpoint must be at the top of the stack and are popped. Any edges that are assigned to s_j and have left endpoint v are pushed onto s_j in descending order (according to σ) of their right endpoints. The *stacknumber* $SN(G)$ of G is the smallest k such that G has a *k-stack layout*. G is said to be a *k-stack* graph if $SN(G) = k$.

A *k-queue layout* of G consists of a total order σ on V along with an assignment of each edge in E to one of k queues, $q_1, q_2, \ldots, q_k$. Each queue q_j operates as follows. The nodes of V are scanned in left-to-right (ascending) order according to σ. When a node v is encountered, any edges assigned to q_j that have v as their right endpoint must be at the front of the queue and are dequeued. Any edges that are assigned to q_j and have left endpoint v are enqueued into q_j in ascending order (according to σ) of their right endpoints. The *queuenumber* $QN(G)$ of G is the smallest k such that G has a *k-queue layout*. G is said to be a *k-queue* graph if $QN(G) = k$.

We now consider directed graphs. Let $G = (V, E)$ be an undirected graph, and let $\vec{G} = (V, \vec{E})$[1] be any dag obtained from G by directing the edges in G. More precisely, each edge $\{u, v\} \in E$ is replaced by either arc (u, v) or by arc (v, u) in $\vec{E}$, and the resulting directed graph $\vec{G} = (V, \vec{E})$ is a dag, i.e., there is no induced directed cycle in $\vec{G}$. G is the *covering graph* of the dag $\vec{G}$; the covering graph of an arbitrary dag is obtained by removing the direction from all the arcs. A dag is *connected* if its covering graph is connected. If $(u, v) \in \vec{E}$, then u is an *in-neighbor* of v and v is an

[1] The $\vec{}$ notation always distinguishes a directed graph from an undirected graph or a set of arcs from a set of edges.

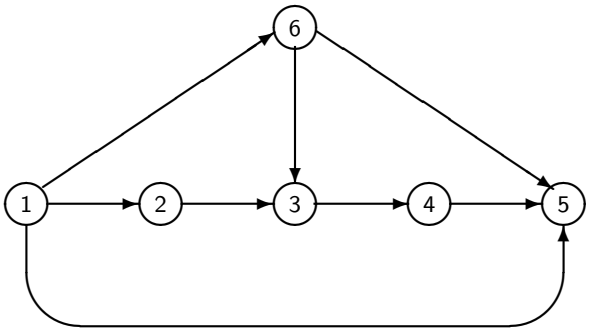


FIG. 1.1. *A* 2*-stack,* 2*-queue dag.*

out-neighbor of u. The *indegree* of v is its number of in-neighbors, while the *outdegree* of v is its number of out-neighbors. Two arcs, (u_1, v_1) and (u_2, v_2), are *independent* if the four nodes u_1, u_2, v_1, and v_2 are distinct. A *topological order* of $\vec{G} = (V, \vec{E})$ is a total order σ on V such that $(u, v) \in \vec{E}$ implies $u <_\sigma v$.

We are particularly interested in some special classes of dags. A *path dag* $\vec{G} = (V, \vec{E})$ is a dag whose covering graph $G = (V, E)$ is a path. Similarly, a *cycle dag* is a dag whose covering graph is a cycle, a *tree dag* is a dag whose covering graph is a tree, and a *unicyclic dag* is a dag whose covering graph contains a single cycle. A tree dag is *rooted* by selecting an arbitrary node to be its *root.* Note that a rooted tree dag does not correspond to the normal notion of a rooted tree, in which all arcs are uniformly directed away from (or toward) the root. Still, we wish the notions of parent and subtree to carry over to rooted tree dags. If a tree dag $\vec{T} = (V, \vec{E})$ is rooted at a node r, then every node v other than r has a *parent* that is its parent in the covering graph $T = (V, E)$ considered as a tree rooted at r. Note that the parent of a node may be either an in-neighbor or an out-neighbor. In a rooted tree dag $\vec{T} = (V, \vec{E})$, the *subtree* $\vec{T}_v$ rooted at v is the subdag of $\vec{T}$ corresponding to the subtree of T rooted at v.

The definition of a stack or a queue layout is now extended to dags. The key distinction is that such a layout is required to follow a topological order. A *k-stack (k-queue) layout of a dag* $\vec{G} = (V, \vec{E})$ is a k-stack (k-queue) layout of the covering graph G such that the total order used in the layout is a topological order of $\vec{G}$. As before, the *stacknumber* $SN(\vec{G})$ is the smallest k such that $\vec{G}$ has a k-stack layout, and the *queuenumber* $QN(\vec{G})$ is the smallest k such that $\vec{G}$ has a k-queue layout. If $SN(\vec{G}) = k$, then $\vec{G}$ is called a *k-stack dag*, and if $QN(\vec{G}) = k$, then $\vec{G}$ is called a *k-queue dag.* As an example, Figure 1.1 shows a dag, Figure 1.2 shows a 2-stack layout, and Figure 1.3 shows a 2-queue layout. In each of the layouts shown in Figures 1.2 and 1.3, the arcs above the line of nodes are assigned to one stack or queue, while the arcs below the line of nodes are assigned to the other stack or queue. The reader may check that the dag in Figure 1.1 has neither a 1-stack nor a 1-queue layout.

Suppose that $\vec{G} = (V, \vec{E})$ is a dag and that σ is a topological order of $\vec{G}$. Let $(u_1, v_1), (u_2, v_2) \in \vec{E}$ be distinct arcs with $u_1 <_\sigma u_2$. If $v_1 <_\sigma v_2$, then (u_1, v_1) and (u_2, v_2) *cross* in σ. If $v_2 <_\sigma v_1$, then (u_1, v_1) and (u_2, v_2) *nest* in σ. The definition of a stack layout of a dag requires that any two arcs that cross must be in different stacks; on the other hand, any two arcs that do not cross may be placed in the same stack. In Figure 1.2, arcs $(2, 3)$ and $(6, 5)$ cross; hence they must be in different stacks.

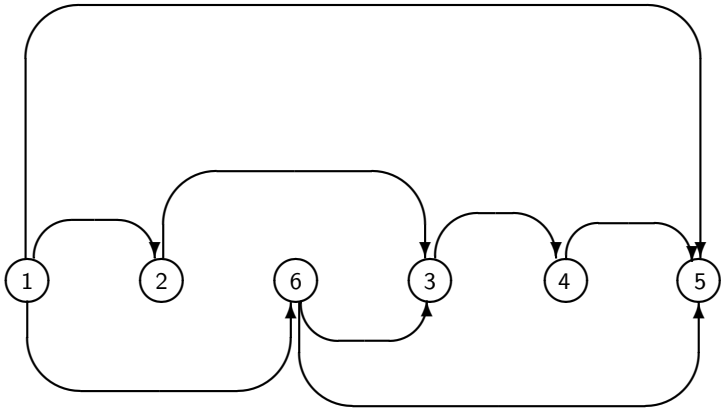


FIG. 1.2. *A 2-stack layout of the dag in Figure* 1.1.

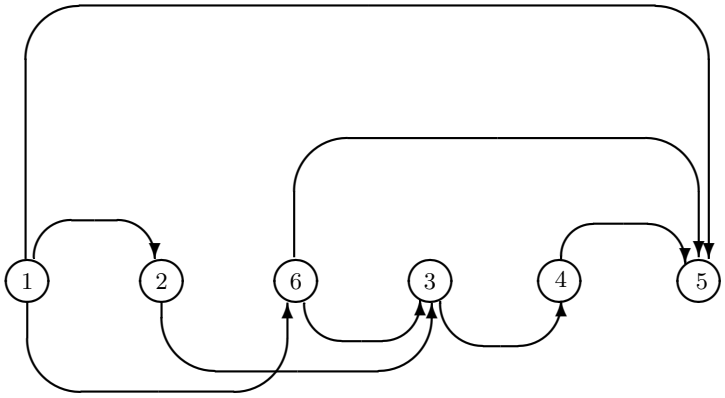


FIG. 1.3. *A 2-queue layout of the dag in Figure* 1.1.

Analogously, the definition of a queue layout of a dag requires that any two arcs that nest must be in different queues; on the other hand, any two arcs that do not nest may be placed in the same queue. In Figure 1.3, arcs $(1, 5)$ and $(2, 3)$ nest; hence they must be in different queues.

Heath and Rosenberg [8] characterize 1-queue undirected graphs as those graphs that have an arched leveled-planar embedding. In a similar manner, we characterize 1-queue dags as those that have a *directed arched leveled-planar embedding.* The definition of such an embedding is developed in the next five paragraphs.

Let $\vec{G} = (V, \vec{E})$ be a dag. A *leveling* of $\vec{G}$ is a function $lev : V \to \mathbf{Z}$ mapping the nodes of $\vec{G}$ to integers such that $lev(v) = lev(u) + 1$ for all $(u, v) \in \vec{E}$. If $lev(v) = j$, then v is a *level-j node.* $\vec{G}$ is a *leveled dag* if it has a leveling. Note that a connected leveled dag has a unique leveling up to an additive constant.

Let ℓ_j denote the vertical line in the Cartesian plane $\ell_j = \{(j, y) \mid y \in \mathbf{R}\}$, where $\mathbf{R}$ is the set of reals. Suppose that there is a partition of the set of nodes V into sets $V_p, V_{p+1}, \ldots, V_q$ for some integers p and q such that:

1. Each arc in $\vec{E}$ is from a node in V_j to a node in V_{j+1} for some integer j, $p \leq j < q$.

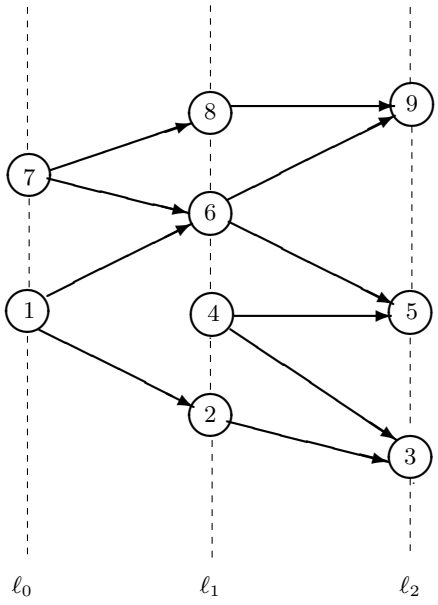


FIG. 1.4. *A leveled-planar dag.*

2. $\vec{G}$ has a planar embedding in which all nodes in V_j are placed on ℓ_j and each arc in $\vec{E}$ is drawn as a straight line segment between lines ℓ_j and ℓ_{j+1} for some integer j, $p \le j < q$; in particular, the drawings of two independent arcs do not intersect in the plane.

Then this planar embedding is called a *directed leveled-planar embedding* of $\vec{G}$. Figure 1.4 shows a directed leveled-planar embedding of a dag. A dag is called a *leveled-planar dag* if it has a directed leveled-planar embedding.

Now suppose that $\vec{G}$ is a leveled-planar dag and that $\mathcal{E}$ is a directed leveled-planar embedding of $\vec{G}$. The function $lev : V \to \mathbf{Z}$, where $lev(u) = j$ if $u \in V_j$, is the *leveling of $\vec{G}$ induced by the embedding $\mathcal{E}$*. The leveling induced by the directed leveled-planar embedding shown in Figure 1.4 is $lev(1) = lev(7) = 0$, $lev(2) = lev(4) = lev(6) = lev(8) = 1$, and $lev(3) = lev(5) = lev(9) = 2$. Thus every leveled-planar dag is a leveled dag. It is easy to see that the converse is not true (e.g., consider a 4-cycle with arcs alternating in direction).

A directed leveled-planar embedding of $\vec{G}$ induces a natural topological order of $\vec{G}$ called the *leveled-planar order,* obtained by scanning the lines $\ell_p, \ell_{p+1}, \ldots, \ell_q$, in that order, each from bottom to top. The leveled-planar order induced by the directed leveled-planar embedding of Figure 1.4 is $1, 7, 2, 4, 6, 8, 3, 5, 9$.

A directed arched leveled-planar embedding is obtained from a directed leveled-planar embedding as follows. Let σ be the leveled-planar order. For each j, $p \le j \le q$, let b_j be the *bottom node* on line ℓ_j, and let t_j be the *top node*. Let s_j be the bottommost node on line ℓ_j that is the in-neighbor of some node on line ℓ_{j+1}; if there are no arcs from V_j to V_{j+1}, then let $s_j = t_j$. For an integer j, $p \le j \le q$, a *level-j arch* is an arc from a node $x \in V_j$, where $b_j \le_\sigma x \le_\sigma s_j$ and $x <_\sigma t_j$, to node t_j. A directed leveled-planar dag augmented by any number of arches can be drawn in the plane by drawing the arches around ℓ_p; such an embedding is called a *directed arched leveled-planar embedding*. A dag that has a directed arched leveled-planar embedding

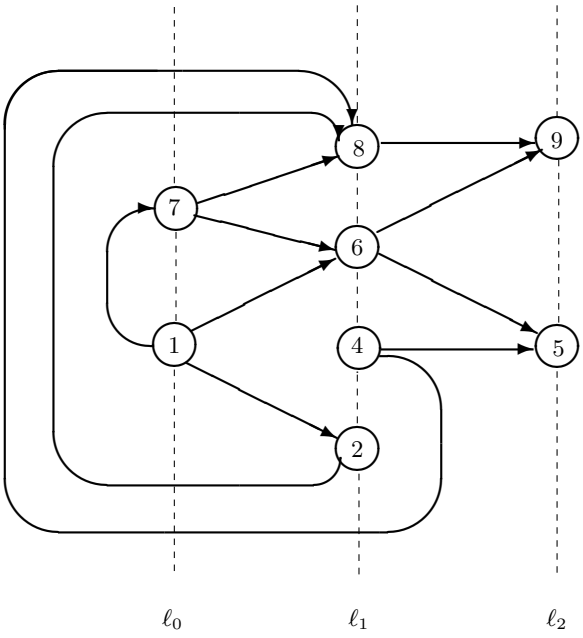


FIG. 1.5. *An arched leveled-planar dag.*

is called an *arched leveled-planar dag*. The arcs in the embedding that are not arches are called *level arcs*. Figure 1.5 shows a directed arched leveled-planar embedding of a dag.

The figure is the same as Figure 1.4 except that node 3 is deleted and arches $(1, 7)$, $(2, 8)$, and $(4, 8)$ are added.

We now state the characterization of 1-queue dags.

PROPOSITION 1.1. *Suppose $\vec{G}$ is a dag. Then $QN(\vec{G}) = 1$ if and only if $\vec{G}$ is an arched leveled-planar dag.*

The proof of this characterization is very similar to the proof that Heath and Rosenberg [8] use in establishing their characterization of 1-queue undirected graphs as exactly those graphs that have an arched leveled-planar embedding (see Theorem 3.2 in [8]).

2. Stack layouts of dags. In this section, we investigate stack layouts of tree dags and unicyclic dags. We need one definition. A node $v \in V$ is *exposed* in a layout with total order σ if there is no arc (x, y) in the layout with $x <_\sigma v <_\sigma y$.

We begin with the following theorem on the stacknumber of tree dags.

THEOREM 2.1. *The stacknumber of any tree dag with at least 2 nodes is 1.*

Proof. Let $\vec{T} = (V, \vec{E})$ be an arbitrary tree dag with $N \geq 1$ nodes. Root $\vec{T}$ at an arbitrary node r. We prove by induction on N that $\vec{T}$ can be laid out in at most 1 stack with the root r *exposed*.

Base case: When $N = 1$, the claim is trivially true.

Induction case: Suppose that $N \geq 2$ and that every tree dag with fewer than N nodes can be laid out in at most 1 stack with its root exposed. Let $u_1, u_2, \ldots, u_s$ be the in-neighbors of r, and let $v_1, v_2, \ldots, v_m$ be the out-neighbors of r. The induction hypothesis implies that each of the tree dags $\vec{T}_{u_p}$, where $1 \leq p \leq s$, and $\vec{T}_{v_q}$, where $1 \leq q \leq m$, can be laid out in one stack with their roots $u_1, u_2, \ldots, u_s, v_1, v_2, \ldots, v_m$

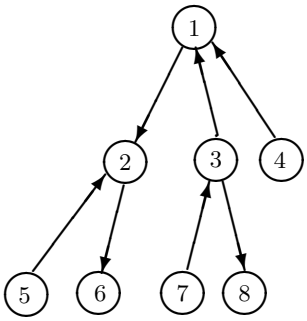


FIG. 2.1. *A tree dag.*

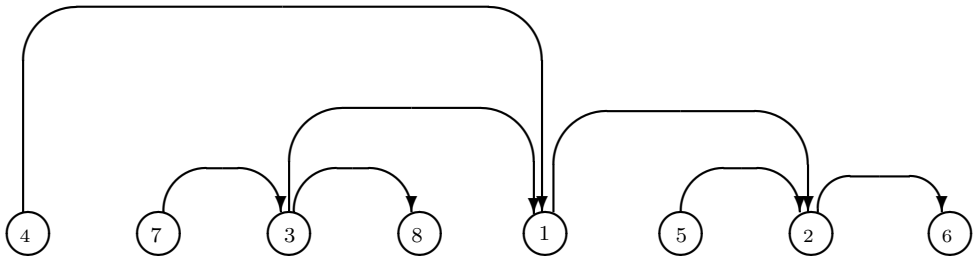


FIG. 2.2. *A 1-stack layout of the tree dag shown in the Figure* 2.1.

exposed. Concatenate these layouts, together with r, in this order:

$$\vec{T}_{u_1}, \vec{T}_{u_2}, \ldots, \vec{T}_{u_s}, r, \vec{T}_{v_1}, \vec{T}_{v_2}, \ldots, \vec{T}_{v_m}.$$

Add the arcs from $u_1, u_2, \ldots, u_s$ to r and from r to $v_1, v_2, \ldots, v_m$. It is clear that the nodes have been laid out in topological order, no two arcs in $\vec{T}$ cross, and r is exposed in the layout so obtained. A 1-stack layout is obtained.

By induction, every tree dag can be laid out in at most 1 stack. Since a tree dag with at least 2 nodes (at least 1 arc) requires at least 1 stack, the theorem follows. □

As an example, Figure 2.1 shows a tree dag, while Figure 2.2 shows the 1-stack layout constructed in the proof of the theorem, where node 1 is the root.

We now turn our attention to unicyclic dags. Not all unicyclic dags can be laid out in 1 stack. Figure 2.3 shows the smallest example of a unicyclic dag whose stacknumber is not 1.

In the remainder of this section, we show that any unicyclic dag can be laid out in 2 stacks and provide a simple characterization of 1-stack unicyclic dags. We first investigate stack layouts of cycle dags. A *directed Hamiltonian path* in a dag $\vec{G} = (V, \vec{E})$ is a directed path in $\vec{G}$ that contains every node in V. The following lemma shows that any cycle dag can be laid out in 2 stacks and that for each n there is exactly one cycle dag on n nodes with stacknumber 1.

LEMMA 2.2. *Let $\vec{C} = (V, \vec{E})$ be a cycle dag. If $\vec{C}$ contains a directed Hamiltonian path, then $SN(\vec{C}) = 1$; otherwise, $SN(\vec{C}) = 2$.*

Proof. First suppose that $\vec{C}$ contains a directed Hamiltonian path. This Hamiltonian path defines a unique topological order of $\vec{C}$, and no two arcs of $\vec{C}$ cross when the nodes of $\vec{C}$ are laid out according to this order. Hence, $SN(\vec{C}) = 1$, as desired.

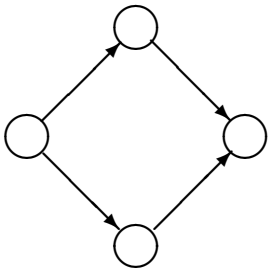


FIG. 2.3. *A unicyclic dag that cannot be laid out in* 1 *stack.*

Now suppose that $\vec{C}$ does not contain a directed Hamiltonian path. We first show that $SN(\vec{C}) \geq 2$ in this case. To obtain a contradiction, suppose that $SN(\vec{C}) = 1$. Let σ be a topological order that yields a 1-stack layout of $\vec{C}$. For any pair of nodes u and v in V such that $u <_\sigma v$, say that u and v are *consecutive nodes* in σ if there exists no w such that $u <_\sigma w <_\sigma v$. Since $\vec{C}$ has no directed Hamiltonian path, there exists a pair of nodes u and v that are consecutive nodes in σ satisfying the properties that $u <_\sigma v$ and $(u, v) \notin \vec{E}$.

Let the cycle $C = (V, E)$ be the covering graph of $\vec{C}$. Of course, the order σ yields a 1-stack layout of C. Let $X = \{w \in V \mid w \leq_\sigma u\}$ be the set of nodes that are no greater than u with respect to σ. Similarly, let $Y = \{w \in V \mid v \leq_\sigma w\}$ be the set of nodes that are no less than v with respect to σ. Note that $V = X \cup Y$. Let $x \in X$ be the greatest node with respect to σ that is adjacent to a node in Y. Of all the nodes in Y that are adjacent to x, let y be the least with respect to σ. If $x = u$ and $y = v$, then $(u, v) = (x, y) \in E$, contradicting the choice of u and v. Hence, either $x <_\sigma u$ or $v <_\sigma y$ (or both). We consider only the case $x <_\sigma u$, the other case being symmetric.

Let $Z = \{w \mid x <_\sigma w \leq_\sigma u\}$ consist of the nodes between x and u with respect to σ together with u. Because of our choice of the edge (x, y) in C, no node in Z has a neighbor in $W = \{w \mid v \leq_\sigma w \leq_\sigma y\}$. Because no two edges of C cross in σ and because (x, y) is an edge in E, any neighbor in C of a node in Z is in $Z \cup \{x\}$. Thus, every path from a node in Z to a node not in Z passes through x. Since $u \in Z$ and $v \notin Z \cup \{x\}$, every path between these two nodes passes through x, implying that x is a cutpoint of C. This is a contradiction because a cycle cannot have a cutpoint. Hence, the assumption that $SN(\vec{C}) = 1$ must be false. It follows that $SN(\vec{C}) \geq 2$.

To show that $SN(\vec{C}) = 2$, we construct a 2-stack layout of any cycle dag $\vec{C}$ as follows. Let u be a node in $\vec{C}$ with indegree 0, and let v and w be the two out-neighbors of u. Delete the node u from $\vec{C}$ and let $\vec{T}$ be the resulting tree dag. By Theorem 2.1, there is a topological order σ of $\vec{T}$ that yields a 1-stack layout of $\vec{T}$. To obtain a 2-stack layout of $\vec{C}$ from this, extend σ to satisfy $u <_\sigma v$, for all nodes v of $\vec{T}$, and assign the two arcs incident on u to a second stack. □

We now establish the stacknumber of unicyclic dags by showing that a unicyclic dag requires as many stacks as the cycle dag it contains.

THEOREM 2.3. *Let $\vec{U} = (V, \vec{E})$ be a unicyclic dag, and let $\vec{C} = (V_C, \vec{E}_C)$ be the unique cycle dag that is a subgraph of $\vec{U}$. Then $SN(\vec{U}) = SN(\vec{C})$.*

Proof. Clearly, $SN(\vec{U}) \geq SN(\vec{C})$. Let $SN(\vec{C}) = k$. By Lemma 2.2, either $k = 1$ or $k = 2$. Without loss of generality, we may assume that $v_1, v_2, \ldots, v_n$ is a topological order of $\vec{C}$ that yields a k-stack layout. By deleting all the arcs in $\vec{C}$ from $\vec{U}$ we obtain n connected components, each a tree dag. Let $\vec{T}_i$ denote the tree dag containing v_i.

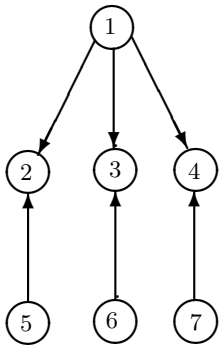


FIG. 3.1. *A tree dag that cannot be laid out in* 1 *queue.*

By the proof of Theorem 2.1, each tree dag $\vec{T}_i$ has a 1-stack layout in which node v_i is exposed. To obtain a 1-stack layout of $\vec{U}$, concatenate the 1-stack layouts of the k tree dags in the order $\vec{T}_1, \vec{T}_2, \ldots, \vec{T}_n$. Since the tree dags are laid out one after the other, no two arcs belonging to distinct tree dags cross. Since each node v_i is exposed, the arcs in the tree dags do not cross the arcs in $\vec{C}$. Hence the arcs in $\vec{T}_i$, for all i, can be assigned to one of the k stacks being used for the arcs in $\vec{C}$. We conclude that $SN(\vec{U}) = SN(\vec{C})$, as desired. □

3. Queue layouts of dags. In this section, we determine the queuenumber of tree dags and of unicyclic dags. We start with tree dags. It is easily verified that the tree dag shown in Figure 3.1 does not have a directed arched leveled-planar embedding and is therefore not a 1-queue dag. However, as shown in the following theorem, 2 queues suffice for any tree dag.

THEOREM 3.1. *Every tree dag has a* 2*-queue layout.*

Proof. Let $\vec{T} = (V, \vec{E})$ be a tree dag with covering graph $T = (V, E)$. Root $\vec{T}$ at an arbitrary node r. Partition $\vec{E}$ into sets $\vec{E}_b$ and $\vec{E}_f$, where

$$\vec{E}_f = \{(u, v) \in \vec{E} \mid u \text{ is the parent of } v \text{ in } \vec{T}\},$$
$$\vec{E}_b = \{(v, u) \in \vec{E} \mid u \text{ is the parent of } v \text{ in } \vec{T}\}.$$

Thus $\vec{E}_f$ contains the *forward arcs*, arcs directed away from r, while $\vec{E}_b$ contains the *backward arcs*, arcs directed towards r. For example, if the root of the tree dag in Figure 3.1 is 1, then $\vec{E}_f = \{(1,2), (1,3), (1,4)\}$ and $\vec{E}_b = \{(5,2), (6,3), (7,4)\}$. Let $\vec{G}_f = (V, \vec{E}_f)$ and $\vec{G}_b = (V, \vec{E}_b)$ be the subdags of $\vec{T}$ induced by the arc sets $\vec{E}_f$ and $\vec{E}_b$, respectively. We construct a 2-queue layout of $\vec{T}$ by placing the nodes of $\vec{T}$ in the Cartesian plane such that directed leveled-planar embeddings of the subgraphs $\vec{G}_f$ and $\vec{G}_b$ are induced. Let *lev* be the unique leveling of $\vec{T}$ such that $lev(r) = 0$. Place each node v in the Cartesian plane on the vertical line ℓ_j if and only if $lev(v) = j$. Let γ be some breadth-first order on V obtained by doing a breadth-first search of T starting at r. Two level-j nodes u and v are placed on ℓ_j such that node u is placed below node v when $u <_\gamma v$, that is, when u is reached in the breadth-first search before v.

We now verify that this placement of the nodes in the plane induces a directed leveled-planar embedding of $\vec{G}_f$. Clearly, each arc in $\vec{E}_f$ can be drawn as a line segment whose endpoints lie on vertical lines ℓ_j and ℓ_{j+1} for some integer j. The fact

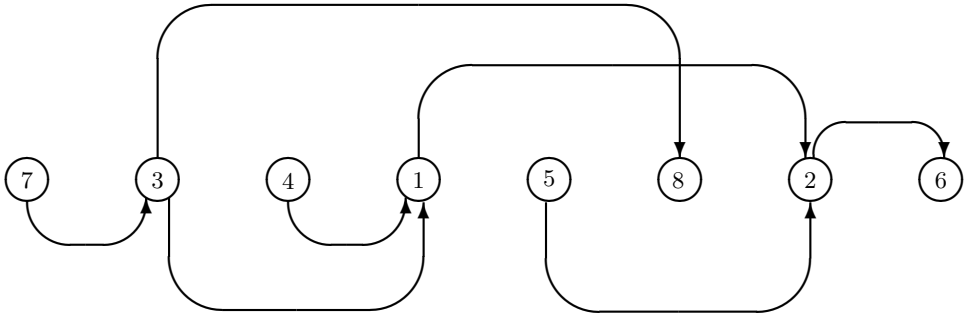


FIG. 3.2. *A 2-queue layout of the tree dag in Figure 2.1.*

that no two arcs in $\vec{E}_f$ intersect can be verified as follows. Consider an independent pair of arcs, (u_1, v_1) and (u_2, v_2) in $\vec{E}_f$, where $lev(u_1) \leq lev(u_2)$. If $lev(u_1) < lev(u_2)$, then $lev(v_1) < lev(v_2)$, so (u_1, v_1) and (u_2, v_2) do not intersect. Now suppose that $lev(u_1) = lev(u_2)$. Without loss of generality, we may assume that $u_1 <_\gamma u_2$. Since (u_1, v_1) is a forward arc and u_1 is the parent of v_1, we have $u_1 <_\gamma v_1$. Since (u_2, v_2) is a forward arc and u_2 is the parent of v_2, we have $u_2 <_\gamma v_2$. By the properties of breadth-first search, we have $v_1 <_\gamma v_2$; hence the arcs (u_1, v_1) and (u_2, v_2) do not intersect. A similar argument shows that this placement of nodes also induces a directed leveled-planar embedding of $\vec{G}_b$. Thus the leveled-planar order of V induced by the leveled-planar embedding of $\vec{G}_f$ yields a 2-queue layout of $\vec{T}$ in which arcs in $\vec{E}_f$ are assigned to one queue and arcs in $\vec{E}_b$ are assigned to the other queue. □

Figure 3.2 gives a 2-queue layout of the tree dag shown in Figure 2.1. This 2-queue layout is constructed as described in the proof of Theorem 3.1, with node 1 being the root.

Next we show that the construction of a 2-queue layout for a tree dag described in Theorem 3.1 actually yields a 1-queue layout for a path dag.

COROLLARY 3.2. *Every path dag has a 1-queue layout.*

Proof. Let $\vec{P} = (V, \vec{E})$ be a path dag with covering graph $P = (V, E)$. Let r be one of the endpoints of P. Consider the placement of the nodes in $\vec{P}$ in the plane as described in the proof of Theorem 3.1. Suppose that the arcs in $\vec{E}$ have been drawn as straight line segments. We have already verified that an independent pair of arcs $(u_1, v_1), (u_2, v_2) \in \vec{E}$ do not intersect if $lev(u_1) \neq lev(u_2)$ or if both arcs are forward arcs or both arcs are backward arcs. Assume that $(u_1, v_1) \in \vec{E}_f$, $(u_2, v_2) \in \vec{E}_b$, and $lev(u_1) = lev(u_2)$. Since $\vec{P}$ is a path dag, u_1 is visited before u_2 in the breadth-first search of P starting from an endpoint r if and only if v_1 is visited before v_2. This implies that the arcs (u_1, v_1) and (u_2, v_2) do not intersect, and we have a directed leveled-planar embedding of $\vec{P}$. We may conclude that $\vec{P}$ has a 1-queue layout, as desired. □

We postpone the problem of characterizing 1-queue tree dags (see section 4.1).

For now we turn our attention to cycle dags. Not all cycle dags can be laid out in 1 queue. As an example, Figure 3.3 shows a smallest cycle dag whose queuenumber is 2. However, the following argument shows that every cycle dag has a 2-queue layout. Let $\vec{C} = (V, \vec{E})$ be a cycle dag, and let u be a node with indegree 0. By Corollary 3.2, the path dag $\vec{C} - u$ has a 1-queue layout. Place the node u on the left of the 1-queue layout of $\vec{C} - u$ and assign the two arcs incident on u to a second queue. This gives

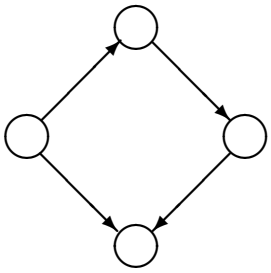


FIG. 3.3. *A smallest cycle dag whose queuenumber is* 2.

a 2-queue layout of $\vec{C}$. We obtain the following theorem.

THEOREM 3.3. *Every cycle dag has a* 2*-queue layout.*

As in the case of tree dags, we postpone characterizing 1-queue cycle dags (see section 4.2).

The observation that any cycle dag can be laid out in 2 queues motivates the question of whether any unicyclic dag can be laid out in 2 queues. A simple argument shows that 3 queues suffice. Let $\vec{U} = (V, \vec{E})$ be a unicyclic dag, and let $\vec{C}$ be the cycle dag that is the subgraph of $\vec{U}$. Let u be a node in $\vec{C}$ with indegree 0 (in $\vec{C}$). Delete the two outgoing arcs in $\vec{C}$ incident on u from $\vec{U}$ to obtain two tree dags $\vec{T}_1$ and $\vec{T}_2$, where $\vec{T}_1$ contains u. A 3-queue layout of $\vec{U}$ is obtained by placing a 2-queue layout of $\vec{T}_1$ to the left of a 2-queue layout of $\vec{T}_2$ and then assigning the two arcs in $\vec{C}$, incident on u, to a third queue. The following theorem gives a more subtle construction of a 2-queue layout of $\vec{U}$.

THEOREM 3.4. *Every unicyclic dag has a* 2*-queue layout.*

Proof. Let $\vec{U} = (V, \vec{E})$ be a unicyclic dag, and let $\vec{C} = (V_C, \vec{E}_C)$ be the unique cycle dag in $\vec{U}$. Without loss of generality, we may assume that $\vec{U}$ is connected. We start with the selection of a particular arc, call it the *closing arc,* to remove from the cycle dag $\vec{C}$. Let $u \in V_C$ be any node of indegree 0 (in $\vec{C}$), and let $v, w \in V_C$ be the out-neighbors of u in $\vec{C}$. Let $\vec{C} - u$ be the path dag obtained by removing u from the cycle dag. Choose a leveling *lev* of $\vec{C} - u$. Without loss of generality, we may assume that $1 = lev(v) \leq lev(w)$. Let $M = lev(w)$. Select (u, w) to be the closing arc. Let $\vec{P} = (V_C, \vec{E}_C - \{(u, w)\}) = (V_C, \vec{E_P})$ be the path dag obtained by removing the closing arc from the cycle dag; let $P = p_1, p_2, \ldots, p_k$ be its covering path, where $p_1 = u$, $p_2 = v$, and $p_k = w$. The leveling *lev* of $\vec{C}$ can be extended to $\vec{P}$ by setting $lev(u) = 0$. Since the minimum value of *lev* occurs at some node of in-degree 0 on $\vec{C}$, we may assume that $0 = lev(p_1) = \min\{lev(p_i) \mid 1 \leq i \leq k\}$. Let $\vec{T} = (V, \vec{E} - \{(p_1, p_k)\})$ be the tree dag obtained by removing the closing arc from the unicyclic dag. The leveling *lev* extends to a leveling of $\vec{T}$. Root $\vec{T}$ at p_1. Take the definitions of forward and backward arcs from the proof of Theorem 3.1; forward arcs are directed away from p_1, and backward arcs are directed toward p_1. Furthermore, partition $\vec{E} - \vec{E}_C$, the set of noncycle arcs, into the set $\vec{E_f}$ of forward arcs and the set $\vec{E_b}$ of backward arcs.

Our task is to construct a topological order σ for $\vec{U}$ while assigning its arcs to one of 2 queues, q_1 and q_2. The construction places each node x on one of the vertical lines ℓ_j, denoted both by $line(x) = j$ and by $\ell(x) = \ell_j$. The order σ is derived from this placement by exactly the same scan that finds the leveled-planar order from a directed leveled-planar embedding, that is, by scanning the lines in the order $\ldots, \ell_j, \ell_{j+1}, \ldots$,

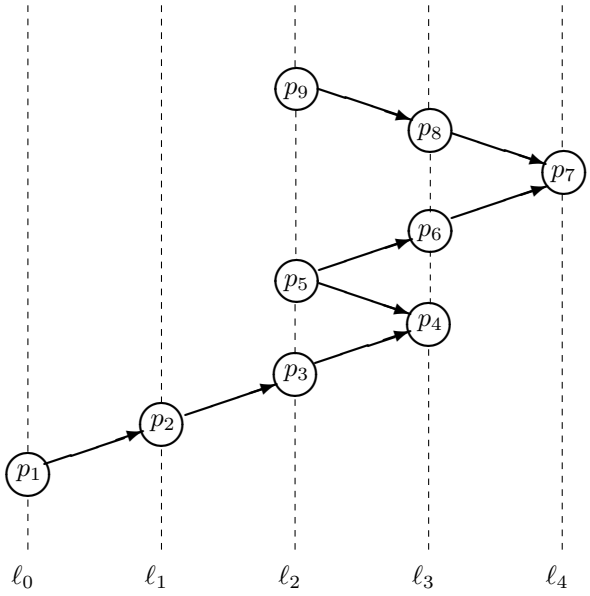


FIG. 3.4. *A leveled-planar layout of the path dag* $\vec{P}$.

each from bottom to top.

The initial phase of the construction places the nodes in $\vec{P}$ as in the proof of Corollary 3.2. See Figure 3.4 for an example. The arcs in $\vec{E_P}$ are assigned to queue q_1. The closing arc is assigned to queue q_2. A 2-queue layout of $\vec{C}$ results. Also, $line(p_i) = lev(p_i)$, for all $p_i \in V_C$. If $(p_i, p_{i+1}) \in \vec{E_P}$ is a forward arc, then define $next(p_i) \in V_C$ to be p_c, where c is smallest such that $i+2 \leq c \leq k$ and $lev(p_c) = lev(p_i)$; note that (p_c, p_{c-1}) must be a backward arc. If there is no such c, let $next(p_i)$ be undefined. In Figure 3.4, $p_9 = next(p_5)$, while $next(p_2)$ is undefined. Because we chose p_1 to have minimum level among nodes in $\vec{P}$, for any backward arc (p_c, p_{c-1}), there is some forward arc (p_i, p_{i+1}) such that $p_c = next(p_i)$. One may verify this assertion for the three backward arcs in Figure 3.4. The construction of Corollary 3.2 places $next(p_i)$ immediately above p_i on $\ell(p_i)$. By the definition of $next(p_i)$, the path from p_i to $next(p_i)$ is to the right of $\ell(p_i)$. Since every arc in that path is placed in queue q_1 and all arcs in $\vec{E_f}$ are to be assigned to queue q_1, there is an impediment to having an arbitrary tree of forward arcs grow from a node placed on line $\ell(p_i)$ between p_i and $next(p_i)$. We say that a node placed between p_i and $next(p_i)$ is in a *forbidden position.* In Figure 3.4, a node placed between p_3 and $p_5 = next(p_3)$ on ℓ_2 is in a forbidden position, while a node placed between p_4 and p_6 on ℓ_3 is not.

The second and final phase of the construction is an inductive extension of the 2-queue layout of $\vec{C}$ to a 2-queue layout of $\vec{U}$, with the addition of one node and one arc at each inductive step. The induction hypothesis is that at each step a 2-queue layout results satisfying these properties:

1. Each arc in $\vec{E_f}$ is assigned to queue q_1.
2. Each arc in $\vec{E_b}$ is assigned to queue q_2.
3. If $(x, y) \in \vec{E_b}$, then $line(x) \leq 0$ or $M \leq line(x)$.
4. No node is in a forbidden position.

Note that properties 1 and 2 imply these additional properties:

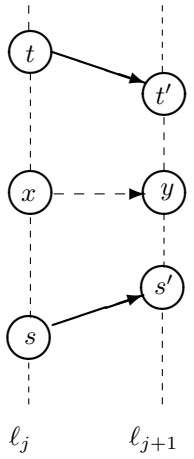


FIG. 3.5. *Illustration of the definitions of s, s', t, and t' for Case* 1.

5. If $x \in V$ and there are distinct arcs (y, x) and (z, x) assigned to queue q_1, then x is some p_i, where $2 \leq i \leq k-1$ and $\{y, z\} = \{p_{i-1}, p_{i+1}\}$. In particular, there exists no $x \in V - V_C$ with two in-neighbors y and z such that (y, x) and (z, x) are q_1 arcs.

6. There exists no $x \in V$ with two out-neighbors y and z such that (x, y) and (x, z) are q_2 arcs.

For the base case, take the construction of the 1-queue layout of $\vec{C}$. This 2-queue layout satisfies the induction properties. At each subsequent step, one remaining arc (x, y) is added; an arc can be added as soon as one of its nodes is already placed and the other is not yet placed. There are three cases for the addition of (x, y). In each case, we specify the placement of the new arc and show that the induction properties are maintained.

Case 1. *Adding a forward arc* $(x, y) \in \vec{E}_f$. Since x is the parent of y, x has already been placed on line $\ell(x)$ and ordered according to σ. Set $line(y) = line(x)+1$. Choose s on $\ell(x)$ largest (with respect to σ) such that $s \leq_\sigma x$, and s has an out-neighbor s' that has already been placed with (s, s'), a q_1 arc; choose s' as large as possible with respect to σ. Choose t on $\ell(x)$ smallest (with respect to σ) such that $x <_\sigma t$ and t has an out-neighbor t' that has already been placed with (t, t'), a q_1 arc; choose t' as small as possible. Either s or t may be undefined. Intuitively, (s, s') and (t, t') are the arcs that we need to place (x, y) between. See Figure 3.5 for an illustration of these arcs. If t is undefined, then place y above all other nodes in $line(y)$; in this case, (x, y) cannot nest with any q_1 arc. Now suppose that t is defined. Place y immediately below t'. If s is undefined, then (x, y) cannot nest with any q_1 arc. Suppose s is defined. Since (s, s') and (t, t') do not nest, we know that $s' \leq_\sigma t'$. If $s' \neq t'$, then (x, y) does not nest with any q_1 arc. Now suppose $s' = t'$. By property 5, (s, s') and (t, t') are consecutive arcs in $\vec{P}$. Moreover, $t = next(s)$. Since (by induction property 4) x is not in a forbidden position, we must have $x = s$. The placement of y cannot cause (x, y) to nest with any q_1 arc. In all cases, we can assign (x, y) to queue q_1, as required by induction property 1.

The continued truth of induction properties 1–3 is clear. It remains to show that y is not in a forbidden position. If y is placed above all other nodes on $\ell(y)$, then y

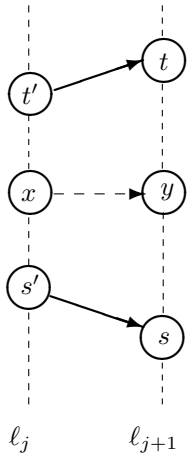


FIG. 3.6. *Illustration of the definitions of s, s', t, and t' for Case* 2.

cannot be in a forbidden position. If $t' \notin V_C$ and y is placed immediately below t', then y is not in a forbidden position, because t' is not. So suppose $t' \in V_C$ and y is placed immediately below t'. Then $(t, t') \in \vec{E_P}$, since otherwise (t, t') would be a q_2 arc. If (t, t') is a forward arc, then t' is not $next(p_i)$ for any $p_i <_\sigma t'$ and y is not in a forbidden position. So suppose (t, t') is a backward arc. Then $t = next(p_i)$ for some $p_i <_\sigma t$, and x is in a forbidden position. This contradicts induction property 4. We conclude that (t, t') is not a backward arc. In all cases, y is not in a forbidden position, and induction property 4 holds.

Case 2. *Adding a backward arc $(x, y) \in \vec{E_b}$ when $line(y) \leq 0$ or $M + 1 \leq line(y)$.* Since y is the parent of x, y has already been placed on line $\ell(y)$ and ordered according to σ. Set $line(x) = line(y) - 1$. Choose s on $\ell(y)$ largest (with respect to σ) such that $s \leq_\sigma y$ and s has an in-neighbor s' that has already been placed and (s', s) is a q_2 arc; choose s' as large as possible. Choose t on $\ell(y)$ smallest (with respect to σ) such that $y <_\sigma t$ and t has an in-neighbor t' that has already been placed and (t', t) is a q_2 arc; choose t' as small as possible. Either s or t may be undefined. See Figure 3.6 for an illustration of these arcs. If t is undefined, then place x above all other nodes on $\ell(x)$; this cannot cause (x, y) to nest with any q_2 arc. If t is defined, then place x just below t'. The only way this can cause (x, y) to nest with a q_2 arc is if s is defined and $s' = t'$; by property 6, this cannot happen. Assign (x, y) to queue q_2, as required by induction property 2.

The continued truth of induction properties 1–3 is clear. It remains to show that x is not in a forbidden position. If x is placed above all other nodes on $\ell(x)$, then x cannot be in a forbidden position. So suppose x is placed just below t'. Since $(t', t) \in \vec{E_b}$, we know that $t' \notin V_C$. Hence x is not in a forbidden position because t' is not. We conclude that x is not in a forbidden position and that induction property 4 holds.

Case 3. *Adding a backward arc $(x, y) \in \vec{E_b}$ when $1 \leq line(y) \leq M$.* Since y is the parent of x, y has already been placed on line $\ell(y)$ and ordered according to σ. Set $line(x) = 0$ to satisfy induction property 3. Choose s largest (with respect to σ) such that $s \leq_\sigma y$ and s has an in-neighbor s' that has already been placed, (s', s) is a q_2 arc, and $line(s') = 0$; choose s' as large as possible. Choose t smallest (with respect to σ) such that $y <_\sigma t$ and t has an in-neighbor t' that has already been placed, (t, t')

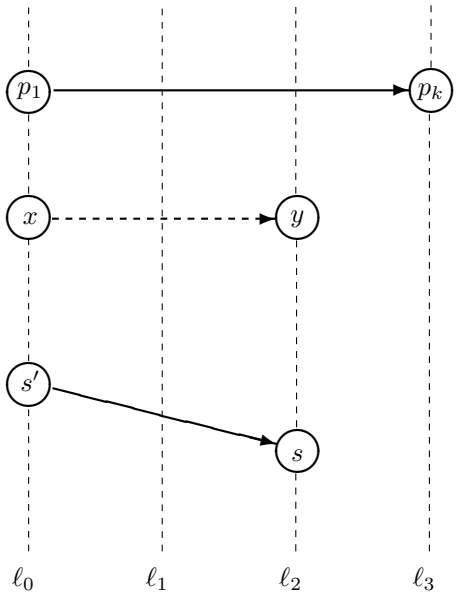


FIG. 3.7. *Illustration of the definitions of s, s', t, and t' for Case* 3.

is a q_2 arc, and $line(t') = 0$; choose t' as small as possible. Either s or t may be undefined but not both, because the closing arc meets the requirements of one of (s', s) and (t', t). See Figure 3.7 for an illustration of these arcs; here $(t', t) = (p_1, p_k)$, the closing arc. If t is undefined, then place x above all other nodes in $line(x)$; this cannot cause (x, y) to nest with any q_2 arc. If t is defined, then place x just below t'. The only way this can cause (x, y) to nest with a q_2 arc is if s is defined and $s' = t'$; by property 6, this cannot happen. Assign (x, y) to queue q_2, as required by induction property 2.

The continued truth of induction properties 1–3 is clear. It remains to show that x is not in a forbidden position. If x is placed above all other nodes on ℓ_0, then x cannot be in a forbidden position. So suppose x is placed just below t'. If $(t', t) \in \vec{E}_b$, then $t' \notin V_C$ and x is not in a forbidden position because t' is not. If (t', t) is the closing arc, then x is not in a forbidden position because it is placed below p_1 on ℓ_0. We conclude that x is not in a forbidden position and that induction property 4 holds.

By induction, we obtain a 2-queue layout of $\vec{U}$, as desired. □

Figure 3.8 shows a unicyclic dag $\vec{U}$ with 12 nodes. $\vec{U}$ contains a cycle dag induced by the nodes $\{1, 2, 3, 4, 5\}$. Figure 3.9 shows a 2-queue layout of $\vec{U}$. The cycle dag $\vec{C}$ is laid out in 2 queues, with closing arc $(1, 5)$ assigned to one queue and the rest assigned to another queue. The remainder of $\vec{U}$ is laid out as in the construction in the proof of Theorem 3.4.

4. Characterization of 1-queue directed trees and directed cycles. In this section we characterize 1-queue tree dags and 1-queue cycle dags. First we prove a theorem about leveled-planar dags used in both characterizations. We need the following definition. Suppose $\vec{G}$ is a leveled dag with leveling *lev*. A queue layout of $\vec{G}$ *respects lev* if all nodes in $lev^{-1}(j)$ appear contiguously in the layout for every j.

THEOREM 4.1. *Suppose that $\vec{G}$ is a leveled dag. Then, $\vec{G}$ is a* 1*-queue dag if and only if it has a directed leveled-planar embedding. Moreover, if lev is any leveling of*

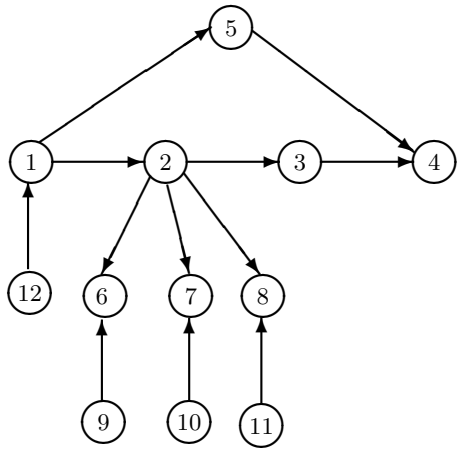


FIG. 3.8. *A unicyclic dag.*

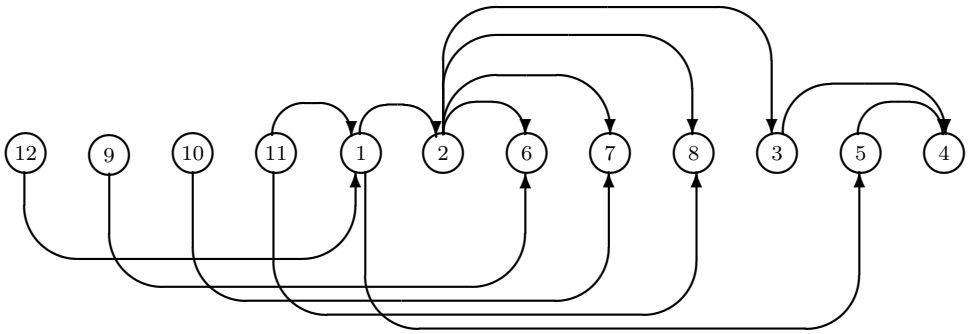


FIG. 3.9. *A 2-queue layout of the unicyclic dag shown in Figure* 3.8.

$\vec{G}$, then $\vec{G}$ has a 1-queue layout that respects lev.

Proof. By Proposition 1.1, if $\vec{G}$ has a directed leveled-planar embedding, then it is a 1-queue dag.

To show the converse, suppose that $\vec{G}$ is a 1-queue dag. Let σ be a total order on V that yields a 1-queue layout of $\vec{G}$. Let *lev* be any leveling of $\vec{G}$. Place each level-j node u on the vertical line ℓ_j. On each vertical line, place nodes bottom to top in the order prescribed by σ and draw each arc in $\vec{E}$ as a straight line segment connecting its two endpoints. We now show that this embedding is a directed leveled-planar embedding. It suffices to show that no two independent arcs (u_1, v_1) and (u_2, v_2) with $lev(u_1) = lev(u_2)$ intersect. Without loss of generality, we may assume that $u_1 <_\sigma u_2$ and that $lev(u_1) = lev(u_2) = j$ for some integer j. Since $u_1 <_\sigma u_2$, u_1 appears below u_2 on ℓ_j. Since σ defines a 1-queue layout, we have $v_1 <_\sigma v_2$. Hence v_1 appears below v_2 on ℓ_{j+1}. We conclude that the arcs (u_1, v_1) and (u_2, v_2) do not intersect. Hence $\vec{G}$ has a directed leveled-planar embedding, as desired.

To show the last assertion, suppose that *lev* is any leveling of $\vec{G}$. Then the above construction of a leveled-planar embedding of $\vec{G}$ yields a 1-queue layout of $\vec{G}$ that respects *lev*, as desired. □

An example of a leveled dag is a tree dag. The above theorem implies that any 1-queue tree dag has a leveled-planar embedding that respects any leveling. This is

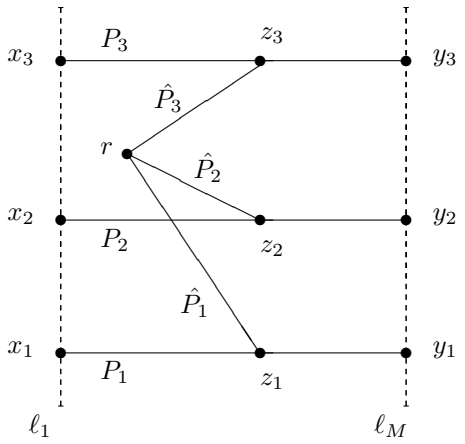


FIG. 4.1. *A typical dag in the forbidden set* $\mathcal{F}$.

a very useful result because it allows us to ignore the possibility that there might be arches in the embedding.

4.1. Characterization of 1-queue directed trees. The next theorem gives a forbidden graph characterization of tree dags with queuenumber equal to 1. We begin with some notation. Let $\vec{T} = (V, \vec{E})$ be a tree dag with covering graph $T = (V, E)$. Root $\vec{T}$ at an arbitrary node r. If u and v are nodes of T, then the (unique) path in T from u to v is denoted $P(u, v)$. For any $u \in V$, let $\vec{T}_u = (V_u, \vec{E}_u)$ be the subtree of $\vec{T}$ rooted of u. We extend this notation to *sets* of nodes: if S is a set of nodes in V, then $\vec{T}_S$ is the directed forest that is the union of the tree dags $\vec{T}_u$ for all $u \in S$.

For any subdag $\vec{D}$ of tree dag $\vec{T}$, with leveling function *lev*, we let $minlev(\vec{D}) = \min\{lev(v) \mid v \in V(\vec{D})\}$. When $\vec{D}$ is the subtree $\vec{T}_u$ for some node u, we shorten $minlev(\vec{T}_u)$ to $m\ell T(u)$. As an example, if the dag in Figure 4.2 is rooted at r, then $m\ell T(b) = 2$, $m\ell T(z_2) = m\ell T(z^*) = 1$, and $m\ell T(z_1) = 3$.

Next we define a set $\mathcal{F}$ of tree dags and show that the set of tree dags with queuenumber equal to 1 is characterized by forbidding the set $\mathcal{F}$. Each dag $\vec{F} = (V_F, \vec{E}_F)$ in $\mathcal{F}$ consists of two sets of three nontrivial path dags, to wit, $\{P_1, P_2, P_3\}$ and $\{\hat{P}_1, \hat{P}_2, \hat{P}_3\}$ (refer to Figure 4.1). There is an integer $M \geq 2$ and a leveling function *lev:* $V_F \to \mathbf{Z}$ for which $1 \leq lev(u) \leq M$ for all $u \in V_F$. Furthermore, the paths P_i, with endpoints x_i and y_i, are node disjoint and have $lev(x_i) = 1$ and $lev(y_i) = M$, for $i = 1, 2, 3$. The paths $\hat{P}_i$ have endpoints r and z_i and share exactly one node r. The paths $\hat{P}_i$ and P_j share precisely the node z_i if $i = j$ and are node disjoint otherwise. The general case is shown in Figure 4.1, where all arcs point from left to right (arc directions are omitted in that figure). Note that the dag in Figure 3.1, in which each of the paths is just a single arc, is the smallest dag in the set $\mathcal{F}$.

THEOREM 4.2. *Let* $\vec{T} = (V, \vec{E})$ *be a tree dag. Then* $\vec{T}$ *is a leveled-planar dag if and only if* $\vec{T}$ *has no subdag in the set* $\mathcal{F}$ *of forbidden dags described above.*

Proof. ($\Longrightarrow$) We show that the graphs in $\mathcal{F}$ are *not* leveled-planar dags. Let $\vec{F}$ be a dag in $\mathcal{F}$ consisting of the paths P_i and $\hat{P}_i$ for $i = 1, 2, 3$. Suppose there were a directed leveled-planar embedding of $\vec{F}$. Choose such an embedding of $\vec{F}$ so that

the leveling function associated with this embedding has the set $\{1, 2, \ldots, M\}$ as its range. The node disjoint paths P_1, P_2, and P_3 divide the strip between the lines ℓ_1 and ℓ_M into four regions (see Figure 4.1). The node r must lie in one of these regions and thus is cut off from at least one of the paths, say P_1. The path $\hat{P}_1$ must cross one of P_2, P_3, a contradiction. Hence $\vec{F}$ is not a leveled-planar dag, as desired.

($\Longleftarrow$) We now show that if $\vec{T}$ does not have a subgraph that belongs to $\mathcal{F}$, then $\vec{T}$ has a directed leveled-planar embedding. We proceed by induction on the number of nodes. Clearly the theorem is true for the graph with one node. Assume that all tree dags with fewer than N nodes, $N \geq 2$, satisfy Theorem 4.2. Let $\vec{T}$ be a tree dag with N nodes. Assume that $\vec{T}$ has no subgraph that belongs to $\mathcal{F}$. We show that $\vec{T}$ has a directed leveled-planar embedding.

Let v be a leaf of $\vec{T}$. Since the set $\mathcal{F}$ of forbidden graphs is symmetric with respect to reversing all arc directions, we may assume that the arc incident on v in $\vec{T}$ points towards v. Let x be the parent of v.

Since $\vec{T} - v$ has no subgraph in $\mathcal{F}$ and has $N - 1$ nodes, it is a leveled-planar dag by the induction hypothesis. Fix a directed leveled-planar embedding $\mathcal{E}$ of $\vec{T} - v$. Let *lev* be the leveling of $\vec{T}$ induced by $\mathcal{E}$ such that $1 \leq lev(u) \leq m$, for all $u \in V$, for some integer $m \geq 2$. Suppose that $lev(x) = j - 1$ for some j, $1 < j \leq m + 1$. If v can be placed on the line ℓ_j so that the line segment connecting x and v does not cause a crossing, then we have a directed leveled-planar embedding of $\vec{T}$, as desired.

Otherwise, it is not hard to see that there exists a level-j node r so that in the embedding $\mathcal{E}$, node r has in-neighbors on line ℓ_{j-1} both above and below x. Root $\vec{T}$ at the node r. Extend *lev* to the domain V by setting $lev(v) = lev(x) + 1 = j$. The dags $\vec{T}$ shown in Figures 4.2 through 4.5 are meant to illustrate the notation and arguments given in this proof. In each case the given embedding of $\vec{T} - v$ is leveled-planar, but the arc (x, v) crosses another arc.

Let S be the set of all in-neighbors of r other than x. Since $\mathcal{E}$ is a leveled-planar embedding of $\vec{T} - v$, the nodes of S all lie on line ℓ_{j-1}. Let B consist of those $u \in S$ such that the tree $\vec{T}_u$ lies entirely to the left of ℓ_j, i.e., $u \in B$ implies that, for all $w \in T_u$, we have $lev(w) \leq j - 1$. For the graph in Figure 4.2, we have $j = 4$, $S = \{z_1, z_2, b, y_1\}$, and $B = \{b\}$. For all $y \in S - B$, define

$$H(y) = V(\vec{T}_y) \cap lev^{-1}(j) \cap \left\{u \mid j = \max_{w \in P(u,y)} lev(w)\right\},$$

that is, $H(y)$ consists of the nodes at which the branches of $\vec{T}_y$ first touch the line ℓ_j. If possible, modify $\mathcal{E}$ so that for each y, the nodes in $H(y)$ lie on the same side of r (i.e., all above r or all below r). Let $h(y)$ be the node in $H(y)$ that is closest to the node r; if there are two possibilities w_1 and w_2 (one above and one below r), choose the one for which $minlev(P(r, w_i))$ is larger and choose arbitrarily if $minlev(P(r, w_1)) = minlev(P(r, w_2))$. By planarity, there can be at most one node y^* for which $H(y^*)$ contains nodes both above and below r on line ℓ_j. If our embedding has such a node y^*, and $h(y^*)$ was chosen above r on line ℓ_j, then reflect the entire embedding about the x-axis, so that $h(y^*)$ is now below r on line ℓ_j. Denote the element of $\{w_1, w_2\}$ which is *not* $h(y^*)$ by $h'(y^*)$. Thus $h'(y^*)$ is above r, $h(y^*)$ is below r, and $minlev(P(y^*, h'(y^*))) \leq minlev(P(y^*, h(y^*)))$. Again referring to the dag of Figure 4.2, we have $H(z_1) = \{g\}$, $h(z_1) = g$, $H(z_2) = \{i, h\}$, $h(z_2) = h$, $H(y_1) = \{e\}$, $h(y_1) = e$, and the node y^* is not present.

Recall that $m\ell T(y) = minlev(\vec{T}_y)$ is the minimum level the tree $\vec{T}_y$ reaches. We also find it useful to use $m\ell P(y)$ as an abbreviated notation for $minlev(P(y, h(y)))$

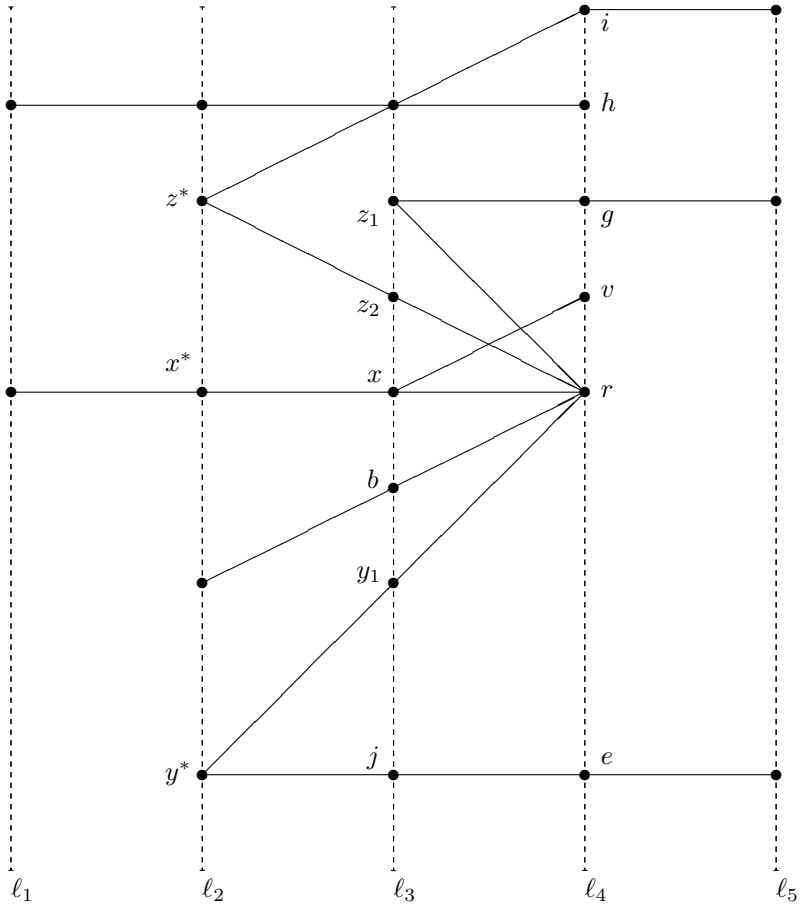


FIG. 4.2. *An example for Case* 1.

when y is a node in $S - B$. Thus $m\ell P(y)$ is the minimum level the path from y to $h(y)$ reaches. Clearly, we have $m\ell T(y) \leq m\ell P(y)$. In the tree dag of Figure 4.2, $m\ell T(z_1) = m\ell P(z_1) = 3$, $m\ell T(z_2) = 1$, and $m\ell P(z_2) = 2$.

We consider the cases $(x, r) \in \vec{E}$ and $(x, r) \notin \vec{E}$ separately.

Case 1. $(x, r) \in \vec{E}$. For this case, refer to Figure 4.2. Make a new embedding of $T - v$ in which the number of nodes z in $S - B$ with $h(z)$ above r is maximized while keeping nodes x and r and paths $P(y^*, h(y^*))$ and $P(y^*, h'(y^*))$ (if y^* is present) fixed in position. Write $S - B = A \cup C$, where the nodes $y \in A$ have $h(y)$ below r and the nodes $z \in C$ have $h(z)$ above r. Thus $|C|$ is as large as possible and y^* (if it is present) is in A.

Next we show that we can place the nodes in B above the nodes in A and below those in C. The embedding of the dag shown in Figure 4.2 has this form with $A = \{y_1\}$, $B = \{b\}$, and $C = \{z_1, z_2\}$. If a node $a \in A$ appeared above a node $c \in C$, then the paths $P(a, h(a))$ and $P(c, h(c))$ would cross, a contradiction. If the node y^* is not present, then there is a gap between the highest node of A and the lowest node of C

on line ℓ_{j-1} into which the nodes of B can be placed and the trees $\vec{T}_b$ for $b \in B$ will face no obstruction. If y^* is present, then by construction, $minlev(P(y^*, h'(y^*))) \leq minlev(P(y^*, h(y^*)))$, so any trees $\vec{T}_b$ with b below y^* can be transferred so that b lies directly above y^* and the trees $\vec{T}_b$ which fit inside $P(y^*, h(y^*))$ also fit inside $P(y^*, h'(y^*))$. Since y^* is the highest node of A (when it is present), this shows that the nodes of B can be placed above those of A and below those of C.

If $A = \emptyset$, then x can be placed as the lowest node on line ℓ_{j-1} and there is room for the entire tree $\vec{T}_x$ (including v) without causing a crossing. This would give a leveled-planar embedding of $\vec{T}$, as desired. Hence we may assume that $|A| \geq 1$. Similarly, we may assume that $|C| \geq 1$, because if $C = \emptyset$ and y^* is present, then x can be placed directly above node y^* on line ℓ_{j-1}, and there is room for the entire tree $\vec{T}_x$ (including v) without causing a crossing. Let $A = \{y_1, y_2, \ldots, y_m\}$, where $i < j$ implies that y_i is below y_j (and thus that $h(y_i)$ is above $h(y_j)$). Note that if y^* is present then $y^* = y_m$. Similarly, let $C = \{z_1, z_2, \ldots, z_n\}$, where $i < j$ implies that z_i is above z_j (and thus that $h(z_i)$ is below $h(z_j)$). The nodes of $A \cup C$ in Figure 4.2 are labeled in this way.

If $minlev(\vec{T}_x) > m\ell P(y_1)$, then the entire tree $\vec{T}_x$ can be placed inside the path $P(y_1, h(y_1))$ with x placed just below y_1 on ℓ_{j-1}. This allows room for v between $h(y_1)$ and r, and the arc (x, v) does not cause a crossing. Thus we would have a directed leveled-planar embedding of $\vec{T}$, as desired.

Otherwise, $minlev(\vec{T}_x) \leq m\ell P(y_1)$, as is the case for the embedding in Figure 4.2. Let t be the minimum positive integer such that $m\ell T(z_t) \leq m\ell P(y_1)$ (such a t exists; otherwise, y_1 could be placed just below z_n on ℓ_{j-1} and added to C, contradicting the maximality of $|C|$). If $t > 1$, then transfer the nested trees $\vec{T}_{z_1}, \ldots, \vec{T}_{z_{t-1}}$ to fit inside $P(y_1, h(y_1))$. In the embedding in Figure 4.2, we have $t = 2$, so $\vec{T}_{z_1}$ is transferred so that z_1 is placed between y_1 and j on line ℓ_3, and g is placed between r and e on line ℓ_4.

Now if $m\ell T(x) > m\ell P(z_t)$, then transfer the entire tree $\vec{T}_x$ inside the path $P(z_t, h(z_t))$. As before, this allows room for the arc (x, v) without causing a crossing, and thus we get a directed leveled-planar embedding of $\vec{T}$. In the embedding in Figure 4.2, $minlev(\vec{T}_x) = 1$ and $m\ell P(z_t) = m\ell P(z_2) = 2$ so the transfer does not take place.

Otherwise, $m\ell T(x) \leq m\ell P(z_t)$, as is the case for the embedding in Figure 4.2. Note that $m\ell T(y_1) \leq m\ell P(z_t)$, for otherwise the trees $\vec{T}_{y_1}, \vec{T}_{z_{t-1}}, \ldots, \vec{T}_{z_1}$ could be nested inside $P(z_t, h(z_t))$, violating the maximality of $|C|$. But now we have the following graph in $\mathcal{F}$ as a subgraph of $\vec{T}$, a contradiction. Let $M = \max\{m\ell P(y_1), m\ell P(z_t)\}$. Since $m\ell T(x) \leq M$, $m\ell T(y_1) \leq M$ and $m\ell T(z_t) \leq M$, we get the node disjoint paths: $P_1 : v \to x^*$, $P_2 : h(y_1) \to y^*$, and $P_3 : h(z_t) \to z^*$, where x^*, y^*, z^* are, respectively, nodes in $\vec{T}_x, \vec{T}_{y_1}, \vec{T}_{z_t}$ at level M. The connecting paths are $\hat{P}_1 : r \to x$, $\hat{P}_2 : r \to \hat{y}$, and $\hat{P}_3 : r \to \hat{z}$, where $\hat{y}$ (resp., $\hat{z}$) is the node farthest from r on both $P(r, y^*)$ and $P(r, h(y_1))$ (resp., $P(r, z^*)$ and $P(r, h(z_t))$). In the embedding in Figure 4.2, we have $M = 2$, $\hat{y} = y^*$, and $\hat{z} = z^*$.

Case 2. $(x, r) \notin \vec{E}$. Let w be the out- or in-neighbor of r such that $x \in \vec{T}_w$. There are two cases, depending on whether $(r, w) \in \vec{E}$ or $(w, r) \in \vec{E}$.

Case 2a. $(r, w) \in \vec{E}$, $lev(w) = j + 1$. For this case, refer to Figure 4.3. Without loss of generality, we may assume that the path $P(w, x)$ approaches x "from above," that is, the path in T from x to w first intersects the line ℓ_{j-1} at a node w_0 which lies above x.

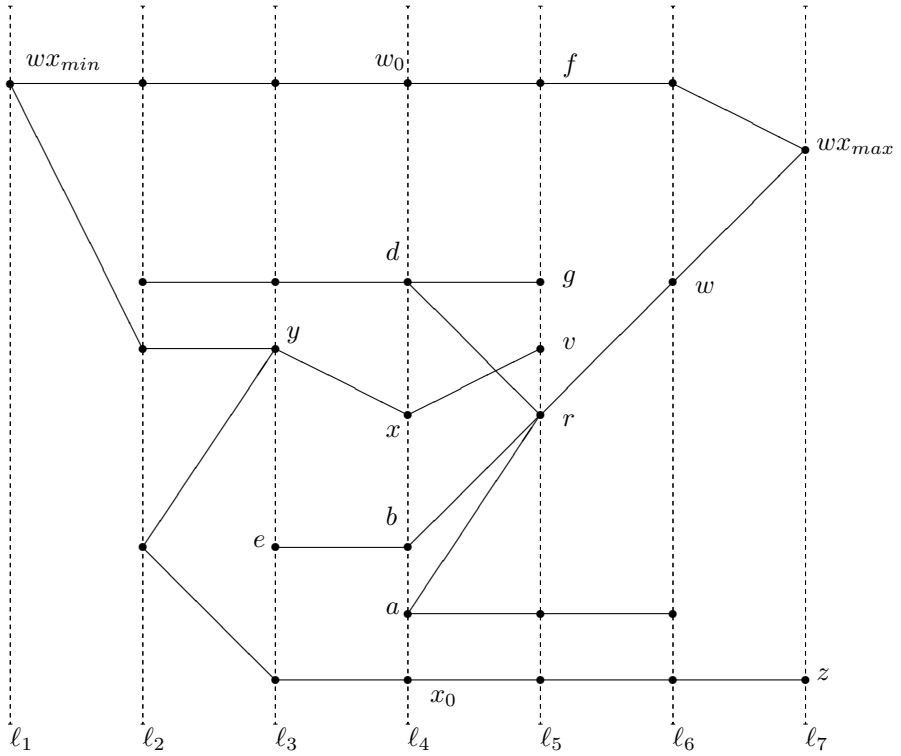


FIG. 4.3. *An example for Case* 2a.

Write $S - B = A \cup C \cup D$, where

$$A = \{u \in S - B \mid h(u) \text{ is below } r \text{ on line } \ell_j\},$$
$$C = \{u \in S - B - A \mid h(u) \text{ is above } r \text{ on line } \ell_j \text{ and } u \text{ is below } x \text{ on line } \ell_{j-1}\},$$
$$D = \{u \in S - B - A \mid h(u) \text{ is above } r \text{ on line } \ell_j \text{ and } u \text{ is above } x \text{ on line } \ell_{j-1}\}.$$

Without loss of generality, we may assume that the nodes of A are below those in B, which are below those in C, which are below those in D. The nodes of S in the embedding in Figure 4.3 are positioned in this way with $j = 5$, $A = \{a\}$, $B = \{b\}$, $C = \emptyset$, and $D = \{d\}$. Let wx_{min} be a node on $P(w, x)$ with $lev(wx_{min}) = \min\{lev(u) \mid u \in P(w, x)\}$ and let wx_{max} be a node on $P(w, x)$ with $lev(wx_{max}) = \max\{lev(u) \mid u \in P(w, x)\}$.

Move node x up so that it is above all nodes in $\vec{T}_D$ on line ℓ_{j-1} but below w_0. This allows room for v on ℓ_j so that the arc (x, v) does not cause a crossing. If the remainder of the tree $\vec{T}_{wx_{min}}$ can be embedded without causing a crossing, then we have a directed leveled-planar embedding of $\vec{T}$, as desired. This would be the case for the embedding in Figure 4.3 if node z were deleted. In that instance, node x would be placed on line ℓ_4 between nodes d and w_0, and v would be placed on line ℓ_5 between nodes g and f, and the rest of $\vec{T}_{wx_{min}}$ would lie above $\vec{T}_D$ but below $P(wx_{max}, wx_{min})$.

If a crossing is forced, then it must be the case that $C = \emptyset$, the node y^* is

not present, and there is a node y on $P(x, wx_{min})$ and a node $z \in \vec{T}_y$ with $lev(z) = lev(wx_{max})$ so that $minlev(P(y,z))$ is less than $minlev(\vec{T}_A)$ and $minlev(\vec{T}_B)$ but greater than or equal to $minlev(\vec{T}_D)$ (see Figure 4.3). In this case, let x_0 be the lowest node on ℓ_{j-1} that is also on $P(y,z)$. Place x just above x_0. This allows room for the arc (x,v) without causing a crossing. Since $lev(wx_{min}) < lev(\vec{T}_D) \leq minlev(P(y,z)) < \min\{minlev(\vec{T}_A), minlev(\vec{T}_B)\}$, each node of $\vec{T}_{wx_{min}}$ (other than wx_{min} itself) can be placed on its appropriate line ℓ_j below the nodes of $\vec{T}_A$ and $\vec{T}_B$. This allows room for the line segment from x to v without causing a crossing; hence we obtain a directed leveled-planar embedding of $\vec{T}$, as desired.

Case 2b. $(w,r) \in \vec{E}$, $lev(w) = j-1$. Write $S - B - \{w\} = A \cup C$, where

$$A = \{u \in S - B - \{w\} \mid h(u) \text{ is below } r\}$$

and

$$C = \{u \in S - B - \{w\} \mid h(u) \text{ is above } r\}.$$

Without loss of generality, we may assume that the nodes of A lie below w and x and those of C lie above w and x (refer to Figures 4.4 and 4.5). Let wx be a node on $P(w,x)$ with $lev(wx) = minlev(P(w,x))$, and let w^* be a node in $\vec{T}_w$ with $lev(w^*) = minlev(\vec{T}_w)$. Note that if y^* is present then the entire tree $\vec{T}_w$ lies inside the path $P(y^*, h'(y^*))$. In the arguments that follow, the node y^* can be treated like any other node of A because the existence of the path $P(y^*, h'(y^*))$ is not relevant.

Consider the integers $m\ell P(u)$ for each $u \in A \cup C$. If $m\ell P(u) < lev(w^*)$ for all $u \in A$, then w can be placed below all nodes in A and the entire tree $\vec{T}_w$ can be nested inside the trees $\vec{T}_u$ (for $u \in A$) without an arc crossing. This would leave room for the arc (x,v) and would give a directed leveled-planar embedding of $\vec{T}$, as desired. The same is true if $m\ell P(u) < lev(w^*)$ for all $u \in C$.

Thus we are left with the case in which there are nodes u of A and of C with $m\ell P(u) \geq lev(w^*)$. Let the set of all such u be $\mathcal{U}$. We consider the case in which there exists a node $z \in \mathcal{U} \cap C$ with $m\ell P(z) \leq m\ell P(u)$ for all $u \in \mathcal{U}$. The remaining case (there exists a $y \in \mathcal{U} \cap A$ with $m\ell P(y) \leq m\ell P(u)$ for all $u \in \mathcal{U}$) is similar. Choose an embedding of $\vec{T} - v$ that maximizes $|\mathcal{U} \cap C|$ and yet preserves the definitions of A and C and retains $z \in C$. Let $A = \{y_1, y_2, \ldots, y_m\}$, where $i < j$ implies that y_i is below y_j. Similarly, let $C = \{z_1, z_2, \ldots, z_n\}$, where $i < j$ implies that z_i is above z_j. Note that $i < j$ implies that $m\ell P(y_i) > m\ell P(y_j)$ and $m\ell P(z_i) > m\ell P(z_j)$.

Let i be the maximum integer such that $m\ell P(y_i) \geq lev(w^*)$, and let j be the maximum integer such that $m\ell P(z_j) \geq lev(w^*)$. By our assumptions, such i, j exist and $m\ell P(y_i) \geq m\ell P(z_j)$. In Figures 4.4 and 4.5, we have $i = 1$ and $j = 3$.

Subcase I. $m\ell P(z_j) \leq lev(wx)$ *and* $m\ell P(y_i) \leq lev(wx)$. For this subcase, refer to Figure 4.4. If $m\ell T(y_i) \leq m\ell P(z_j)$, then we have a forbidden graph in $\vec{T}$ with $M = m\ell P(z_j)$. Hence $m\ell T(y_i) > m\ell P(z_j)$ as is the case in Figure 4.4. If $j = 1$, then the tree $\vec{T}_{y_i}$ can be nested inside $P(z_1, h(z_1))$, contradicting the maximality of $|C|$. If $j > 1$ and $m\ell T(z_{j-1}) \leq m\ell P(y_i)$, we have a forbidden graph in $\vec{T}$ with $M = m\ell P(y_i)$. In the embedding in Figure 4.4, $m\ell T(z_2) = 4$ and $m\ell P(y_2) = 4$ and the forbidden dag consists of the paths $P_1 = P(a,e)$, $P_2 = P(wx,v)$, $P_3 = P(b,f)$, $\hat{P}_1 = P(r,a)$, $\hat{P}_2 = P(r,wx)$, and $\hat{P}_3 = P(r,d)$. Otherwise (e.g., if node "b" were deleted from the embedding in Figure 4.4), $\vec{T}_{y_i}$ could be nested between $P(z_{j-1}, h(z_{j-1}))$ and $P(z_j, h(z_j))$, contradicting the maximality of $|C|$.

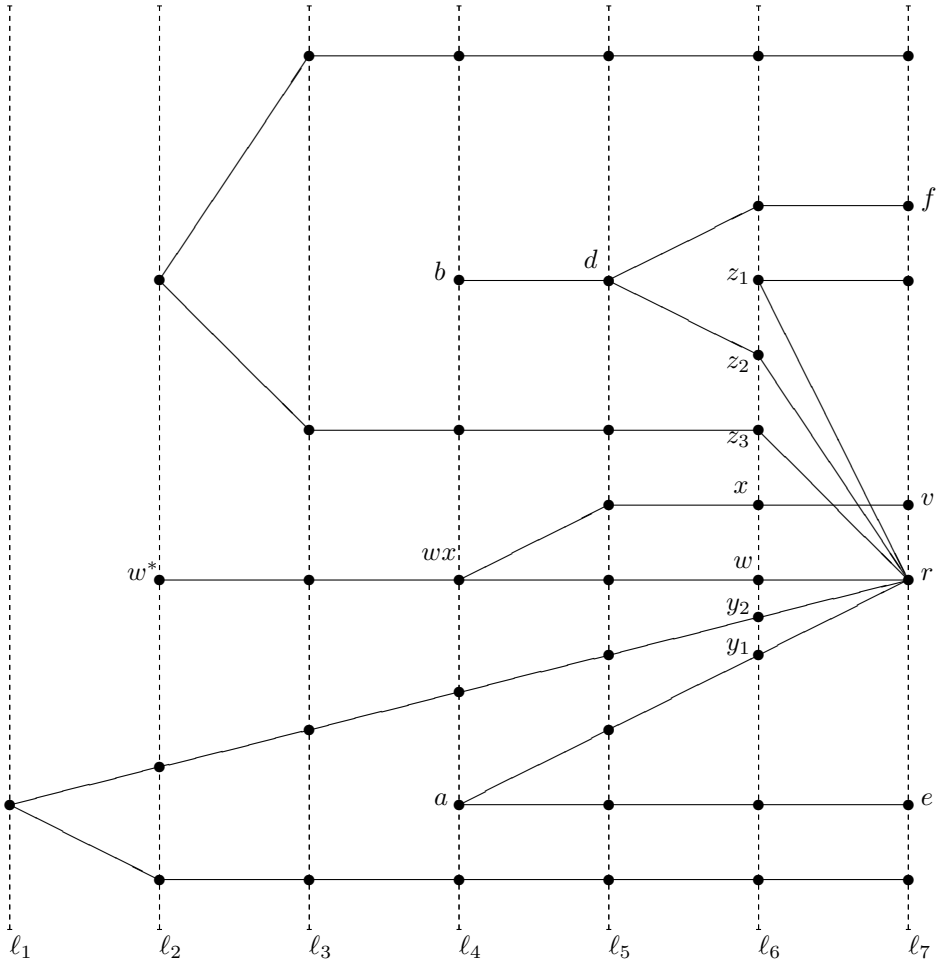


FIG. 4.4. *An example for Subcase* I *of Case* 2b.

Subcase II. $m\ell P(z_j) \leq lev(wx)$ *and* $m\ell P(y_i) > lev(wx)$. For this subcase, refer to the embedding in Figure 4.5 for which $i = 1$ and $j = 3$. If $m\ell T(y_i) \leq m\ell P(z_j)$, then we have a forbidden graph with $M = m\ell P(z_j)$. If $m\ell T(y_i) > lev(wx)$ (which would be the case if node "a" were removed from the embedding in Figure 4.5), then nest $\vec{T}_w$ around $\vec{T}_{y_i}$ and inside $P(y_{i+1}, h(y_{i+1}))$. This allows room for the arc (x, v) without causing a crossing and gives a directed leveled-planar embedding of $\vec{T}$, as desired.

Otherwise, as is the case for the embedding in Figure 4.5, $m\ell P(z_j) < m\ell T(y_i) \leq lev(wx)$. If there were a $c < j$ with $m\ell T(y_i) \leq m\ell P(z_c) \leq lev(wx)$, then we would have a forbidden graph with $M = m\ell P(z_c)$. Hence if we let p be the minimum integer for which $m\ell P(z_p) \leq lev(wx)$, then we have $m\ell T(y_i) > m\ell P(z_p)$. If $p = 1$ (as would be the case if the node z_1 were deleted from the embedding in Figure 4.5), then nest the trees $\vec{T}_{y_1}, \vec{T}_{z_2}, \ldots, \vec{T}_{y_i}$ inside the path $P(z_p, h(z_p))$ and nest $\vec{T}_w$ inside $P(y_{i+1}, h(y_{i+1}))$. This allows room for the arc (x, v) without causing a crossing and gives a directed leveled-planar embedding of $\vec{T}$, as desired.

If $p > 1$ and $m\ell T(z_{p-1}) \leq lev(wx)$, then we have a forbidden graph with

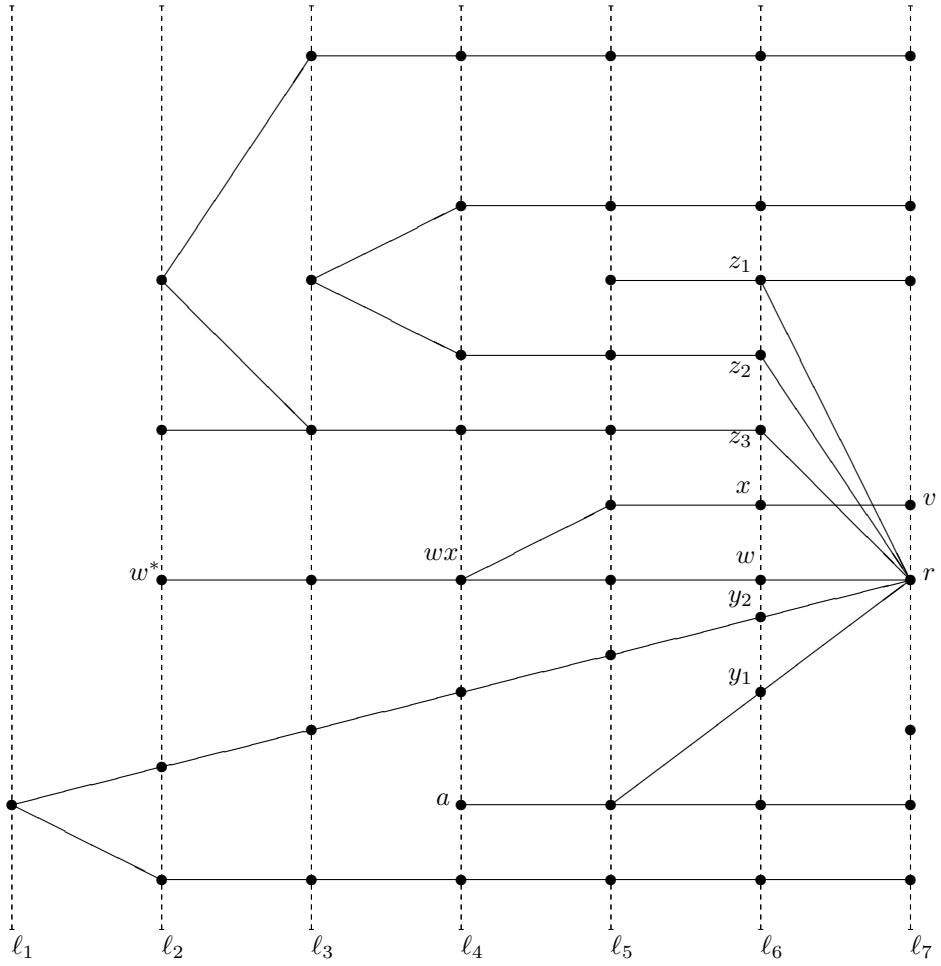


FIG. 4.5. *An example for Subcase* II *of Case* 2b.

$M = lev(wx)$. Otherwise, $p > 1$ and $m\ell T(z_{p-1}) > lev(wx)$. This is the case for the embedding in Figure 4.5 with $p = 2$. In this case, swap the nested trees $\vec{T}_{z_1}, \vec{T}_{z_2}, \ldots, \vec{T}_{z_{p-1}}$ with $\vec{T}_{y_1}, \vec{T}_{z_2}, \ldots, \vec{T}_{y_i}$ and nest $\vec{T}_w$ around $\vec{T}_{z_{p-1}}$ (now below x) and inside $\vec{T}_{y_{i+1}}$. This allows room for the arc (x, v) without causing a crossing and gives a directed leveled-planar embedding of $\vec{T}$, as desired.

Subcase III. $m\ell P(z_j) > lev(wx)$ *and* $m\ell P(y_i) > lev(wx)$. If $m\ell T(y_i) > lev(wx)$, nest $\vec{T}_w$ between $P(y_i, h(y_i))$ and $P(y_{i+1}, h(y_{i+1}))$. If $m\ell T(z_j) > lev(wx)$, nest $\vec{T}_w$ between $P(z_j, h(z_j))$ and $P(z_{j+1}, h(z_{j+1}))$. In either case, there is room for the arc (x, v) without causing a crossing.

Otherwise, $m\ell T(y_i) \leq lev(wx)$ and $m\ell T(z_j) \leq lev(wx)$, in which case we have a forbidden graph with $M = lev(wx)$.

In all cases, we obtain a leveled-planar embedding of $\vec{T}$. By induction, every tree dag having no subdag in $\mathcal{F}$ is a leveled-planar dag. The theorem follows. □

The following corollary to Theorems 4.1 and 4.2 is a complete characterization of 1-queue tree dags.

COROLLARY 4.3. *If $\vec{T}$ is a tree dag, then $\vec{T}$ is a directed* 1*-queue graph if and only if $\vec{T}$ has no subdag in the set $\mathcal{F}$.*

4.2. Characterization of 1-queue directed cycles. Throughout this section, we assume that $\vec{C} = (V, \vec{E})$ is a cycle dag with covering graph $C = (V, E)$. For notational convenience, we also assume that C is the cycle $u_1, u_2, \ldots, u_n$. Without loss of generality, we may assume that the indegree of node u_1 is 0. Choose a leveling *lev* of $\vec{C} - u_1$. Without loss of generality, we may assume that $1 = lev(u_2) \leq lev(u_n)$. Let $\vec{P} = (V, \vec{E_P})$ be the path dag $\vec{C} - (u_1, u_n)$. Extend *lev* to a leveling of $\vec{P}$, i.e., by making $lev(u_1) = 0$. We start with a partial characterization of 1-queue cycle dags based on the value of $lev(u_n)$, as given in the following lemma.

LEMMA 4.4. *Let $\vec{C}$, $\vec{P}$, and lev be as described above. If $lev(u_n) \geq 3$, then $\vec{C}$ is not a* 1*-queue dag. If $lev(u_n) = 2$, then $\vec{C}$ is a* 1*-queue dag and has a directed arched leveled-planar embedding with the single arch (u_1, u_2). If $lev(u_n) = 1$, then $\vec{C}$ is a* 1*-queue dag if and only if $\vec{C}$ has a directed leveled-planar embedding.*

Proof. We have three cases, based on the value of $lev(u_n)$.

Case 1. $lev(u_n) \geq 3$. To obtain a contradiction, suppose that $\vec{C}$ has a 1-queue layout. By Proposition 1.1, $\vec{C}$ has a directed arched leveled-planar embedding. Recall the terminology and notation from section 1. Let $\mathcal{E}$ be such an embedding with as few arches as possible.

We first argue that no level j has more than one arch. Suppose (x, t_j) and (y, t_j) are level-j arches, where $x <_\sigma y <_\sigma t_j$. By the definition of arches, $x <_\sigma y \leq_\sigma s_j$, so neither x nor any other node below y on ℓ_j has an out-neighbor on ℓ_{j+1}. Hence we can move t_j to the bottommost place on ℓ_{j+1} and still have an arched leveled-planar embedding with two fewer arches than $\mathcal{E}$. Since $\mathcal{E}$ was chosen to have as few arches as possible, we have a contradiction. This contradiction establishes that no level j has more than one arch.

We now define forward and backward arcs with respect to a traversal of C. A *forward arc* is some (u_i, u_{i+1}), where $1 \leq i < m$. A *backward arc* is either (u_1, u_m) or some (u_{i+1}, u_i), where $1 \leq i < m$. Let α be the number of forward arches, β the number of backward arches, γ the number of forward level arcs, and δ the number of backward level arcs. During the traversal of C, every forward level arc moves one vertical line to the right, while every backward level arc moves one vertical line to the left; arches cause no horizontal movement. Since $\vec{C}$ is a cycle dag, $\gamma = \delta$. We know that $\vec{C}$ contains $lev(u_n) - 1 \geq 2$ more forward arcs than backward arcs; that is, $\alpha + \gamma - (\beta + \delta) \geq 2$. Hence $\alpha - \beta \geq 2$.

Suppose (a, b) is a level-j_1 arch, (c, d) is a level-j_2 arch, $j_1 < j_2$, and there are no arches at levels strictly between j_1 and j_2. Then the cyclic order of these nodes in C is a, b, d, c because if the cyclic order was a, b, c, d, then the path between b and c that avoids a, plus the arch (c, d), would make it impossible to put the path from d to a that avoids c in the directed arched leveled-planar embedding. Hence, if (a, b) is a forward arc, then (c, d) is a backward arc and vice versa. Considering all arches, we obtain that $|\alpha - \beta| \leq 1$. This is a contradiction to $\alpha - \beta \geq 2$. This contradiction establishes that $\vec{C}$ does not have a 1-queue layout, as desired.

Case 2. $lev(u_n) = 2$. In this case, we construct a directed arched leveled-planar embedding of $\vec{C}$ in which the arc (u_1, u_2) is the only arch. First, construct a directed leveled-planar embedding of $\vec{C} - u_1$ by placing all level-j nodes on ℓ_j in decreasing order by index from bottom to top. This is the mirror image of the construction in the proof of Corollary 3.2. Note that node u_2 is the topmost node on line ℓ_1, and

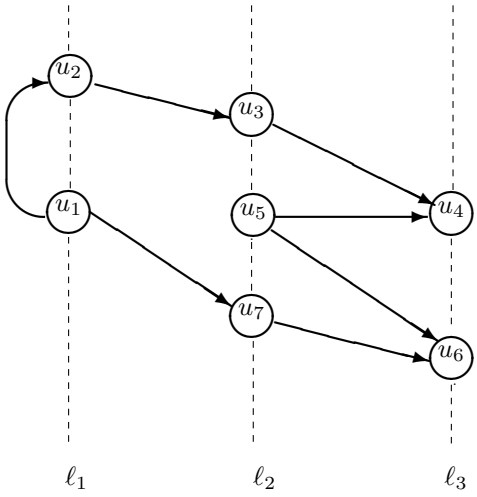


FIG. 4.6. *Construction of a directed arched leveled-planar embedding of $\vec{C}$ with exactly one arch, when $lev(u_n) = 2$.*

node u_n is the bottommost node on line ℓ_2. Now add node u_1 to the embedding as the bottommost node on line ℓ_1. Add arc (u_1, u_n) as a level arc and arc (u_1, u_2) as an arch. Figure 4.6 illustrates the above construction for a cycle dag of 7 nodes. The construction yields a directed arched leveled-planar embedding of $\vec{C}$ with the single arch (u_1, u_2), as desired.

Case 3. $lev(u_n) = 1$. If $lev(u_n) = 1$, then $\vec{C}$ is a leveled dag. By Theorem 4.1, $\vec{C}$ is a 1-queue dag if and only if it has a directed leveled-planar embedding, as desired.

From these three cases, the lemma follows. □

Recall that we assumed, without loss of generality, that $lev(u_n) \geq 1$. Thus the lemma completely characterizes all 1-queue cycle dags. When $lev(u_n) = 1$, $\vec{C}$ is a leveled cycle dag, and we can give a simple property that is equivalent to $\vec{C}$ being a 1-queue dag. Define

$$m = \min\{lev(u_i) \mid 1 \leq i \leq n\},$$
$$M = \max\{lev(u_i) \mid 1 \leq i \leq n\}.$$

$\vec{C}$ contains an *N-dag* if there exist four nodes, u_a, u_b, u_c, and u_d satisfying $1 \leq a < b < c < d \leq n$, such that either $lev(a) = lev(c) = m$ and $lev(b) = lev(d) = M$ or $lev(a) = lev(c) = M$ and $lev(b) = lev(d) = m$. Figure 4.7 shows a path dag in which $lev(u_1) = lev(u_7) = 0$ and $lev(u_3) = lev(u_8) = 2$. If we add the arc (u_{10}, u_9) to this path dag, then we obtain a cycle dag that contains an N-dag. Notice that the cycle dag so obtained is a leveled dag that does not have a directed leveled-planar embedding. Whether $\vec{C}$ contains an N-dag is independent of *lev*.

THEOREM 4.5. *Let $\vec{C}$ be a leveled dag. Then $\vec{C}$ is a 1-queue dag if and only if $\vec{C}$ does not contain an N-dag.*

Proof. First suppose that $\vec{C}$ is a leveled dag that contains an N-dag. To obtain a contradiction, assume that $\vec{C}$ is a 1-queue dag. By Theorem 4.1, $\vec{C}$ has a directed leveled-planar embedding—call it $\mathcal{E}$. Let *lev* be the leveling of $\vec{C}$ induced by $\mathcal{E}$. Let m and M be as defined above. Since $\vec{C}$ contains an N-dag, without loss of generality, we may assume that $lev(u_a) = lev(u_c) = m$, $lev(u_b) = lev(u_d) = M$, and

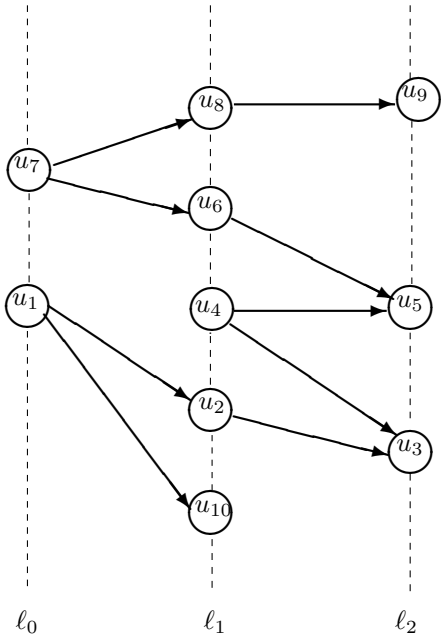


FIG. 4.7. *Illustration of an N-dag.*

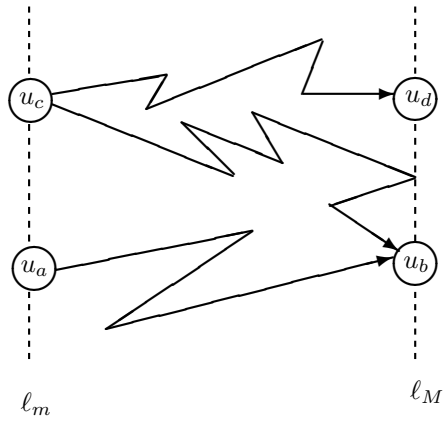


FIG. 4.8. *The path between u_c and u_b partitions the strip between ℓ_m and ℓ_M into disconnected regions.*

$1 \le a < b < c < d \le n$. As seen in Figure 4.8, the strip between the vertical lines ℓ_m and ℓ_M is partitioned into two disconnected regions by the path between u_c and u_b. The nodes u_a and u_d lie in distinct disconnected regions and cannot be connected by a path lying wholly between ℓ_m and ℓ_M without crossing the path between u_c and u_b. Hence, $\vec{C}$ does not have a directed leveled-planar embedding, a contradiction to the existence of $\mathcal{E}$. This contradiction establishes that $QN(\vec{C}) \neq 1$.

Now suppose that $\vec{C}$ is a leveled dag that does not contain an N-dag. We construct a directed leveled-planar embedding of $\vec{C}$, thus showing that $QN(\vec{C}) = 1$. Without

loss of generality, we may assume that $lev(u_1) = m$. Define

$$a = \min\{i \mid lev(u_i) = m\},$$
$$b = \max\{i \mid lev(u_i) = m\},$$
$$c = \min\{i \mid lev(u_i) = M\},$$
$$d = \max\{i \mid lev(u_i) = M\}.$$

Hence $a = 1$, $a \leq b$, and $c \leq d$. We consider three cases based on the relationships between a and b and between c and d.

First, suppose $a = b$. Embed $\vec{P}$ as in the proof of Corollary 3.2. Since there is only one level-m node, (u_1, u_2) is the only arc drawn between ℓ_m and ℓ_{m+1}. Since u_n is on line ℓ_{m+1}, the arc (u_1, u_n) can be added to the embedding without intersecting any arc, yielding a directed leveled-planar embedding of $\vec{C}$.

Now suppose $a \neq b$ and $c = d$. Let $\vec{C}^R$ be the cycle dag gotten by reversing all the arcs in $\vec{C}$. The construction of the last paragraph yields a directed leveled-planar embedding of $\vec{C}^R$, which is the mirror image of a directed leveled-planar embedding of $\vec{C}$.

Finally, suppose $a \neq b$ and $c \neq d$. Without loss of generality, we may assume that $1 = a < b < c < d < n$, since $\vec{C}$ does not contain an N-dag. First, construct a directed leveled-planar embedding $\mathcal{E}_1$ of the path dag $u_1, u_2, \ldots, u_c$, as in the proof of Corollary 3.2. Observe that all level-m nodes in $\vec{C}$ are in $\mathcal{E}_1$, u_1 is the lowest node on ℓ_m, and u_c is the only node on ℓ_M. Second, construct a directed leveled-planar embedding $\mathcal{E}_2$ of the path dag $u_{c+1}, \ldots, u_d$ so that the nodes of level-j are placed on ℓ_j in decreasing order by index. This embedding is the mirror image of the embedding in the proof of Corollary 3.2. Third, combine $\mathcal{E}_1$ and $\mathcal{E}_2$ into a directed leveled-planar embedding $\mathcal{E}_3$ of the path dag $u_1, u_2, \ldots, u_d$ by placing $\mathcal{E}_2$ below $\mathcal{E}_1$ and adding arc (u_{c+1}, u_c). Observe that all level-m and level-M nodes are in $\mathcal{E}_3$, u_1 is the lowest node on ℓ_m, and u_d is the lowest node on ℓ_M. Fourth, construct a directed leveled-planar embedding $\mathcal{E}_4$ of the path dag $u_{d+1}, \ldots, u_n$, as in the proof of Corollary 3.2. Fifth, construct a directed leveled-planar embedding of $\vec{C}$ by placing $\mathcal{E}_4$ below $\mathcal{E}_3$ and adding arcs (u_1, u_n) and (u_{d+1}, u_d).

In all cases, we have a directed leveled-planar embedding of $\vec{C}$. The theorem follows. □

5. Conclusion. In this paper, we have initiated the study of stack and queue layouts of dags and have established the stacknumber and queuenumber of path dags, cycle dags, tree dags, and unicyclic dags. We have also presented forbidden subgraph characterizations of 1-queue tree dags and 1-queue cycle dags. By comparing Theorem 2.1 to Theorem 3.1, one might think that stacks are more powerful than queues for laying out dags. However, in the following examples, we show that the truth is not so simple.

Example. Figure 5.1 shows a dag with 6 nodes whose covering graph is outerplanar. (Bernhart and Kainen [1] show that undirected outerplanar graphs are exactly the 1-stack graphs.) This dag has a unique topological order that yields stacknumber 1 and queuenumber 3. More generally, the path dag $v_1, v_2, \ldots, v_{2n}$ augmented with the arcs (v_i, v_{2n-i+1}) defines a sequence of dags with outerplanar covering graphs with stacknumber 1 and queuenumber n. An analogous sequence for undirected graphs is unknown (see Heath, Leighton, and Rosenberg [4] for discussion).

Example. Figure 5.2 shows a dag with 8 nodes whose covering graph is planar. This dag has a unique topological order that yields stacknumber 4 and queuenumber

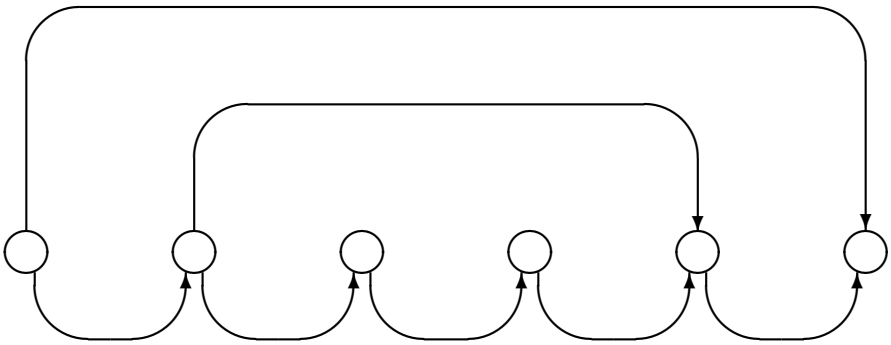


FIG. 5.1. *A dag with stacknumber* 1 *and queuenumber* 3.

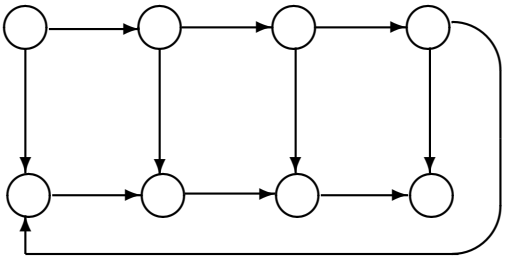


FIG. 5.2. *A dag with stacknumber* 4 *and queuenumber* 2.

2. By expanding the directed grid in this example to the right, we obtain a sequence of dags with planar covering graphs, in which each dag with $2n$ nodes has stacknumber n and queuenumber 2. By way of contrast, Yannakakis [11] shows that every undirected planar graph has stacknumber at most 4. Heath, Leighton, and Rosenberg [4] give an example of a sequence of undirected graphs for which the stacknumber is exponentially larger than the queuenumber.

Our investigation of dags with outerplanar covering graph has left us with the belief that there exists a constant c such that any dag with outerplanar covering graph can be laid out in c or fewer stacks. So we conclude with a conjecture.

CONJECTURE 1. *The stacknumber of the class of directed outerplanar graphs is bounded above by a constant.*

REFERENCES

[1] F. BERNHART AND P. C. KAINEN, *The book thickness of a graph*, J. Combin. Theory Ser. B, 27 (1979), pp. 320–331.

[2] S. N. BHATT, F. R. K. CHUNG, F. T. LEIGHTON, AND A. L. ROSENBERG, *Scheduling tree-dags using FIFO queues: A control-memory trade-off*, J. Parallel Distrib. Comput., 33 (1996), pp. 55–68.

[3] F. R. K. CHUNG, F. T. LEIGHTON, AND A. L. ROSENBERG, *Embedding graphs in books: A layout problem with applications to VLSI design*, SIAM J. Alg. Discrete Methods, 8 (1987), pp. 33–58.

[4] L. S. HEATH, F. T. LEIGHTON, AND A. L. ROSENBERG, *Comparing queues and stacks as mechanisms for laying out graphs*, SIAM J. Discrete Math., 5 (1992), pp. 398–412.

[5] L. S. HEATH AND S. V. PEMMARAJU, *Stack and queue layouts of directed acyclic graphs: Part* II, SIAM J. Comput., to appear.

[6] L. S. Heath and S. V. Pemmaraju, *Stack and queue layouts of posets*, SIAM J. Discrete Math., 10 (1997), pp. 599–625.
[7] L. S. Heath, S. V. Pemmaraju, and C. J. Ribbens, *Sparse Matrix-Vector Multiplication on a Small Linear Array*, Springer-Verlag, Berlin, 1993.
[8] L. S. Heath and A. L. Rosenberg, *Laying out graphs using queues*, SIAM J. Comput., 21 (1992), pp. 927–958.
[9] R. Nowakowski and A. Parker, *Ordered sets, pagenumbers and planarity*, Order, 6 (1989), pp. 209–218.
[10] M. M. Sysło, *Bounds to the page number of partially ordered sets*, in Proc. 15th International Workshop on Graph-Theoretic Concepts in Computer Science, WG '89, M. Nagl, ed., 1989, pp. 181–195.
[11] M. Yannakakis, *Embedding planar graphs in four pages*, J. Comput. System Sci., 38 (1989), pp. 36–67.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1541–1551

DIAGNOSIS OF WIRING NETWORKS: AN OPTIMAL RANDOMIZED ALGORITHM FOR FINDING CONNECTED COMPONENTS OF UNKNOWN GRAPHS*

WEIPING SHI† AND DOUGLAS B. WEST‡

Abstract. We want to find the vertex sets of components of a graph G with a known vertex set V and unknown edge set E. We learn about G by sending an oracle a query set $S \subseteq V$, and the oracle tells us the vertices connected to S. The objective is to use the minimum number of queries to partition the vertex set into components. The problem is also known as interconnect diagnosis of wiring networks in VLSI. We present a deterministic algorithm using $O(\min\{k, \lg n\})$ queries and a randomized algorithm using expected $O(\min\{k, \lg k + \lg\lg n\})$ queries, where n is the number of vertices and k is the number of components. We also prove matching lower bounds.

Key words. randomized algorithm, lower bound, fault diagnosis, graph, component, connection class

AMS subject classifications. 68Q25, 68R10, 05C85, 05C40, 94C12

PII. S0097539795288118

1. Introduction.

We study how to find the vertex sets of components of an unknown undirected graph $G = (V, E)$ on a known vertex set V. Vertices u and v are connected if there is a path between them. The components of G are its maximal connected subgraphs. The connection relation is an equivalence relation on the vertex set V, and the vertex sets of the components are the equivalence classes of the connection relation, also called the *connection classes*. When we say "finding the components," we mean finding the connection classes. In our problem, we are given V but not E. We do not know the number of components or their sizes. The only operation we may use to obtain information about G is to query an oracle. For any query set $S \subseteq V$, the oracle will tell us $Q(S)$, the set of vertices connected to vertices of S

$$Q(S) = \{v \in V : \text{ there exists } u \in S \text{ such that } u \text{ and } v \text{ are connected}\}.$$

Note that the response $Q(S)$ does not identify which vertex in S is connected to each vertex in $Q(S)$. The objective is to find the connection classes using the minimum number of queries.

This problem comes from the interconnect diagnosis of wiring networks of logic circuits. It has applications to design and testing of very large scale integration (VLSI), multichip module (MCM), and printed circuit board (PCB) systems [2, 3, 4, 5, 8]. A wiring network consists of a set of nets, each having one driver and one receiver. The logic value of a good net is controlled by its driver and observed by its receiver. When some nets are involved in a short fault, their receivers all receive the logical OR of the values of their drivers. To diagnose a wiring network, a test engineer sends a test vector of logical 0's and 1's from the drivers and observes the outputs from the receivers. Diagnosing a wiring network is the same as finding the

*Received by the editors June 26, 1995; accepted for publication (in revised form) February 13, 1998; published electronically April 27, 1999.
http://www.siam.org/journals/sicomp/28-5/28811.html

†Department of Computer Science, University of North Texas, Denton, TX 76203 (wshi@cs.unt.edu). The research of this author was supported by NSF grant MIP-9309120.

‡Department of Mathematics, University of Illinois at Urbana-Champaign, Urbana, IL 61801 (west@math.uiuc.edu). The research of this author was supported by NSA/MSP grant MDA904-93-H-3040.

TABLE 1.1
Number of queries required to find connection classes.

Deterministic	$\Theta(\min\{k, \lg n\})$
Randomized	$\Theta(\min\{k, \lg k + \lg\lg n\})$
Nondeterministic	$\Theta(\lg k)$

connection classes of the graph of short faults, and applying test vectors to the nets is the same as querying the oracle for that graph.

Kautz [5] studied the problem for the special case of testing $G = \overline{K}_n$, which he phrased as testing whether there is any short among n nets. Garey, Johnson, and So [3] observed that if we are given partial information about the edge set of G, then finding an optimal algorithm to test $G = \overline{K}_n$ becomes NP-complete (reduction from chromatic number). For our problem of finding all connection classes, Jarwala and Yau [4] provided a heuristic using $\lg n + n - k$ queries, where k is the number of components. There is also a nonadaptive version of the problem where the inputs of all queries are decided before asking the oracle any question [2, 8, 9]. The nonadaptive version is used in applications where the query set is built into the computer hardware and the test is performed automatically. Shi and Fuchs [8] showed that $n-1$ queries are necessary and sufficient to find all connection classes nonadaptively. Shi and Fuchs also presented a recursive version of the deterministic algorithm of section 2, for the case of the interconnect diagnosis problem. Cheng, Lewandowski, and Wu [2] studied a variation of the nonadaptive diagnosis problem where the objective is to report all vertices in components that contain more than one vertex, without having to identify the connection classes. Chen and Hwang [1] studied the problem under a different model, called group testing. In group testing, the inputs of each query are two sets S and T, and the oracle answers yes or no depending on whether some vertex in S is connected to some vertex in T. Kavraki et al. [6] studied the problem of finding connection classes of an unknown graph, where the oracle looks at one entry of the adjacency matrix in each query. Recently, Shi and West [9] studied how to find the connection classes if partial information about the edge set of G is given.

Table 1.1 summarizes our results, where n is the number of vertices and k is the number of components, which is not given as part of the input. No assumption is made on k other than $1 \le k \le n$. All logarithms in this paper are base 2. We measure the query complexity in terms of both the input size n and the number of components k. We present algorithms achieving the upper bounds and prove matching lower bounds. Note that randomization may permit an exponential reduction in the number of queries compared to the deterministic algorithm.

2. Deterministic algorithm. There is a straightforward deterministic algorithm to find the connection classes in k queries: Iteratively pick a vertex v and make a query. Record $Q(\{v\})$ as one class, and delete $Q(\{v\})$ from the vertex set. Each query finds one new class. The upper bound k can be improved when $n < 2^k$. We present in Algorithm 1 and Procedure $\mathcal{D}$ an algorithm that uses $\lceil \lg n \rceil$ queries to find all connection classes. Procedure $\mathcal{D}$ will also be used by the randomized algorithm in section 3.

The algorithm iteratively maintains a *component structure* $P = \{(S_i, R_i) : i = 1, 2, \ldots, t\}$, where $S_1, S_2, \ldots, S_t$ are subsets of vertices that form a partition of V, and $R_1, R_2, \ldots, R_t$ are "chosen" subsets of vertices such that $R_i \subseteq S_i$ and $Q(R_i) = S_i$. This property of the chosen subsets implies that each S_i is a union of connection classes. Algorithm 1 initializes with the component structure $P = \{(V, V)\}$, and then

makes $\lceil \lg n \rceil$ calls to Procedure $\mathcal{D}$ to refine the partition, cutting the sizes of the chosen sets in half at each step. When the algorithm terminates, every R_i is reduced to a single vertex, and therefore every S_i is one connection class.

ALGORITHM 1.
Input: A vertex set V of size n.
Output: Vertex sets of all components of unknown G.
1: $P \leftarrow \{(V, V)\}$.
2: **For** $i \leftarrow 1$ **to** $\lceil \lg n \rceil$ **do**
3: $P \leftarrow \mathcal{D}(P)$.
4: **For** each $(S_j, R_j) \in P$
5: Report S_j as one class.

PROCEDURE $\mathcal{D}(P)$.
Input: A component structure $P = \{(S_j, R_j) : j = 1, 2, \ldots, t\}$.
Output: A refined component structure $P' = \{(S'_j, R'_j) : j = 1, 2, \ldots, t'\}$.
1: **For** $j \leftarrow 1$ **to** t **do**
2: Arbitrarily pick $R'_j \subseteq R_j$ such that $|R'_j| = \lceil \frac{1}{2}|R_j| \rceil$.
3: Perform the single query $\cup R'_j$, with result $S' \leftarrow Q(\cup R'_j)$.
4: $P' \leftarrow \emptyset$.
5: **For** $j \leftarrow 1$ **to** t **do**
6: Let $T_j = S_j \cap S'$.
7: $P' \leftarrow P' \cup \{(T_j, R'_j)\}$.
8: **If** $T_j \neq S_j$ **then** $P' \leftarrow P' \cup \{(S_j - T_j, R_j - T_j)\}$.
9: **Return** P'.

LEMMA 2.1. *Given a component structure in which the largest chosen subset has m vertices, $\lceil \lg m \rceil$ iterations of Procedure $\mathcal{D}$ finds all the connection classes.*

Proof. With each query, we divide the maximum size of the chosen subsets by 2. Hence there are at most $\lceil \lg m \rceil$ queries. To prove that the algorithm works it suffices to show that the property $Q(R_j) = S_j$ is maintained by Procedure $\mathcal{D}$. If so, then when all $|R_j| = 1$, all vertices in S_j are connected to the chosen vertex, and S_j is the connection class containing it.

For any $|R_j| > 1$, the procedure performs a query and splits S_j into T_j and $S_j - T_j$. Because $Q(R_j) = S_j$, the query $Q(\cup R'_j)$ tells us that $Q(R'_j) = T_j$ and there is no connection between T_j and $S_j - T_j$. If $S_j - T_j$ is nonempty, then since all of S_j were connected to R_j, we must have $Q(R_j - T_j) = S_j - T_j$. □

The following theorem is manifest.

THEOREM 2.2. *For any graph with n vertices, Algorithm* 1 *finds all connection classes using $\lceil \lg n \rceil$ queries.*

An algorithm using $2(\min\{k, \lceil \lg n \rceil\})$ queries can be obtained by combining the k-query algorithm and the $\lceil \lg n \rceil$-query algorithm. We alternately perform one query for each of the two algorithms, stopping whenever one of the two algorithms has found all the connection classes.

3. Randomized algorithm. In this section, we present a randomized algorithm that may reduce the number of queries exponentially. First the algorithm calls the randomized Procedure $\mathcal{R}$ for $\lceil \lg \lg n \rceil$ iterations, and then it calls the deterministic Procedure $\mathcal{D}$ to complete the refinement of the partition.

In Algorithm 2 we also maintain a partition and chosen subsets, and the idea is still to partition S_j into T_j connected to R'_j and $S_j - T_j$ connected to $R_j - T_j$. The only difference between Procedure $\mathcal{R}$ and Procedure $\mathcal{D}$ is in step 2, where we randomly pick $\lceil\sqrt{|R|}\rceil$ vertices in $\mathcal{R}$, instead of $\lceil\frac{1}{2}|R|\rceil$ vertices in $\mathcal{D}$. This difference leads to more rapid reduction in the size of the query set.

ALGORITHM 2.
Input: A vertex set V of size n.
Output: Vertex sets of all components of unknown G.
1: $P \leftarrow \{(V, V)\}$.
2: **For** $i \leftarrow 1$ **to** $\lceil \lg \lg n \rceil$ **do**
3: $P \leftarrow \mathcal{R}(P)$.
4: **While** there exists $(S_j, R_j) \in P$ such that $|R_j| > 1$
5: $P \leftarrow \mathcal{D}(P)$.
6: **For** each $(S_j, R_j) \in P$
7: Report S_j as one class.

PROCEDURE $\mathcal{R}(P)$.
Input: A component structure $P = \{(S_j, R_j) : j = 1, 2, \ldots, t\}$.
Output: A refined component structure $P' = \{(S'_j, R'_j) : j = 1, 2, \ldots, t'\}$.
1: **For** $j \leftarrow 1$ **to** t **do**
2: Randomly pick $R'_j \subseteq R_j$ such that $|R'_j| = \lceil\sqrt{|R_j|}\rceil$.
3: Perform the single query $\cup R'_j$, with result $S' \leftarrow Q(\cup R'_j)$.
4: $P' \leftarrow \emptyset$.
5: **For** $j \leftarrow 1$ **to** t **do**
6: Let $T_j = S_j \cap S'$.
7: $P' \leftarrow P' \cup \{(T_j, R'_j)\}$.
8: **If** $T_j \neq S_j$ **then** $P' \leftarrow P' \cup \{(S_j - T_j, R_j - T_j)\}$.
9: **Return** P'.

The correctness of Algorithm 2 follows as in Lemma 2.1. To estimate the number of queries used by Algorithm 2, we need to estimate the size of the largest chosen subset after $\lceil \lg \lg n \rceil$ calls to Procedure $\mathcal{R}$. Define a sequence of random variables $X_0, X_1, \ldots, X_{\lg \lg n}$, where $X_0 = n$, and X_i is the size of the largest chosen subset after i calls to Procedure $\mathcal{R}$.

LEMMA 3.1. *For any positive integer m and real number $\alpha \in [0, 1]$,*

$$\alpha(1 - \alpha)^m < \frac{1}{em},$$

where $e = 2.71828\ldots$ is the base of the natural logarithm.

Proof. The derivative of $f(\alpha) = \alpha(1 - \alpha)^m$ is

$$\frac{df(\alpha)}{d\alpha} = (1 - \alpha)^m - m\alpha(1 - \alpha)^{m-1} = (1 - \alpha)^{m-1}(1 - \alpha - m\alpha).$$

If $f'(\alpha) = 0$, then one of the factors $(1 - \alpha)$ and $(1 - \alpha - m\alpha)$ must be zero, which happens only at $\alpha = 1$ or $\alpha = \frac{1}{m+1}$. Checking the values of f at these points and the

boundary $\alpha = 0$, we find that $f(\alpha)$ reaches its maximum when $\alpha = \frac{1}{m+1}$. Therefore,

$$f(\alpha) \le \frac{1}{m+1}\left(1 - \frac{1}{m+1}\right)^m < \frac{1}{(m+1)}\frac{1}{\left(1-\frac{1}{m+1}\right)}\frac{1}{e} = \frac{1}{em}. \qquad \square$$

LEMMA 3.2. *For the sequence of random variables X_i defined above, the conditional expectation $E[X_i \mid X_{i-1}]$ is bounded by*

$$E[X_i \mid X_{i-1}] < \sqrt{X_{i-1}} \cdot k^2.$$

Proof. Assume after $i-1$ calls to $\mathcal{R}$ we have a component structure $P = \{(S_j, R_j) : j = 1, 2, \ldots, t\}$ in which the largest chosen subset is R_l of size X_{i-1}. At iteration i when we apply $\mathcal{R}(P)$, the pair (S_l, R_l) is split into $(Q(R'_l), R'_l)$, and possibly $(S_l - Q(R'_l), R_l - Q(R'_l))$, where $|R'_l| = \sqrt{X_{i-1}}$. Denote the chosen subset $R_l - Q(R'_l)$ by R''_l. The size of R''_l is a random variable with its value anywhere from 0 to $X_{i-1} - \sqrt{X_{i-1}}$. Consider the following conditional expectation.

$$\begin{aligned} E[\max\{|R'_l|, |R''_l|\} \mid X_{i-1}] &= E[\max\{\sqrt{X_{i-1}}, |R''_l|\} \mid X_{i-1}] \\ &\le E[\sqrt{X_{i-1}} + |R''_l| \mid X_{i-1}] \\ &= \sqrt{X_{i-1}} + E[|R''_l| \mid X_{i-1}]. \end{aligned}$$

Let the k components of G be $G_1, G_2, \ldots, G_k$, and their contributions to R_l have sizes $n_1, n_2, \ldots, n_k$, with $n_j = |V(G_j) \cap R_l|$. The vertices in R''_l belong to the components not hit by the query $Q(R'_l)$. Each successive vertex selected for R'_l has probability at least $n_j/|R_l|$ of belonging to G_j. Hence the probability of missing G_j completely is at most $(1 - n_j/|R_l|)^{\sqrt{|R_l|}}$. When we do miss G_j, it contributes n_j vertices to R''_l. Therefore,

$$\begin{aligned} E[|R''_l| \mid X_{i-1}] &\le \sum_{j=1}^{k} n_j \left(1 - \frac{n_j}{X_{i-1}}\right)^{\sqrt{X_{i-1}}} \\ &= X_{i-1} \sum_{j=1}^{k} \frac{n_j}{X_{i-1}} \left(1 - \frac{n_j}{X_{i-1}}\right)^{\sqrt{X_{i-1}}}. \end{aligned}$$

Letting $\alpha_j = n_j/X_{i-1}$, we have $\sum_{j=1}^{k} \alpha_j = 1$ and $0 \le \alpha_j \le 1$ for $j = 1, 2, \ldots, k$. Using Lemma 3.1 we obtain

$$\begin{aligned} E[|R''_l| \mid X_{i-1}] &\le X_{i-1} \sum_{j=1}^{k} \alpha_j (1-\alpha_j)^{\sqrt{X_{i-1}}} \\ &\le X_{i-1} \cdot k \cdot \max_{0 \le \alpha \le 1} \alpha(1-\alpha)^{\sqrt{X_{i-1}}} \\ &< \frac{X_{i-1} \cdot k}{e\sqrt{X_{i-1}}} = \frac{k\sqrt{X_{i-1}}}{e}. \end{aligned}$$

This computation depends only on $|R_l|$, which equals X_{i-1}. Because the expression grows with $|R_l|$, it also provides an upper bound on the expected size of the larger piece in the refinement of some other R_j. Also, although we don't know which piece will provide the new maximum, certainly its size will be less than the sum of maximum

sizes from all the pieces. These two comments yield an upper bound on $E[X_i \mid X_{i-1}]$ by taking k times the bound from R_l:

$$\begin{aligned} E[X_i \mid X_{i-1}] &\le k \cdot E[\max\{|R_l'|, |R_l''|\} \mid X_{i-1}] \\ &< k \cdot \left(\sqrt{X_{i-1}} + \frac{k\sqrt{X_{i-1}}}{e} \right). \end{aligned}$$

For $k \ge 2$, we conclude that $E[X_i \mid X_{i-1}] < \sqrt{X_{i-1}} \cdot k^2$. □

THEOREM 3.3. *For a graph with n vertices and k components, the expected number of queries used by Algorithm 2 to find all connection classes is $O(\lg k + \lg\lg n)$.*

Proof. Consider the maximum size X_i of the chosen subsets after i iterations. We prove by induction on i that $E[X_i] < n^{2^{-i}} k^4$. This is immediate for $i = 0$. Using Lemma 3.2, the inequality $E[f(X)] \le f(E[X])$ for concave f (such as square root), and the induction hypothesis, we have

$$\begin{aligned} E[X_i] = E(E[X_i|X_{i-1}]) &< E[\sqrt{X_{i-1}} \cdot k^2] \le \sqrt{E[X_{i-1}]} \cdot k^2 \\ &\le \sqrt{n^{2^{-(i-1)}} k^4} \cdot k^2 = n^{2^{-i}} k^4. \end{aligned}$$

With $i = \lceil \lg\lg n \rceil$, we obtain $E[X_{\lg\lg n}] < 2k^4$. From Markov's inequality,

$$\Pr[X_{\lg\lg n} \ge 2k^4 \lg n] < \frac{2k^4}{2k^4 \lg n} = \frac{1}{\lg n}.$$

If $X_{\lg\lg n} < 2k^4 \lg n$, then $\lg(2k^4 \lg n)$ additional queries suffice. With probability at most $\frac{1}{\lg n}$, we are left with chosen subsets greater than $2k^4 \lg n$ after $\lceil \lg\lg n \rceil$ iterations of Procedure $\mathcal{R}$. Procedure $\mathcal{D}$ can resolve these instances with at most $\lceil \lg n \rceil$ further queries. Therefore the expected number of queries used by Algorithm 2 is at most

$$\lg\lg n + \lg(2k^4 \lg n) + \frac{1}{\lg n} \lg n = 2\lg\lg n + 4\lg k + 2. \qquad \square$$

Again, an algorithm using $O(\min\{k, \lg\lg n + \lg k\})$ queries can be obtained by alternating between the k-query algorithm and Algorithm 2.

4. Lower bounds. In this section we obtain optimal lower bounds (within a constant multiplicative factor) on the number of queries for nondeterministic, deterministic, and randomized algorithms.

THEOREM 4.1. *For a graph with k components, every algorithm finding the connection classes uses at least $\lg k$ queries.*

Proof. Since the response to a query $Q(S)$ cuts each subset $U \subseteq V$ that is known to be a union of connection classes into at most two subsets, $U \cap Q(S)$ and $U - Q(S)$, with no edges between them, we need at least $\lg k$ queries to separate the set of vertices into k classes. □

Theorem 4.1 provides a lower bound for every algorithm. A nondeterministic algorithm can guess the connection classes and use $\lceil \lg k \rceil + 1$ queries to verify them.

THEOREM 4.2. *Let A be a deterministic algorithm finding the connection classes of unknown graphs. Over graphs with n vertices and k components, in the worst case A uses at least $\min\{k, \lg n\}$ queries.*

Proof. Consider the following adversary. In response to our first query S, the adversary makes a graph in which S induces a single component if $|S| \le n/2$, or a graph in which $V - S$ induces a single component if $|S| > n/2$. This leaves a subproblem with $k - 1$ components and at least $n/2$ vertices. □

The randomized lower bound is more involved. We first describe Yao's corollary [7] of von Neumann's minimax principle. By considering a matrix game in which the rows correspond to deterministic algorithms and the columns to input instances, Yao observed that the expected performance of the optimal randomized algorithm on the worst input instance for it equals the expected cost of the worst input distribution against the best deterministic algorithm for it. More precisely, let P denote a distribution over deterministic algorithms A; Q denote a distribution over input instances G, and $c(A, G)$ denote the cost of running algorithm A on input instance G. We then have

$$\min_P \max_G E_P[c(A,G)] = \max_Q \min_A E_Q[c(A,G)].$$

Hence, to provide a lower bound for a randomized algorithm, it suffices to prove a lower bound for the expectation of every deterministic algorithm against a particularly bad input distribution.

We will need several lemmas.

LEMMA 4.3 (randomized reduction lemma). *For $i \in \{1,2\}$, let $\mathcal{G}_i$ be a set of input instances, Π_i be a property for $\mathcal{G}_i$, and A_i be a deterministic algorithm for testing $\mathcal{G}_i \in \Pi_i$. Let $B : \mathcal{G}_1 \to \mathcal{G}_2$ be a deterministic or randomized transformation such that $G_1 \in \Pi_1$ if and only if $B(G_1) \in \Pi_2$. Let Q_1 be a distribution over $\mathcal{G}_1$ of a random input G_1, and let Q_2 be the resulting distribution of $B(G_1)$. Under these conditions,*

$$\min_{A_1} E_{Q_1}[c(A_1,G_1)] \le E_{Q_1}[c(B,G_1)] + \min_{A_2} E_{Q_2}[c(A_2,G_2)].$$

Proof. We may view B as an algorithm on the set $\mathcal{G}_1$. Our general cost function c uses an arbitrary cost measure; by $c(B, G_1)$ we mean the cost in this measure of performing the transformation. If B is deterministic, then combining B and A_2 gives a deterministic algorithm for recognizing Π_1. Thus the claim holds for deterministic transformations.

If B is randomized, then combining B and A_2 gives a randomized algorithm for Π_1. The performance of a randomized algorithm is a weighted average of the performance of deterministic algorithms, weighted by some distribution. The combination of B and A_2 costs at least as much as the best randomized algorithm; let P be the best distribution over deterministic algorithms. Since the expectation over P is a convex combination over deterministic algorithms, for each fixed input distribution Q_1 we have the inequality below, which completes the proof:

$$\min_{A_1} E_{Q_1}[c(A_1,G_1)] \le \min_P E_P[E_{Q_1}[c(A_1,G_1)]]. \qquad \square$$

We will apply Yao's corollary to a particular distribution $R(n,k)$ over graphs with $n + o(n)$ vertices and k components, where $k \le \frac{1}{2}\lg\lg n$. To select a graph according to $R(n,k)$, we form cliques $C_1, C_2, \ldots, C_k$ with vertex sets of sizes $n^{\epsilon_1}, n^{\epsilon_2}, \ldots, n^{\epsilon_k}$, respectively. We let $\epsilon_1 = 0$, $\epsilon_2 = 1$ and $\epsilon_3 = 1/2$. For $i \ge 4$, ϵ_i is chosen with the following distribution:

$$\Pr\left[\epsilon_i = \epsilon_{i-1} + \left(\frac{1}{2}\right)^{i-2}\right] = \Pr\left[\epsilon_i = \epsilon_{i-1} - \left(\frac{1}{2}\right)^{i-2}\right] = \frac{1}{2}. \tag{4.1}$$

This generates $\sum_{i=1}^{k} n^{\epsilon_i} = n + o(n)$ vertices in total. Finally, apply a random permutation to the vertex labels to complete the generation of a graph from $R(n,k)$. Figure 4.1 illustrates a selection of the exponents.

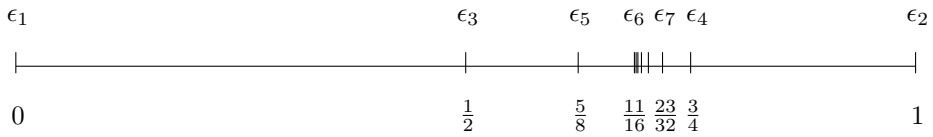


FIG. 4.1. *Example distribution of the exponents.*

In the above definition and the rest of this section, we omit the ceiling function $\lceil\rceil$ on the number of vertices for simplicity. To do so does not affect the claimed bound, since each time a ceiling is omitted the number of vertices is affected by an additive term of at most 1, while the smallest component that might be affected is of size at least $\lg n$.

Given $\epsilon_1, \ldots, \epsilon_k$, let π be the permutation of $\{1, 2, \ldots, k\}$ such that $\epsilon_{\pi(1)} < \epsilon_{\pi(2)} < \cdots < \epsilon_{\pi(k)}$. We call each interval $(\epsilon_{\pi(i)}, \epsilon_{\pi(i+1)}]$ an *ϵ-interval*. The length of $(\epsilon_{\pi(i)}, \epsilon_{\pi(i+1)}]$ is $\epsilon_{\pi(i+1)} - \epsilon_{\pi(i)}$. The real numbers $\epsilon_{\pi(1)}, \ldots, \epsilon_{\pi(k)}$ partition $(0, 1]$ into $k-1$ disjoint ϵ-intervals. Of the $k-1$ ϵ-intervals, there is one with length 2^{-i} for each $i = 1, 2, \ldots, k-3$, and there are two with length $2^{-(k-2)}$ that are consecutive.

LEMMA 4.4. *For any fixed integer $k \geq 3$ and fixed real number $x \in (0, 1]$, under the distribution of $R(n, k)$, the probability that x is in an ϵ-interval of length 2^{-i} is 2^{-i} when $1 \leq i \leq k-3$, and it is $2^{-(i-1)}$ when $i = k-2$.*

Proof. When $k = 3$, every x is in an ϵ-interval of length $1/2$. When $k > 3$, x remains in the interval of length $1/2$ with probability $1/2$; otherwise, x falls into some interval determined by applying the process for $k-1$ steps to an interval of length $1/2$. Multiplying all lengths and probabilities by $1/2$ in that distribution yields the remainder of the specified distribution for k, so the claim holds by induction. □

LEMMA 4.5. *Given positive integers $x_1, x_2, \ldots, x_k$, let $P(x_1, \ldots, x_k)$ be the distribution over graphs consisting of disjoint cliques having $x_1, \ldots, x_k$ vertices that is obtained by assigning vertices to components at random. For any positive integers $n_1, \ldots, n_k, c$, the expected number of queries used by a deterministic algorithm that finds the connection classes against $P_2 = P(n_1 \cdot c, \ldots, n_k \cdot c)$ is at least the expected number of queries used by an optimal deterministic algorithm against $P_1 = P(n_1, \ldots, n_k)$.*

Proof. We exhibit a randomized algorithm B that transforms P_1 to a distribution P' without making any query. By Lemma 4.3, the expected number of queries against P' is at least the expected number against P_1. We then observe that the expected number of queries against P' is the same as the expected number used by the same algorithm against P_2.

Given G_1 drawn from P_1, for each vertex $v_i \in V_1$ we add a clique Q_i having $c-1$ vertices and an edge between v_i and Q_i. We then apply a random permutation σ to the resulting set of vertices to obtain a graph G_2. We have changed each component having n_i vertices in G_1 to a component having $c \cdot n_i$ vertices in G_2, and we did not add any components or make any queries.

Permutation σ guarantees that vertices are assigned to components at random in G_2. Therefore, the probability of a particular partition of the vertices into connection classes depends only on the sizes of the connection classes, which are fixed. Thus the distribution over answers to the connection-class problem is the same for P' as for P_2. Furthermore, the two problems are solved by the same algorithms and with the same expected number of queries, because instances with the same probabilities can be paired up so that the responses to all queries are exactly the same. □

LEMMA 4.6. *Let k be a positive integer satisfying $k \le \frac{1}{2}\lg\lg n$. If F satisfies*

$$F(n,k) \ge 1 + \sum_{i=1}^{k-2} 2^{-i}\left(1 - \frac{1}{\lg n}\right) F(n^{2^{-i}}, k-i-2)$$

for $k \ge 2$, then $F(n,k) \ge k/5$ for all $k \ge 2$.

Proof. If $k \le \frac{1}{2}\lg\lg n$, then

$$k - i - 2 < k - \frac{i}{2} \le \frac{\lg\lg n}{2} - \frac{i}{2} = \frac{\lg\lg n^{2^{-i}}}{2}.$$

Thus we can apply induction on k. When $k = 2$, since empty sums equal 0, the initial condition is $F(n,k) \ge 1 > k/5$. When $k > 2$, the induction hypothesis yields $F(n,i) \ge i/5$ for $2 \le i < k$. Now

$$F(n,k) \ge 1 + \sum_{i=1}^{k-2} \frac{1}{2^i}\left(1 - \frac{1}{\lg n}\right)\frac{k-i-2}{5}.$$

To evaluate the sum, we first prove by induction on j that $\sum_{i=1}^{j-2} 2^{-i}(j-i) = j-2$. This is trivial for $j = 2$ or $j = 3$. For $j > 3$, we have

$$\sum_{i=1}^{j-2}\frac{j-i}{2^i} = \sum_{i=1}^{j-3}\frac{j-1-i}{2^i} + \sum_{i=1}^{j-3}\frac{1}{2^i} + \frac{2}{2^{j-2}} = j-3 + \left(1 - \frac{1}{2^{j-3}}\right) + \frac{1}{2^{j-3}} = j-2.$$

Also applying $\sum_{i=1}^{k-2} 2^{1-i} < 2$, we obtain

$$\begin{aligned} F(n,k) &> 1 + \left(1 - \frac{1}{\lg n}\right)\left(\frac{k-4}{5}\right) \\ &> \frac{k}{5} + \frac{1}{5} - \frac{k-4}{5\lg n} \\ &> \frac{k}{5} + \frac{1}{5} - \frac{\lg\lg n}{10\lg n} > \frac{k}{5}. \end{aligned}$$

□

LEMMA 4.7. *For $k \le \frac{1}{2}\lg\lg n$, the expected number of queries used by any deterministic algorithm A finding connection classes against distribution $R(n,k)$ is $\Omega(k)$.*

Proof. Let V be the set of vertices and $S \ne \emptyset$ be the set of vertices algorithm A picks to make the first query. We consider three cases concerning S, each of which leads to the same recurrence.

Define x by $|S| = n^{1-x}$. In Cases 1 and 2, we suppose that $|S| < n$, and thus $x > 0$. Since $x \in (0,1]$, there exists an ϵ-interval $(\epsilon_{\pi(i)}, \epsilon_{\pi(i+1)}]$ containing x. By Lemma 4.4, the probability is 2^{-j} that the ϵ-interval containing x has length 2^{-j}. Let us consider this possibility.

Intervals of length 2^{-j} are created when ϵ_{j+2} is selected, and ϵ_{j+2} lies between two such intervals. Further choices are made in one of those intervals, so ϵ_{j+2} is the bottom or top of the ϵ-interval of length 2^{-j} that remains. Thus $j+2$ is $\pi(i)$ or $\pi(i+1)$. We prove that in either case, with probability at least $1/\lg n$ we still have $C_{j+3}, \ldots, C_k$ within a single set of our partition.

Case 1. If $j + 2 = \pi(i)$, then

$$\epsilon_{\pi(i)} - 2^{-j} < \epsilon_{j+3}, \epsilon_{j+4}, \ldots, \epsilon_k \le \epsilon_{\pi(i-1)} < \epsilon_{\pi(i)} < x \le \epsilon_{\pi(i+1)}.$$

Consider the probability that S contains no vertex from components $C_{j+3}, C_{j+4}, \ldots, C_k$. With $C = C_{j+3} \cup \cdots \cup C_k$, we have

$$\Pr[S \cap C = \emptyset] = \frac{\binom{|V|-|C|}{|S|}}{\binom{|V|}{|S|}} > \left(\frac{|V|-|C|-|S|+1}{|V|-|S|+1}\right)^{|S|} = \left(1 - \frac{|C|}{|V|-|S|+1}\right)^{|S|}.$$

Since $x > \epsilon_{\pi(i)} \geq 2^{-k+2} \geq 4/\sqrt{\lg n} > 1/\lg n$ and $n^{-1/\lg n} = 1/2$, we have $|S| < n/2$. Using $|V| - |S| + 1 > n/2$, $|C| < 2n^{\epsilon_{\pi(i-1)}}$, and $|S| = n^{1-x}$, we have

$$\Pr[S \cap C = \emptyset] > \left(1 - \frac{4n^{\epsilon_{\pi(i-1)}}}{n}\right)^{n^{1-x}} > 1 - \frac{4}{n^{x-\epsilon_{\pi(i-1)}}} > 1 - \frac{4}{n^{\epsilon_{\pi(i)}-\epsilon_{\pi(i-1)}}}.$$

The middle inequality uses $(1-y)^z > 1 - yz$ for $z > 1$ and $0 < yz < 1$, which is verified by taking natural logarithms and expansions of both sides.

Since $\epsilon_{\pi(i)} - \epsilon_{\pi(i-1)} \geq 2^{-k+2} \geq \frac{4}{\sqrt{\lg n}}$, we have

$$\Pr[S \cap C = \emptyset] > 1 - \frac{1}{n^{4/\sqrt{\lg n}}} = 1 - \frac{1}{2^{4\sqrt{\lg n}}} > 1 - \frac{1}{2^{\lg\lg n}} = 1 - \frac{1}{\lg n}.$$

Thus $1 - 1/\lg n$ is a lower bound on the probability that all of $C_{j+3}, \ldots, C_k$ lie in $V - Q(S)$.

Case 2. If $j + 2 = \pi(i+1)$, then the various definitions yield

$$\epsilon_{\pi(i)} < x \leq \epsilon_{\pi(i+1)} < \epsilon_{\pi(i+2)} \leq \epsilon_{j+3}, \epsilon_{j+4}, \ldots, \epsilon_k < \epsilon_{\pi(i+1)} + 2^{-j}.$$

Let p be the probability that S intersects each of $C_{j+3}, \ldots, C_k$. The set S misses C_l with probability

$$\Pr[S \cap C_l = \emptyset] = \frac{\binom{|V|-|C_l|}{|S|}}{\binom{|V|}{|S|}} < \left(\frac{|V|-|C_l|}{|V|}\right)^{|S|} = \left(1 - \frac{n^{\epsilon_l}}{n+o(n)}\right)^{n^{1-x}}$$
$$< \left(1 - \frac{n^{\epsilon_l}}{2n}\right)^{n^{1-x}} \leq \exp\left(-\frac{n^{\epsilon_l - x}}{2}\right).$$

The last inequality uses $1 - z \leq e^{-z}$ for $0 \leq z \leq 1$. Avoiding each of the bad events separately yields

$$p \geq 1 - \sum_{l=j+3}^{k} \Pr[S \cap C_l = \emptyset] > 1 - k\Pr[S \cap C_{j+3} = \emptyset]$$
$$> 1 - k\exp\left(-\frac{n^{\epsilon_{j+3}-x}}{2}\right) \geq 1 - k\exp\left(-\frac{n^{\epsilon_{\pi(i+2)}-\epsilon_{\pi(i+1)}}}{2}\right).$$

Let $y = n^{\epsilon_{\pi(i+2)}-\epsilon_{\pi(i+1)}}$. Since $\epsilon_{\pi(i+2)} - \epsilon_{\pi(i+1)} \geq 2^{-k+2}$ and $k \leq \frac{1}{2}\lg\lg n$, we have $y > k$ and thus $ky < y^2 < e^{y/2}$ when y is sufficiently large, which yields $ke^{-y/2} < 1/y$. Therefore, $p > 1 - 1/n^{\epsilon_{\pi(i+2)}-\epsilon_{\pi(i+1)}}$. The same computation as in Case 1 now yields $p > 1 - 1/\lg n$.

Case 3. The remaining case is $|S| \geq n$; that is, $x \leq 0$. In Case 2, we proved that when $0 < x < 4/\sqrt{\lg n}$, the probability is at least $1 - 1/\lg n$ that S hits all of $C_{j+3}, \ldots, C_k$. When S becomes even larger, the probability of hitting all these components cannot decrease.

In each of Cases 1–3, we obtain the same recursive lower bound. Given the occurrence of one of these cases, with probability 2^{-j} the probability is at least $1 - 1/\lg n$

that the first query leaves us with components $C_{j+3}, C_{j+4}, \ldots, C_k$ all in $V - Q(S)$ or all in $Q(S)$. The numbers of vertices in $C_{j+3}, C_{j+4}, \ldots, C_k$ are $n^{\epsilon_{j+3}}, n^{\epsilon_{j+4}}, \ldots, n^{\epsilon_k}$, respectively. With $\delta = \epsilon_{j+2} - 2^{-j}$, the components of sizes $n^{\epsilon_{j+3}-\delta}$, $n^{\epsilon_{j+4}-\delta}$, $\ldots$, $n^{\epsilon_k-\delta}$ have the distribution $R(n^{2^{-j}}, k-j-2)$. According to Lemma 4.5, solving the problem for the graph induced by $V - Q(S)$ or $Q(S)$ expects to use at least as many queries as solving it against $R(n^{2^{-j}}, k-j-2)$.

Let $F(n,k)$ be the expected number of queries used by A against $R(n,k)$. We have the following recurrence relation:

$$F(n,k) \geq 1 + \sum_{j=1}^{k-2} \frac{1}{2^j} \cdot \left(1 - \frac{1}{\lg n}\right) \cdot F(n^{2^{-j}}, k-j-2))$$

and $F(n,2) \geq 1$. From Lemma 4.6, $F(n,k) = \Omega(k)$. □

THEOREM 4.8. *For every randomized algorithm finding connection classes of graphs with n vertices and k components, there is a graph in that class for which the expected number of queries used by the algorithm is $\Omega(\min\{k, \lg k + \lg\lg n\})$.*

Proof. If $k \geq \lg n$, then Theorem 4.1 yields $\lg k \geq \lg\lg n$ as a lower bound.

If $\lg n > k > \frac{1}{2}\lg\lg\frac{n}{2}$, consider the input distribution P formed by starting with a sample from $R(\frac{n}{2}, \frac{1}{2}\lg\lg\frac{n}{2})$ and then adding $k - \frac{1}{2}\lg\lg\frac{n}{2}$ arbitrary components so that the total number of vertices is n. Now we have a distribution over graphs with k components and n vertices. By Lemma 4.3, the expected number of queries used against distribution P is at least the expected number of queries used against distribution $R(\frac{n}{2}, \frac{1}{2}\lg\lg\frac{n}{2})$. By Lemma 4.7, this is $\Omega(\lg\lg n)$.

If $k \leq \frac{1}{2}\lg\lg\frac{n}{2}$, we start with $R(n/2, k-1)$ and add a single component to reach a total of n vertices. Lemma 4.3 yields an expected cost of at least the expected cost against $R(n/2, k-1)$. By Lemma 4.7, this is $\Omega(k)$.

By Theorem 4.1, $\lg k$ is always a lower bound. This completes the proof. □

Acknowledgments. The authors thank Edward Reingold and Steve Tate for helpful discussions and the referees for improving the presentation.

REFERENCES

[1] C. C. CHEN AND F. HWANG, *Detecting and locating electrical shorts using group testing*, IEEE Trans. Circuits Systems, 36 (1989), pp. 1113–1116.

[2] W.-T. CHENG, J. L. LEWANDOWSKI AND E. WU, *Optimal diagnostic methods for wiring interconnects*, IEEE Trans.Computer-Aided Design, 11 (1992), pp. 1161–1166.

[3] M. GAREY, D. JOHNSON, AND H. SO, *An application of graph coloring to printed circuit testing*, IEEE Trans. Circuits Systems, 23 (1976), pp. 591–599.

[4] N. JARWALA AND C. W. YAU, *A new framework for analyzing test generation and diagnosis algorithms for wiring interconnect*, in Proceedings of the IEEE International Testing Conference, 1989, pp. 63–70.

[5] W. H. KAUTZ, *Testing for faults in wiring networks*, IEEE Trans. Comput., 23 (1973), pp. 358–363.

[6] L. KAVRAKI, J.-C. LATOMBE, R. MOTWANI AND P. RAGHAVAN, *Randomized query processing in robot motion planning*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, 1995, pp. 353–362.

[7] R. MOTWANI AND P. RAGHAVAN, *Randomized Algorithms*, Cambridge University Press, Cambridge, 1995.

[8] W. SHI AND W. K. FUCHS, *Optimal interconnect diagnosis of wiring networks*, IEEE Trans. VLSI Systems, 3 (1995), pp. 430–436.

[9] W. SHI AND D. B. WEST, *Optimal structural diagnosis of wiring networks*, in Proceedings of the 27th IEEE International Symposium on Fault Tolerant Computing, 1997, pp. 162–169.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1552–1575

PRODUCT RANGE SPACES, SENSITIVE SAMPLING, AND DERANDOMIZATION*

HERVÉ BRÖNNIMANN†, BERNARD CHAZELLE‡, AND JIŘÍ MATOUŠEK§

Abstract. We introduce the concept of a *sensitive ε-approximation* and use it to derive a more efficient algorithm for computing ε-nets. We define and investigate *product range spaces*, for which we establish sampling theorems analogous to the standard finite *VC-dimensional* case. This generalizes and simplifies results from previous works. Using these tools, we give a new deterministic algorithm for computing the convex hull of n points in $\mathbb{R}^d$. The algorithm is obtained by derandomization of a randomized incremental algorithm, and its running time of $O(n \log n + n^{\lfloor d/2 \rfloor})$ is optimal for any fixed dimension $d \geq 2$.

Key words. convex hull, deterministic algorithm, optimal

AMS subject classifications. 52B55, 68Q20

PII. S0097539796260321

1. Introduction. During the last decade, randomized algorithms have been proposed as an efficient and elegant solution to several geometric problems [12, 13]. Derandomization aims at producing deterministic algorithms whose running times are within a constant factor of the running times of their randomized counterpart [9, 19, 23]. This process has successfully produced several geometric algorithms whose complexities are the best among those of the existing deterministic algorithms for the problems considered [5, 6, 7, 10]. For instance, the problem of computing the convex hull of n points in $\mathbb{R}^d$ is solved deterministically in [7] in time $O(n^{\lfloor d/2 \rfloor})$ for any $d \geq 4$, which is optimal. In section 3, we describe a new deterministic algorithm for computing the convex hull of n points in $\mathbb{R}^d$. Its running time of $O(n \log n + n^{\lfloor d/2 \rfloor})$ is optimal in any fixed dimension d. It uses a method similar to the one given in [7], but it is arguably simpler. Furthermore, there is no need to treat the two- and three-dimensional cases separately, as is done in [7].

Deterministic constructions of ε-nets and ε-approximations [10, 18, 20, 21] play a key role in the derandomization of probabilistic geometric algorithms [5, 6, 7, 9, 10, 19], and they also do for our convex hull algorithm. A *range space* $\Sigma = (X, \mathcal{R})$ is a pair of a set X and a collection $\mathcal{R}$ of subsets of X [17]. An *ε-approximation* for Σ is a subset A of X such that

$$\left| \frac{|R|}{|X|} - \frac{|R \cap A|}{|A|} \right| \leq \varepsilon$$

*Received by the editors August 14, 1996; accepted for publication (in revised form) May 16, 1997; published electronically April 27, 1999. A preliminary version of this paper appeared in *Proc. 34th IEEE Symposium on Foundations of Computer Science*, Palo Alto, CA, 1993, pp. 400–409.
http://www.siam.org/journals/sicomp/28-5/26032.html

†INRIA Sophia Antipolis, BP. 93, 06902 Sophia Antipolis, Cedex, France (herve.bronnimann@sophia.inria.fr). The research of this author was supported in part by NSF grant CCR-90-02352 and Ecole Normale Supérieure de Paris.

‡Department of Computer Science, Princeton University, Princeton, NJ 08544 (chazelle@cs.princeton.edu). This research was supported in part by NSF grant CCR-90-02352 and the Geometry Center, University of Minnesota, an STC funded by NSF, DOE, and Minnesota Technology, Inc.

§Department of Applied Mathematics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic. The research of this author was supported by Humboldt Research Fellowship, Czech Republic grant GAČR 0194, and Charles University grants 193,194.

for every set R in $\mathcal{R}$. To be an *ε-net*, A only needs to intersect every set R in $\mathcal{R}$ whose size is greater than $\varepsilon|X|$. It is a classical result [17] that the existence of small subsets having these properties is linked to the finiteness of a parameter of the set system, called its *VC-dimension*. Namely, if the range space has finite VC-dimension d, there exists an ε-net of size $O(d\varepsilon^{-1}\log(d\varepsilon^{-1}))$ and an ε-approximation of size $O(d\varepsilon^{-2}\log(d\varepsilon^{-1}))$ (see section 2).

In geometric applications, it is common to use the range space $(X, \mathcal{R})$ consisting of a set X of hyperplanes and the collection $\mathcal{R}$ of all the subsets of X consisting of the hyperplanes stabbed by a line segment. By definition, an ε-approximation allows us to estimate how many hyperplanes separate any two points (with a level of accuracy depending on ε). It was shown in [6] that an ε-approximation can also be used to estimate how many vertices of the arrangement formed by X lie within a given simplex: this feature is essential in the recent work on point location [6], convex hull [7], and weak ε-nets for convex sets [8].

We generalize this idea by introducing the notion of a *product range space*. We discuss the problem of sampling such a space, and we explain the apparent paradox that product range spaces can be sampled even though they may have unbounded VC-dimension. We prove that the product of finite VC-dimensional range spaces can be sampled almost as efficiently as the original spaces, meaning that they admit ε-approximations and ε-nets of a size polynomial in $1/\varepsilon$ and, most importantly, independent of the size of the range spaces. We specialize these sampling theorems to a geometric setting, and we build tools for numerically integrating functions defined over the vertices of an arrangement of hyperplanes.

We also introduce the notion of *sensitive* sampling. Formally, we say that a subset $A \subseteq X$ is a *sensitive ε-approximation* for Σ if

$$\left| \frac{|R|}{|X|} - \frac{|R \cap A|}{|A|} \right| \leq \frac{\varepsilon}{2}\left(\sqrt{\frac{|R|}{|X|}} + \varepsilon\right)$$

for every set R in $\mathcal{R}$. The bound on the right-hand side may appear strange, but it arises naturally from what we expect of a random sample. Observe that a sensitive ε-approximation is at once both an ε-approximation and an ε^2-net. For a set system with finite VC-dimension d, we show the existence of a sensitive ε-approximation of size $O(d\varepsilon^{-2}\log(d\varepsilon^{-1}))$. We also modify an algorithm of [10] for computing ε-approximations so that it computes sensitive ε-approximations. If the underlying range space Σ has VC-dimension d, then, under standard computational assumptions given in section 2, a sensitive $(1/r)$-approximation of size $O(dr^2\log(dr))$ can be computed in time $O(d)^{3d}r^{2d}\log^d(dr)|X|$. This gives an algorithm for computing a $(1/r)$-net of size $O(dr\log(dr))$ for $(X, \mathcal{R})$ in time $O(r^d\log^d r)|X|$ time, which significantly improves on the bound $O(r^{2d}\log^d r)|X|$ given in [10].

In a preliminary version of this paper, we also claimed an $O(n\log^3 n)$-time algorithm for deterministically computing the diameter of n points in 3-space. This part contained an error (kindly pointed out to us by E. Ramos). In the meantime, a simpler deterministic algorithm with the same time complexity was found by Amato, Goodrich, and Ramos [3]. They use ideas akin to ours, but they use sampling over a different geometric range space $(X, \mathcal{R}')$ that contains ours: given a set X of hyperplanes, a range in $\mathcal{R}'$ consist of all the hyperplanes of X that intersect a given simplex of any dimension. We refer to their paper for a correct description of the algorithm. For further developments in the derandomization of geometric algorithms, we also refer to [4, 5].

2. Terminology and sampling theorems. In this section we consider general range spaces. First we review standard definitions and facts [17, 20]. A range space is a set system (or equivalently, a hypergraph), whose elements are called *points* and whose sets are called *ranges.* Let $\Sigma = (X^*, \mathcal{R}^*)$ be a (possibly infinite) range space. If $Y \subseteq X^*$, we denote by $(Y, \mathcal{R}^*|_Y)$ the *subspace induced by* Y, where $\mathcal{R}^*|_Y = \{\, R \cap Y : R \in \mathcal{R}^* \,\}$. A subset $Y \subseteq X^*$ is *shattered* (by $\mathcal{R}^*$) if $\mathcal{R}^*|_Y = 2^Y$. The maximum size of any shattered subset of X^* is called the VC-dimension of Σ; note that it can be infinite. We define the *shatter function* π_Σ of Σ as follows: $\pi_\Sigma(m)$ is the maximum possible number of sets in the subsystem of $(X^*, \mathcal{R}^*)$ induced by any m-point subset of X^*. It is well known that the shatter function of a range space of VC-dimension d is bounded $\Phi(m, d) = \binom{m}{0} + \cdots + \binom{m}{d}$, which is less that $m^d + 1$ (see, for instance, [2]). Conversely, if the shatter function is bounded by a polynomial, then the VC-dimension is bounded by a constant.

In practice, we usually deal with finite subsystems of a range space. Let X be a finite subset of X^*, and let $\mathcal{R}$ be a shorthand for $\mathcal{R}^*|_X$; by abuse of terminology we still call the pair $(X, \mathcal{R})$ a range space. As we mentioned earlier, given any $0 < \varepsilon < 1$, a subset $A \subseteq X$ is called an ε-approximation for the range space $(X, \mathcal{R})$ if

$$\left| \frac{|R|}{|X|} - \frac{|R \cap A|}{|A|} \right| \leq \varepsilon$$

for each $R \in \mathcal{R}$. A subset $N \subseteq X$ is an ε-net for $(X, \mathcal{R})$ if $|R| > \varepsilon |X|$ implies that $R \cap N \neq \emptyset$. An ε-approximation is also an ε-net, but the converse is false in general.

For instance, consider the range space $(X^*, \mathcal{R})$ mentioned in the introduction, where X^* is a set of hyperplanes in $\mathbb{R}^d$ and $\mathcal{R}^*$ is the collection of all the subsets of X^* consisting of the hyperplanes stabbed by a given line segment. Pick a finite subset X of m hyperplanes. Then it can be easily verified that the subsets of X^* consisting of the hyperplanes stabbed by two line segments are identical if the endpoints of the segments can be paired up so that a pair lie in the same cell of the arrangement of X. There are $O(m^d)$ such cells, therefore the shatter function $\pi_\Sigma(m)$ of $(X^*, \mathcal{R})$ is bounded above by $O(m^{2d})$. This in turn implies that the range space has finite VC-dimension.

Efficient deterministic constructions of ε-nets and ε-approximations were given in [20] for the particular range space described above; see also [10] for slightly simpler proofs that are expressed in terms of general range spaces. Let d be the VC-dimension of Σ, or for that matter, any constant such that $\pi_\Sigma(m) = O(m^d)$. We assume that the range space admits a *subsystem oracle of dimension* d, meaning that, given any $Y \subseteq X$, all the sets of $\mathcal{R}|_Y$ can be computed explicitly in time $O(|Y|^{d+1})$. This assumption is justified in practice, as can be checked for instance on the range space given above: one may simply construct the arrangement of Y, choose a point inside each cell, and each pair of points yields a segment s and a range R_s of the hyperplanes stabbed by s; moreover, each range is enumerated exactly once in this process. Given any $r > 1$, one can in time $O(d)^{3d} r^{2d} \log^d(dr)|X|$ compute a $(1/r)$-approximation for $(X, \mathcal{R})$ of size $O(dr^2 \log(dr))$ and a $(1/r)$-net of size $O(dr \log(dr))$. These time bounds are linear in $|X|$ if r is a constant.

2.1. Sensitive approximations. Let $\Sigma = (X, \mathcal{R})$ be a range space of a dimension bounded by a constant d. It is known that if one wants to get a $(1/r)$-approximation for Σ, it suffices to pick a random sample $A \subseteq X$ of size $O(r^2 \log r)$. Such a sample, however, has still better approximation properties if we are only interested in small ranges. The fact that A is, with high probability, a $(1/t)$-net for Σ

with t being almost r^2 can be seen as a manifestation of this phenomenon. If we look at the dependence of the error with which a random sample approximates a range on the size of that range, we arrive at the following definition. A subset $A \subseteq X$ is a sensitive ε-approximation for Σ if

$$\left| \frac{|R|}{|X|} - \frac{|R \cap A|}{|A|} \right| \leq \frac{\varepsilon}{2} \left(\sqrt{\frac{|R|}{|X|}} + \varepsilon \right)$$

for every set $R \in \mathcal{R}$. In [5], it is shown that a random sample of size $O(dr^2 \log(dr))$ possesses the sensitive $(1/r)$-approximation property. It follows from the definition that a sensitive ε-approximation is an ε-approximation as well as an ε^2-net. By this observation the next result gives an immediate improvement over the $O(d)^{3d} r^{2d} \log^d(dr)|X|$-time construction of a $(1/r)$-net given in [20], while at the same time keeping the same size bound.

THEOREM 2.1. *Let $(X, \mathcal{R})$ be a range space with a subsystem oracle of dimension d. Given any $r > 1$, in time $O(d)^{3d} r^{2d} \log^d(dr)|X|$ one can compute a sensitive $(1/r)$-approximation for $(X, \mathcal{R})$ of size $O(dr^2 \log(dr))$.*

COROLLARY 2.2. *Let $(X, \mathcal{R})$ be a range space with a subsystem oracle of dimension d. Given any $r > 1$, in time $O(d)^{3d} r^d \log^d(dr)|X|$ one can compute a $(1/r)$-net for $(X, \mathcal{R})$ of size $O(dr \log(dr))$.*

Proof of Theorem 2.1. We begin with a restriction of Theorem 2.1 to the case where ε is very small. We then adapt a recursive construction given in [10] to generalize this result and establish the theorem.

LEMMA 2.3. *Let $(X, \mathcal{R})$ be a range space of finite VC-dimension, where $|X| = n$ is even and large enough, and let $m = |\mathcal{R}|$. In $O(nm)$ time it is possible to compute a sensitive ε-approximation for $(X, \mathcal{R})$ of size $n/2$ for $\varepsilon = 15\sqrt{\ln(6m+6)/n}$.*

Proof. We select a random sample $A_1 \subseteq X$ of expected size $n/2$ by picking every element independently with probability $1/2$. Tail estimates show that A_1 (or its complement $X \setminus A_1$) has the sensitive ε-approximation property with high probability, to be made precise below. To obtain an approximation of size exactly $n/2$, we then show how to trim some elements from the bigger of A_1 and $X \setminus A_1$ while keeping the trimmed set a sensitive ε-approximation.

Given $0 < p < 1$, let $x_1, \ldots, x_n$ be independent random variables, each equal to $p-1$ with probability p, respectively, equal to p with probability $1-p$. The following tail estimate can be found in [2]:

$$\text{Prob}\left[\left| \sum_{i=1}^{n} x_i \right| > \Delta \right] < 2e^{-2\Delta^2/n}.$$

Select a subset $A_0 \subseteq X$ by picking each element of X with probability p. Define an auxiliary function

$$\Delta(x) = \sqrt{x \ln(6m+6)/2},$$

and let $\mathcal{R}' = \mathcal{R} \cup \{X\}$. Given $R \in \mathcal{R}'$, the previous tail estimate indicates that, for our choice of Δ, the following holds with a probability greater than $1 - 1/(3m+3)$:

$$\left| |R \cap A_0| - p|R| \right| \leq \Delta(|R|). \tag{2.1}$$

Assume that A_0 satisfies (2.1) for each $R \in \mathcal{R}'$. We call such a subset *p-good* for $(X, \mathcal{R}')$: note that a random A_0 is p-good with probability of at least $2/3$. Set

$p = 1/2$, and let A_1 be the larger of the two sets A_0 and $X \setminus A_0$. Trivially, A_1 consists of at least $n/2$ elements and is p-good. As an effect of adding X as a range, A_1 also has little more than $n/2$ elements, namely, at most $n/2 + \Delta(n)$.

We now want to remove some elements from A_1 so that it has exactly $n/2$ elements (this exact halving will be convenient in the forthcoming algorithm). If our goal was an ε-approximation only, we could remove an appropriate number of elements quite arbitrarily (as it is done in [20]). Removing arbitrary elements could, however, destroy the sensitive ε-approximation properties for small ranges, so we choose the elements to remove more carefully—we use a suitable random sample from A_1.

Let $q = 4\Delta(n)/n$; note that the finite VC-dimension hypothesis implies $m = n^{O(1)}$, so $q < 1$ for n large enough. With probability of at least $2/3$, a random sample $A_2 \subseteq A_1$ (with each element chosen independently with probability q) is a *q-good* subset for $(A_1, \mathcal{R}'|_{A_1})$. For such an A_2, since $A_1 = X \cap A_1$ is a range in $\mathcal{R}'|_{A_1}$, we have

$$|A_2| \geq q|A_1| - \Delta(|A_1|) \geq \frac{qn}{2} - \Delta(n) \geq \Delta(n) \geq |A_1| - \frac{n}{2},$$

and therefore we can pick $|A_1| - n/2$ elements in A_2 (any of them) and remove them from A_1, thus producing a subset $A \subseteq X$ of size $n/2$.

Note that our probabilistic construction of A can be derandomized in a straightforward fashion by using the method of conditional probabilities of Alon and Spencer [2], Raghavan [23], and Spencer [24]. With a little care, this can be accomplished in $O(nm)$ time; see [3, 20] for a similar construction.

It remains to show that A is a sensitive ε-approximation for $(X, \mathcal{R})$ for a choice of $\varepsilon = 15\sqrt{\log(6m+6)/n}$. We have

$$\begin{aligned}\left| |R \cap A| - \frac{|R|}{2} \right| &\leq \Big| |R \cap A| - |R \cap A_1| \Big| + \left| |R \cap A_1| - \frac{|R|}{2} \right| \leq |R \cap A_2| + \Delta(|R|) \\ &\leq q|A_1 \cap R| + 2\Delta(|R|) \leq \frac{q|R|}{2} + (q+2)\Delta(|R|)\,.\end{aligned}$$

As $q < 1$ for n large enough and $q|R|/2 = 2|R|\Delta(n)/n \leq 2\Delta(|R|)$, we obtain $\Big||R \cap A| - |R|/2\Big| \leq 5\Delta(|R|)$, and hence,

$$\left| \frac{|R \cap A|}{|A|} - \frac{|R|}{n} \right| \leq \frac{2}{n}\, 5\Delta(|R|) = \frac{10}{n}\sqrt{|R| \ln(6m+6)/2} \leq \frac{\varepsilon}{2}\sqrt{\frac{|R|}{n}},$$

which proves Lemma 2.3. □

Sensitive approximations can be *refined* (Lemma 2.4) and *composed* (Lemma 2.5) in a fashion similar to standard ε-approximations [10].

LEMMA 2.4. *If A is a sensitive ε-approximation for a range space $(X, \mathcal{R})$ and B is a sensitive δ-approximation for $(A, \mathcal{R}|_A)$, then B is a sensitive $(\varepsilon + 2\delta)$-approximation for $(X, \mathcal{R})$.*

Proof. Consider a range $R \in \mathcal{R}$. For short, we write $\rho_X = |R|/|X|$, $\rho_A = |R \cap A|/|B|$, $\rho_B = |R \cap B|/|B|$. We have

$$|\rho_X - \rho_B| \leq |\rho_X - \rho_A| + |\rho_A - \rho_B| \leq \frac{\varepsilon}{2}\left(\sqrt{\rho_X} + \varepsilon\right) + \frac{\delta}{2}\left(\sqrt{\rho_A} + \delta\right)$$

by the definition of a sensitive ε-approximation. We estimate

$$\sqrt{\rho_A} \leq \sqrt{\rho_X + \frac{\varepsilon}{2}\left(\sqrt{\rho_X} + \varepsilon\right)} \leq \sqrt{\rho_X} + \sqrt{\frac{\varepsilon}{2}\left(\sqrt{\rho_X} + \varepsilon\right)} \leq \sqrt{\rho_X} + \frac{\varepsilon/2 + \sqrt{\rho_X} + \varepsilon}{2}\,,$$

where we used the AG-mean inequality $\sqrt{ab} \le (a+b)/2$ in the last step. With this estimate, we calculate

$$|\rho_X - \rho_B| \le \frac{\varepsilon}{2}\left(\sqrt{\rho_X} + \varepsilon\right) + \frac{\delta}{2}\left(\frac{3}{2}\sqrt{\rho_X} + \frac{3}{4}\varepsilon + \delta\right) \le \frac{\varepsilon + 2\delta}{2}\left(\sqrt{\rho_X} + \varepsilon + 2\delta\right).$$

Since this is true for any $R \in \mathcal{R}$, the proof is complete. ∎

LEMMA 2.5. *Let $(X, \mathcal{R})$ be a range space, and let $\{X_i\}_{1 \le i \le m}$ be a partition of X into m equal-size subsets. If A_i is a sensitive ε-approximation for $(X_i, \mathcal{R}|_{X_i})$ and all the A_i's have the same size, then $A = \cup_i A_i$ is a sensitive ε-approximation for $(X, \mathcal{R})$.*

Proof. Consider any range $R \in \mathcal{R}$. For short, we put $\rho_{X_i} = |R \cap X_i|/|X_i|$, $\rho_{A_i} = |R \cap X_i|/|X_i|$, $\rho_X = |R|/|X|$, and $\rho_A = |R \cap A|/|A|$. Since the X_i's are disjoint, we have $\rho_X = \frac{1}{m}\sum_{i=1}^m \rho_{X_i}$ and $\rho_A = \frac{1}{m}\sum_{i=1}^m \rho_{A_i}$, where the factor $\frac{1}{m}$ accounts for the difference in the denominators of ρ_X and the ρ_{X_i}'s. Therefore,

$$|\rho_X - \rho_A| \le \frac{1}{m}\sum_{i=1}^m |\rho_{X_i} - \rho_{A_i}| \le \frac{\varepsilon}{2}\left(\frac{1}{m}\sum_{i=1}^m \sqrt{\rho_{X_i}} + \varepsilon\right) \le \frac{\varepsilon}{2}\left(\sqrt{\rho_X} + \varepsilon\right),$$

where the last inequality follows by the concavity of the square root function. ∎

We are now ready to build a sensitive $(1/r)$-approximation for any value of r. The algorithm is almost identical to the construction of nonsensitive ε-approximations given in [10]. We begin with a simplifying observation. If $n = |X|$ is not a power of two, let us pad X by adding up to $n-1$ artificial points so as to obtain a power of two. This gives us a new range space $\Sigma' = (X \cup X_0, \mathcal{R} \cup \{X_0\})$; note that the set X_0 of artificial points is added as a range. Let A be a sensitive $(\varepsilon/6)$-approximation for this new range space. It is not hard to show that $B = X \cap A$ is a sensitive ε-approximation for $(X, \mathcal{R})$. Here are the details: Given any $R \in \mathcal{R}$, we have

$$\begin{aligned}\left|\frac{|R|}{|X|} - \frac{|R \cap B|}{|B|}\right| &= \frac{|X \cup X_0|}{|X|}\left|\frac{|R|}{|X \cup X_0|} - \frac{|R \cap B|}{|B|} \cdot \frac{|X|}{|X \cup X_0|}\right| \\ &\le 2\left(\left|\frac{|R|}{|X \cup X_0|} - \frac{|R \cap B|}{|A|}\right| + \frac{|R \cap B|}{|B|}\left|\frac{|X|}{|X \cup X_0|} - \frac{|B|}{|A|}\right|\right). \qquad (2.2)\end{aligned}$$

Using the sensitive $(\varepsilon/6)$-approximation property of A, we get that the difference in the first absolute value is at most $\frac{\varepsilon}{12}\left(\sqrt{|R|/|X|} + \varepsilon/6\right) \le \varepsilon/6$, and so is the difference in the second absolute value. Substituting this and the trivial estimate $|R \cap B|/|B| \le 1$ into (2.2), we obtain

$$\frac{|R \cap B|}{|B|} \le \frac{|R|}{|X|} + \frac{2\varepsilon}{3}.$$

Substituting this improved upper bound into (2.2) then yields

$$\left|\frac{|R|}{|X|} - \frac{|R \cap B|}{|B|}\right| \le \frac{\varepsilon}{2}\left(\sqrt{\frac{|R|}{|X|}} + \varepsilon\right),$$

which shows that B is a sensitive ε-approximation. This allows us to assume that n is now a power of two.

We begin with one piece of terminology. Applying Lemma 2.3 to an even-sized subset $Y \subseteq X$ is called *halving* Y. Note that this results in a sensitive $h(|Y|)$-approximation, where

$$h(t) = 15\sqrt{\log(6t^d + 6)/t}.$$

It is easy to see that if $t = \Omega(d \log d)$, we have $h(t) < 1$ and $h(2t) \le \frac{3}{4}h(t)$. Moreover, it is also not hard to show that $h(dr^2 \log(dr)) = O(1/r)$ for any $r > 1$. We are now ready to describe the algorithm for constructing a sensitive $(1/r)$-approximation for $(X, \mathcal{R})$.

To begin, we divide up the set X into subsets of size 2^k (for some appropriate parameter k) and we associate each subset with the leaves of a complete binary tree. Next, we process the tree bottom-up level by level. At each internal node, we merge together the two sets associated with its children, and if the level of the node is not divisible by $d + 2$ (leaves being at level 0), we halve the union. The resulting set is *associated* with the node in question. Once the tree is completely processed, i.e., the set associated with its root has been computed, we say that the first phase is over and we move on to the second phase. We take the set associated with the root and we keep halving it until its size is equal to $c_1 dr^2 \log(dr)$ for some appropriate constant $c_1 > 0$.

The union of all the sets associated with the nodes at level i constitutes a sensitive ε_i-approximation for some ε_i, which we call the *error* at level i. During the first $d+1$ levels, each individual set remains of size 2^k (since halving and merging alternate). Note that halving is applied to sets of size 2^{k+1}. By Lemmas 2.4 and 2.5, the total error after the first $d + 1$ levels is $2(d + 1)h(2^{k+1})$. At level $d + 2$, no additional error is incurred since we skip the halving step. The next $d + 1$ steps are similar to the first batch of $d + 1$, except that the size of the individual sets has now doubled; thus, the total error incurred up to level $2d+3$ is $2(d+1)\left(h(2^{k+1}) + h(2^{k+2})\right)$. From level to level, the error follows a geometrically decreasing series, so the total error incurred at the end of the first phase is $O(d) \times h(2^k)$. If we choose $2^k = c_2 d^3 r^2 \log(dr)$, for some constant c_2 large enough, this makes the error at most $1/2r$. In the second phase, the error is still bounded by a geometrically increasing series whose last term is $O(h(c_1 dr^2 \log(dr)))$, meaning that the additional error contributed by the second phase is $O(h(c_1 dr^2 \log(dr)))$. Choosing c_1 large enough keeps this error under $1/2r$. Combining the two phases shows that the final set, which is of the desired size $O(d)r^2 \log(dr)$, is a sensitive $(1/r)$-approximation for $(X, \mathcal{R})$.

What is the running time of the algorithm? In the first halving step, we apply Lemma 2.3 to $|X|/2^k$ sets of size $2^k = O(d^3)r^2 \log(dr)$ each; the total time needed for these operations is $O(d)^{3d} r^{2d} \log^d(dr)|X|$. In the next $d + 1$ halving steps, the sets remain of the same size but their numbers decrease geometrically. Thus, the cost of processing the first level is dominant. At level $d + 2$, the size of each individual set doubles, which increases the cost of applying Lemma 2.3 by a factor of 2^{d+1}, but the $(d + 2)$ previous merging steps in the previous round have reduced the total number of sets by 2^{d+2}, so the running time of the following round is at most half of the time for a previous round. Similarly, in the second phase, the running time follows a geometrically decreasing sequence at each step. Thus, the total running time of the algorithm is $O(d)^{3d} r^{2d} \log^d(dr)|X|$, and the proof of Theorem 2.1 is complete. □

2.2. Product range spaces. Let $\Sigma_1 = (X, \mathcal{R})$ and $\Sigma_2 = (Y, \mathcal{S})$ be (finite) range spaces. We define the product range space $\Sigma_1 \otimes \Sigma_2$ to be $(X \times Y, \mathcal{T})$, where $\mathcal{T}$ consists

of all subsets $T \subseteq X \times Y$ such that all the *cross-sections* $S_x = \{ y \in Y : (x, y) \in T\}$ are sets of $\mathcal{S}$, and similarly, all $R_y = \{ x \in X : (x, y) \in T\}$ are sets of $\mathcal{R}$.

To illustrate this definition, consider the bichromatic arrangement of n red and n blue lines in $\mathbb{R}^2$. Ranges of the blue (resp., red) space consist of blue (resp., red) lines that intersect a given line segment. The product $\Sigma_1 \otimes \Sigma_2$ of the blue space by the red space is a range space $(Z, \mathcal{T})$, where Z consists of all the bichromatic intersections. A range is a subset T of Z such that the intersections in T that are incident upon any given line appear consecutively (among those of Z). For example, the bichromatic intersections that fall inside any convex set constitute a range. This suggests that the product of finite VC-dimensional spaces might not be itself of finite VC-dimension. Indeed, this can best be seen by observing that in our example, any bichromatic pairing of the lines gives a collection of n bichromatic intersections and that *any* of its 2^n subsets is a valid range!

THEOREM 2.6. *Let $\Sigma_1 = (X, \mathcal{R})$ and $\Sigma_2 = (Y, \mathcal{S})$ be two range spaces. If A (resp., B) is a δ-approximation (resp., ε-approximation) for Σ_1 (resp., Σ_2) for $0 \leq \delta, \varepsilon, \leq 1$, then $A \times B$ is a $(\delta + \varepsilon)$-approximation of $\Sigma_1 \otimes \Sigma_2$.* It is worth observing that even though an ε-approximation of a product space is of size $O(\varepsilon^{-4} \log^2 \varepsilon^{-1})$, its representation as a set product has size of only $O(\varepsilon^{-2} \log \varepsilon^{-1})$.

A range space of infinite VC-dimension has, for infinitely many n, a shattered subset A of size n, and clearly an ε-approximation for the subspace induced by such A must be of size at least $(1 - \varepsilon)n$. This might seem to contradict Theorem 2.6. To explain this apparent paradox, we must observe that, in general, a subspace of a product range space is not itself a product range space. In particular, even though the ground set contains very large shattered subsets, the subsystems induced by these subsets are not product range spaces; therefore, the fact that they cannot be sampled has no bearing on Theorem 2.6. In fact, the proper definition of a subspace in the context of product range spaces would be the product of subspaces of standard range spaces.

Proof of Theorem 2.6. Given two range spaces of finite VC-dimension, $\Sigma_1 = (X, \mathcal{R})$ and $\Sigma_2 = (Y, \mathcal{S})$, recall that the product $\Sigma_1 \otimes \Sigma_2$ is defined as $(Z, \mathcal{T})$, where $Z = X \times Y$ and $\mathcal{T}$ consists of all the subsets $T \subseteq Z$ such that for any $x \in X$ and $y \in Y$ the sets T^x and T_y are ranges of $\mathcal{S}$ and $\mathcal{R}$, resp., where

$$T^x = \{ y : (x, y) \in \mathcal{T} \},$$

$$T_y = \{ x : (x, y) \in \mathcal{T} \}.$$

As we observed, $\Sigma_1 \otimes \Sigma_2$ usually does not have finite VC-dimension. For example, if Σ_1 and Σ_2 are the (infinite) range spaces defined by two secant lines and their intervals, the product space ranges include all the convex regions of the plane.

It is helpful to use a slightly different formulation of an ε-approximation. Let Prob_X be a probability distribution on X, and $\mathrm{Prob}_X[\, R \,|\, A \,]$ be the conditional probability that a random element in X belongs to R, given that it is in A. Thus, for A to be a δ-approximation for Σ_1, it is equivalent to say that, for every $R \in \mathcal{R}$,

$$|\, \mathrm{Prob}_X[\, R \,|\, A \,] - \mathrm{Prob}_X[R] \,| \leq \delta \,.$$

A simple technical observation will greatly simplify our discussion below. In essence, it is nothing more than Fubini's theorem and asserts that we can sum the probabilities

first on x then on y, or first on y then on x, and obtain the same result. To put it in mathematical notation, given any $A \subseteq X$, $B \subseteq Y$, and $T \in \mathcal{T}$, we have

$$\mathrm{Prob}_Z\,[\,T \,|\, A \times B\,] = \begin{cases} \mathbf{E}_X\,[\,\mathrm{Prob}_Y\,[\,T^x \,|\, B\,]\,|\,A\,], \\ \mathbf{E}_Y\,[\,\mathrm{Prob}_X\,[\,T_y \,|\, A\,]\,|\,B\,], \end{cases}$$

where $\mathbf{E}_X[\cdot|A]$ denotes the expectation for a random element of X given that the element belongs to A, and $\mathbf{E}_Y[\cdot|B]$ is the analogous conditional expectation on Y. To prove Theorem 2.6, we apply this observation twice. Recall that A (resp., B) is a δ-approximation (resp., ε-approximation) of Σ_1 (resp., Σ_2); then

$$\begin{aligned} \mathrm{Prob}_Z\,[\,T \,|\, A \times B\,] &= \mathbf{E}_X\,[\,\mathrm{Prob}_Y\,[\,T^x \,|\, B\,]\,|\,A\,] \\ &= \mathbf{E}_X\,[\,\mathrm{Prob}_Y\,[T^x]\,|\,A\,] + \varepsilon' \\ &= \mathbf{E}_Y\,[\,\mathrm{Prob}_X[\,T_y \,|\, A\,]\,] + \varepsilon' \\ &= \mathbf{E}_Y\,[\,\mathrm{Prob}_X\,[T_y]\,] + \delta' + \varepsilon' \\ &= \mathrm{Prob}_Z\,[T] + \delta' + \varepsilon', \end{aligned}$$

where $|\varepsilon'| \le \varepsilon$, $|\delta'| \le \delta$, which completes the proof of Theorem 2.6. □

A similar result exists for sensitive approximations, but the formulas are a little more complicated. It is easy to show, however, that the product of a sensitive δ-approximation with a sensitive ε-approximation is a sensitive $\sqrt{2}(\delta + \varepsilon)$-approximation [5].

Finally, we should note that the product described here is associative. We can thus take the d-fold product $\Sigma \otimes \cdots \otimes \Sigma$ of a range space $\Sigma = (X, \mathcal{R})$. Theorem 2.6 may be extended straightforwardly.

COROLLARY 2.7. *If A is an ε-approximation for a range space Σ, then the d-fold Cartesian product A^d is a $(d\varepsilon)$-approximation of the d-fold product $\Sigma \otimes \cdots \otimes \Sigma$.* For sensitive approximations, the theorem can be extended similarly. It is easy to show that the d-fold product of a sensitive ε-approximation is a sensitive $(d^2\varepsilon)$-approximation [5].

Let us show, for instance, how to use this range space product to estimate the number of vertices of an arrangement of hyperplanes inside a convex region. The range space $\Sigma = (X, \mathcal{R})$ under consideration here is the one described above: given a set H of hyperplanes in $\mathbb{R}^d$, $\mathcal{R}$ is the collection of all the subsets of H consisting of the hyperplanes stabbed by a given line segment. We let Σ^d be the d-fold product of Σ. Of particular interest is the subset $H^{(d)}$ of H^d consisting of the d-tuples of hyperplanes of H in general position: such d-tuples intersect in a unique point of $\mathbb{R}^d$ which is a vertex of the arrangement of H. Let us denote by $V(H)$ the set of these vertices. For a convex region σ (not necessarily full-dimensional), consider the arrangement of the intersections of hyperplanes of H with the affine hull of σ, and let $V(H, \sigma)$ be the set of vertices of this arrangement lying inside σ.

THEOREM 2.8. *Let H be a set of hyperplanes in general position, and let A be an ε-approximation for $\Sigma = (H, \mathcal{R})$. Then, for any convex region σ of dimension j in $\mathbb{R}^d$,*

$$\left| \frac{|V(H,\sigma)|}{|H|^j} - \frac{|V(A,\sigma)|}{|A|^j} \right| \le \varepsilon.$$

Proof. This theorem was already shown in [6] for the particular case of simplices. The proof in terms of range space products is particularly simple. We may assume

that σ is d-dimensional; otherwise the result may be proved by considering the j-fold product of Σ.

Let $(h_1, \ldots, h_d)$ be a d-tuple in $H^{(d)}$, and let $f(h_1, \ldots, h_d) = h_1 \cap \ldots \cap h_d$ be the corresponding vertex of $V(H)$. Note that because the hyperplanes are in general position, f is a $d!$ to one map.

Given any convex region σ, the inverse image $T = f^{-1}(V(H, \sigma))$ is a range in Σ^d. Indeed, it suffices to prove that its sections $T^i_{(h_1,\ldots,h_d)}$ consisting of the hyperplanes h such that $(h_1, \ldots, h_{i-1}, h, h_{i+1}, \ldots, h_d)$ is in T are exactly the hyperplanes stabbed by some segment s in $\mathbb{R}^d$. Note that $\cap_{j \neq i} h_j$ is a line in $\mathbb{R}^d$ and that it intersects σ along such a segment s. Moreover, a d-tuple $(h_1, \ldots, h_{i-1}, h, h_{i+1}, \ldots, h_d)$ is in T if and only if h is stabbed by the line segment s.

Finally, we note that the size of $T = f^{-1}(V(H, \sigma))$ is $d!$ times the number of vertices of $V(H, \sigma)$. Using Corollary 2.7, we conclude that

$$\left| \frac{|V(H,\sigma)|}{|H|^d} - \frac{|V(A,\sigma)|}{|A|^d} \right| \leq \frac{d\varepsilon}{d!} \leq \varepsilon. \qquad \square$$

A similar result can be proved for sensitive approximations, with slightly worse approximation bounds. For instance, it is proved in [5] that

$$\left| \frac{|V(H,\sigma)|}{|H|^j} - \frac{|V(A,\sigma)|}{|A|^j} \right| \leq \frac{4\varepsilon}{2} \left(\sqrt{\frac{|V(H,\sigma)|}{|H|^j}} + 4\varepsilon \right)$$

for a set H of hyperplanes in general position, an ε-approximation A for $\Sigma = (H, \mathcal{R})$, and any convex region σ of dimension j in $\mathbb{R}^d$.

3. Computing convex hulls. We describe a new deterministic algorithm for computing the convex hull of n points. Its running time of $O(n \log n + n^{\lfloor d/2 \rfloor})$ is optimal in any fixed dimension d. Our strategy is similar to the derandomization scheme used in [7]. In particular, it is still built around Raghavan's [23], Spencer's [24], and Alon and Spencer's [2] methods of conditional probabilities. The main difference is in the underlying probabilistic model and the maintenance of approximation tools. The result is an algorithm that is arguably simpler.

The convex hull problem is reducible, by duality, to computing the intersection of n halfspaces. This problem can be solved in optimal expected time by a randomized incremental algorithm [13]: the halfspaces are inserted in random order, and the current intersection is maintained after each insertion.

We aim at derandomizing such an algorithm. For technical reasons, we use a slightly different randomized algorithm as a basis, to be described below.

3.1. Notation and preliminaries. Let H be a fixed collection of n hyperplanes in $\mathbb{R}^d$, and let O (the *origin*) be a given point not lying on any hyperplane of H. Using simulation of simplicity if necessary [16], we may assume that the hyperplanes of H are in general position. We also may enclose $\mathbb{R}^d$ in a box that contains all the vertices of the arrangement of H. The set H_0 of hyperplanes bounding this box is added to H. In what follows, when we let R be a subset of H, we assume that R is a subset of H that contains H_0. This ensures that we always deal with polytopes and not polyhedra and thus clears the issue of unboundedness.

Let R be a subset of H. We let $R^{\cap}$ denote the closure of the cell enclosing O of the arrangement of R. In what follows, the word "simplex" always means a *relatively*

open simplex. Similarly, the word "face" always means a relatively open face of a polytope or of an arrangement.

Given a simplex s, let $R_{|s}$ denote the subset of hyperplanes of R that intersect s but do not contain it.

Given a polytope P (such as $R^{\cap}$), we let $V(P)$ denote the set of vertices of P. If R is a set of hyperplanes, we let $V(R)$ be the set of vertices of the arrangement of R. For a simplex s (not necessarily full-dimensional), consider the arrangement of the intersections of hyperplanes of R with the affine hull of s, and let $V(R,s)$ be the set of vertices of this arrangement lying inside s.

For a vertex $v \in V(H)$, we define the *conflict list* of v, denoted $H_{|Ov}$, to be the set of hyperplanes separating v from the origin. The *level of* v, denoted by n_v, is the size of $H_{|Ov}$. Similarly we define the conflict list for a simplex s, denoted by $H_{|Os}$, as the set of hyperplanes of H intersecting the relative interior of the convex hull of $s \cup \{O\}$. We note that the conflict list of a simplex is the union of the conflict lists of its vertices. We set $n_s = |H_{|Os}|$ (but note that n_s and $|H_{|s}|$ may be different for simplices s on the boundary of $R^{\cap}$).

In our algorithm, we use a special kind of triangulation for $R^{\cap}$ called the *geode of* R that is denoted by $\mathcal{G}(R)$. Formal definitions and important properties are given in [7]. The geode consists of a triangulation of the boundary of $R^{\cap}$ along with a central lifting of that triangulation towards the origin O. The triangulation is defined recursively and is similar to the so-called *bottom vertex triangulation* [11]. To triangulate a face f of dimension k of $\partial R^{\cap}$, we first triangulate its faces of dimension $\le k-1$ and then we lift these triangulations to the *apex* of f, where the apex is the vertex contained in this face with the smallest conflict list (ties being broken using some systematic rule, such as taking the vertex with lexicographically smallest coordinate vector). The resulting geode contains $O(n^{\lfloor d/2 \rfloor})$ simplices, and the choice of the apex leads to the following property.

LEMMA 3.1 (see [7]). *For any integer c, and any $R \subseteq H$, there is a constant $b = b(c)$ such that for any $c' \le c$, we have*

$$\sum_{s \in \mathcal{G}(R)} n_s^{c'} \le b \sum_{v \in V(R^{\cap})} n_v^{c'}. \tag{3.1}$$

Proof. The proof is the same as in [7], but the result there is stated for $H_{|s}$ instead of n_s (even though the proof itself is actually stated in terms of n_s). We recall it for completeness.

We let $\bar{f}$ denote the closure of a face f of $R^{\cap}$. We prove by induction that, for any k-face of $R^{\cap}$, the sum $\sum_{s \in \mathcal{G}(R) \cap \bar{f}} n_s^c$, denoted A_f, is at most

$$k!(2^c+1)^k \sum_{v \in V(R^{\cap}) \cap \bar{f}} n_v^c \ .$$

The lemma follows from the case $k = d$, where f is the interior of $R^{\cap}$.

The case $k = 0$ is immediate, as the 0-faces of $R^{\cap}$ are precisely its vertices. Assume the induction hypothesis is true for some $k - 1 < d$. We observe that, by choosing the apex of f as the lifting vertex for the geode (or the origin, if $k = d$), $A_f \le (2^c+1) \sum_g A_g$, where g ranges over all the $(k-1)$-faces of $R^{\cap}$ incident upon f. The term 1 comes from the contribution of the faces incident upon f, while the term 2^c accounts (conservatively) for the effect of lifting g toward the apex w (or $w = O$ if $k = d$; note that $n_w \le n_s$ for any s contained in the closure of g, by definition

of w). By our general position assumption, a vertex cannot belong to more than k $(k-1)$-faces, so we cannot count it more than k times in the above sum ranging on g. Substituting for A_g with the induction hypothesis gives the result. ∎

In our algorithm and analysis, various constants will appear (dependent on d, as a rule). To avoid complicated implicit dependencies between them, we express most constants as functions of two basic parameters C and c. For all estimates to work, one first chooses c as a sufficiently large constant and then C as a still much larger constant. The $O()$ notation in the proofs may hide constants dependent on c (and d), but *not on* C; where the hidden constant does depend on C, we use the $O_C()$ notation.

We need an estimate for the higher moments of the binomial distribution.

LEMMA 3.2. *Let $X = X_{[1,n]} = X_1 + X_2 + \cdots + X_n$, where the X_i are independent random variables, each attaining value* 1 *with probability p and value* 0 *with probability $1-p$. Then for any natural number c, $\mathbf{E}[X^c] \le (c+np)^c$.*

This must be part of the folklore, but since we haven't discovered an explicit reference, we include a short proof.

Proof. We prove more generally that $\mathbf{E}[(X+a)^c] \le (c+np+a)^c$ for all natural numbers n, c, and a. The inequality is clear if $n = 0$ or $c = 0, 1$. Assuming this is true for some c and for all n and a, we derive by induction that

$$\begin{aligned}\mathbf{E}[(X+a)^{c+1}] &= \sum_{i=1}^{n} p\mathbf{E}[(X+a)^c | X_i = 1] + a\mathbf{E}[(X+a)^c] \\ &= np\mathbf{E}[(X_{[1,n-1]} + a + 1)^c] + a\mathbf{E}[(X+a)^c] \\ &\le np(c + (n-1)p + a + 1)^c + a(c+np+a)^c \\ &\le (c+1+np+a)^{c+1}. \qquad \blacksquare\end{aligned}$$

3.2. The underlying randomized algorithm. In our underlying randomized algorithm, hyperplanes are inserted in rounds. In the first round, a suitable constant number c of hyperplanes are chosen (arbitrarily, not necessarily at random) and inserted. Suppose that after the $(j-1)$st round, a set $R \subseteq H$ has been inserted, $|R| = r$. We assume a suitable representation of $\mathcal{G}(R)$, the geode of R. We also keep the conflict list of every simplex $s \in \mathcal{G}(R)$.

In the jth round, we fix a probability

$$p = \frac{2}{3}\,\frac{r}{n-r}$$

and we choose a random sample S from $H \setminus R$ by picking each hyperplane of $H \setminus R$ into S randomly and independently with probability p. For each simplex $s \in \mathcal{G}(R)$, we compute the portion of the arrangement of S lying within s; then we isolate the portion of $(R \cup S)^{\cap}$ within s from the portion of the arrangement, and we glue these pieces together, obtaining the facial lattice of $(R \cup S)^{\cap}$. Using the conflict lists of the vertices, we finally compute the geode $\mathcal{G}(R \cup S)$ and the conflict lists of its simplices.

The expected number of hyperplanes in S is $\frac{2}{3}r$, thus the size of R increases geometrically between rounds and the expected number of rounds is $O(\log n)$. When the number of hyperplanes in R exceeds n/c, we insert all the remaining hyperplanes of $H \setminus R$ (in a manner similar to adding a new sample S) and finish.

The work in the jth round of this algorithm is at most proportional to

$$\sum_{s \in \mathcal{G}(R)} \left(|S_{|Os}|^d + n_s \right) \tag{3.2}$$

(see also [7] for a more detailed description of the required computations and of their time complexity). Intuitively, we should expect the sizes n_s of the conflict lists to be about n/r and each $S_{|Os}$ to have about constant size. This is not quite true of *all* simplices in the randomized algorithm (and even less so in the derandomized version). However, as was observed by Clarkson in a somewhat different context [12], the averages of $|S_{|Os}|^c$ and of $\left(n_s \frac{r}{n}\right)^c$ over all simplices of $\mathcal{G}(R)$ are expected to be bounded by a constant in the randomized algorithm for any constant c, and this is what we will also aim at in the derandomized version. From this point of view, we might appropriately call the quantities $|S_{|Os}|$ and $n_s \frac{r}{n}$ *quasi constant*. To simplify the notation, we introduce the symbols

$$q_s = n_s \frac{r}{n} + 1\,,$$
$$r_s = |S_{|Os}| + 1\,.$$

The basic property of a quasi-constant quantity x_s is that its moment of order c is $\sum_{s\in\mathcal{G}(R)} x_s^c = O_C(1) r^{\lfloor d/2\rfloor}$ for a constant C that will be determined later. Note that this also implies the same property for the moments of order $c' \le c$ by a routine application of Hölder's inequality.

The rest of this section is devoted to proving that the above quantities q_s and r_s are expected to be quasi constant.

Following [7], we say that the geode of R is a *semicutting* if

$$\sum_{v\in V(R^\cap)} n_v^c \le N \stackrel{\text{def}}{=} C r^{\lfloor d/2\rfloor} \left(\frac{n}{r}\right)^c .$$

Again, a routine application of Hölder's inequality shows that

$$\sum_{v\in V(R^\cap)} n_v^{c'} \le C r^{\lfloor d/2\rfloor} \left(\frac{n}{r}\right)^{c'}$$

for any $0 \le c' \le c$. Note that the definition of semicutting involves only the conflict lists of the vertices and not of the simplices of the geode. This has no bearing on the analysis, however; recall from Lemma 3.1 that the particular triangulation of $R^\cap$ we use, the geode, is chosen in such a way that we also have

$$\sum_{s\in\mathcal{G}(R)} n_s^{c'} \le b \sum_{v\in V(R^\cap)} n_v^{c'}$$

for any $0 \le c' \le c$ and for some constant $b = b(c')$. Note that this implies that $\sum_{s\in\mathcal{G}(R)} q_s^{c'} = O_C(1) r^{\lfloor d/2\rfloor}$ and hence that q_s is quasi constant.

Inductively, we assume that the geode of R built in the previous rounds of the algorithm is a semicutting. For the sample S, we postulate the following conditions:

C1. $r/2 \le |S| \le r$.

C2. $\sum_{s\in\mathcal{G}(R)} r_s^c \le C^2 r^{\lfloor d/2\rfloor}$.

C3. The geode of $R \cup S$ is a semicutting.

As we will see shortly, the randomized algorithm yields these properties with high probability.[1]

[1] The reader might wonder why we look at high moments when the complexity of the randomized algorithm only involves n_s and the dth power of r_s. The reason is that in the derandomization, we need auxiliary computations whose complexity is a larger polynomial in q_s and r_s.

Corresponding to these properties, we introduce three functions measuring the quality of the sample S. We put

$$F_1(S) = \frac{1}{4r}\left(|S| - \frac{2}{3}r\right)^2,$$
$$F_2(S) = \frac{1}{C^2 r^{\lfloor d/2 \rfloor}} \sum_{s \in \mathcal{G}(R)} r_s^c,$$
$$F_3(S) = \frac{1}{N} \sum_{v \in V((R \cup S)^{\cap})} n_v^c .$$

Further we define the quantities

$$\mathcal{E}_j = \mathbf{E} F_j(S)$$

for $j = 1, 2, 3$, where the expectation is taken with respect to a random choice of S (it implicitly depends on R, which we consider fixed). We put $\mathcal{E} = \mathcal{E}_1 + \mathcal{E}_2 + \mathcal{E}_3$. The quantities $\mathcal{E}$ and $\mathcal{E}_j$ will be referred to as *energy* (for reasons more apparent later). We now bound $\mathcal{E}$.

LEMMA 3.3. *We have* $\mathcal{E}_1 \le 1/6$.

Proof. It is immediate that $\mathcal{E}_1 = \frac{\mathbf{var}\,|S|}{4r} = \left(\frac{n-r}{4r}\right) p(1-p) \le \frac{1}{6}$. □

LEMMA 3.4. *If the geode of R is a semicutting, then $\mathcal{E}_2 = O(1/C)$.*

Proof. Consider a simplex $s \in \mathcal{G}(R)$. The contribution of every hyperplane $h \in H_{|Os}$ to $|S_{|Os}|$ is a 0/1 random variable attaining value 1 with probability p, and so by Lemma 3.2 we have $\mathbf{E}\,|S_{|Os}|^c \le (c + p|H_{|Os}|)^c \le (c + pn_s)^c = O(q_s^c)$. Summing over all the simplices $s \in \mathcal{G}(R)$ shows that the expectation of $\sum_{s \in \mathcal{G}(R)} r_s^c = O(\sum_{s \in \mathcal{G}(R)} q_s^c)$. Using (3.1), the latter expression is $O(Cr^{\lfloor d/2 \rfloor})$, which proves the lemma. □

The next lemma concerns $\mathcal{E}_3$. Unlike the previous lemma, it does not assume that the geode of R is a semicutting; thus, no matter how "bad" R might be, a random S guarantees that the geode of $R \cup S$ is a semicutting (with high probability). This robustness property will be crucial: in the derandomized version, the computed R presumably won't be as good as a true random sample would be, but the error will not propagate between rounds, as each new round alone would suffice to produce a semicutting for *any* R provided that S is random or imitates a random sample well enough.

LEMMA 3.5. *Let $R \subset H$ be arbitrary, $c \le |R| = r \le n/c$, and let S be a random sample from $H \setminus R$ obtained by choosing each hyperplane independently with probability p. Then*

$$\mathcal{E}_3 = \frac{1}{N} \sum_{v \in V(H) \cap R^{\cap}} p^{d_v} (1-p)^{n_v} n_v^c = O(1/C)\,, \tag{3.3}$$

where d_v denotes the number of hyperplanes of $H \setminus R$ passing thru v.

Proof. For a vertex $v \in V(H) \cap R^{\cap}$, the probability of appearing as a vertex in $V((R \cup S)^{\cap})$ is equal to $p^{d_v}(1-p)^{n_v}$, and thus the middle sum in (3.3) is the expectation of

$$\sum_{v \in V((R \cup S)^{\cap})} n_v^c\,,$$

which equals $N\mathcal{E}_3$. Recall that $N = Cr^{\lfloor d/2 \rfloor}(n/r)^c$.

To prove the upper bound of $O(1/C)$, we consider another sample $\bar{S}$ drawn from $H \setminus R$ with probability $\bar{p} = p/2$. Let Q denote the quantity

$$\sum_{v \in V(H) \cap R^\cap} \bar{p}^{d_v} (1 - \bar{p})^{n_v} .$$

This sum is nothing more than the expected number of vertices of $(R \cup \bar{S})^\cap$, which is at most $\mathbf{E}\,|R \cup \bar{S}|^{\lfloor d/2 \rfloor} = O(r^{\lfloor d/2 \rfloor})$, using the upper bound theorem and Lemma 3.2. We estimate

$$\frac{1 - \bar{p}}{1 - p} \geq 1 + \frac{p}{2} \geq e^{p/4} \geq e^{r/8n} .$$

Then we rewrite

$$Q = \sum_v p^{d_v} 2^{-d_v} (1 - p)^{n_v} \left(\frac{1 - \bar{p}}{1 - p} \right)^{n_v} \geq 2^{-d} \sum_v p^{d_v} (1 - p)^{n_v} e^{n_v r/8n} . \tag{3.4}$$

We have

$$n_v^c = \left(\frac{8cn}{r} \right)^c \left(\frac{n_v r}{8cn} \right)^c \leq \left(\frac{8cn}{r} \right)^c e^{n_v r/8n} .$$

Substituting this estimate into the middle sum in (3.3) and comparing with the lower bound for Q in (3.4), we obtain

$$\sum_v p^{d_v} (1 - p)^{n_v} n_v^c \leq 2^d \left(\frac{8cn}{r} \right)^c Q = O((n/r)^c r^{\lfloor d/2 \rfloor}) = O(N/C). \qquad \square$$

Putting these three lemmas together shows that $\mathcal{E} < 1/2$ for a suitable choice of C. Using Markov's inequality, this shows that $F_1(S) + F_2(S) + F_3(S) < 1$ or that conditions C1–C3 are satisfied, with probability of at least $1/2$.

We can now finish the running time analysis of the randomized algorithm under the condition that at each round, the sample S picked by the algorithm satisfies conditions C1–C3. Indeed, the work in the jth round is given in (3.2). Due to condition C2, the first term of the inner sum adds up to $O_C(r^{\lfloor d/2 \rfloor})$ over all the simplices of the geode, while condition C3 implies that the second term adds up to $O_C(r^{\lfloor d/2 \rfloor}) \frac{n}{r} = O_C(n r^{\lfloor d/2 \rfloor - 1})$. Suppose that after the $(j-1)$st round, r hyperplanes have been inserted. The work in the jth round is then $O_C(r^{\lfloor d/2 \rfloor} + n r^{\lfloor d/2 \rfloor - 1})$. Finally, condition C1 implies that the size of the sample grows geometrically between rounds. Thus, if conditions C1–C3 are satisfied at every round, the total running time sums up as $O_C(n \log n + n^{\lfloor d/2 \rfloor})$.

In the derandomized algorithm, conditions C1–C3 will be fulfilled at all rounds. The total running time of the deterministic algorithm will thus be $O_C(n \log n + n^{\lfloor d/2 \rfloor})$ plus whatever time is needed to perform the deterministic computation of a suitable sample at each round.

The randomized algorithm does not check if the conditions are satisfied, however. Thus a bad sample at a given round could severely slow down the algorithm, however unprobable this may be. Nevertheless, the expected time of the algorithm is, by linearity of expectations, the sum of the expected times of all the rounds, and the expected time of a round is bounded by $O_C(r^{\lfloor d/2 \rfloor} + n r^{\lfloor d/2 \rfloor - 1})$, as shown by Lemmas 3.3–3.5. Therefore, the expected time of the randomized algorithm is $O_C(n \log n + n^{\lfloor d/2 \rfloor})$.

3.3. Derandomization—a first attempt. Let us first consider a straightforward derandomization of the above described algorithm by the Raghavan–Spencer method, recalling the basic strategy of that method and introducing some more notation. We are at the beginning of a round, with the geode of R as a semicutting, and we want to find $S \subseteq H \setminus R$ such that $F_1(S) + F_2(S) + F_3(S) \le 1$ (thus satisfying conditions C1–C3). We order the hyperplanes of $H \setminus R$ into a sequence $h_1, h_2, \dots, h_{n-r}$ (arbitrarily), and we process them one by one, deciding for each h_i whether to *accept* it (that is, to include it in S) or to *reject* it (not to include it in S).

Having processed $h_1, \dots, h_k$, let $S^{(k)}$ denote the set of accepted hyperplanes among them. We define the energies $\mathcal{E}_j^{(k)}$ as the conditional expectations

$$\mathcal{E}_j^{(k)} = \mathbf{E}\left(F_j(S) \,|\, S \cap \{h_1, \dots, h_k\} = S^{(k)}\right),$$

and $\mathcal{E}^{(k)}$ is again their sum. Further we let

$$\mathcal{E}_j^{(k|A)} = \mathbf{E}\left(F_j(S) \,|\, S \cap \{h_1, \dots, h_k\} = S^{(k)},\, h_{k+1} \in S\right)$$

measure what the energy would be after accepting h_{k+1}, and similarly we let

$$\mathcal{E}_j^{(k|R)} = \mathbf{E}\left(F_j(S) \,|\, S \cap \{h_1, \dots, h_k\} = S^{(k)},\, h_{k+1} \notin S\right)$$

measure what the energy would be after rejecting h_{k+1}.

The strategy dictated by the Raghavan–Spencer method is as follows: the hyperplane h_{k+1} is accepted or rejected, whichever decision gives a lower total energy $\mathcal{E}^{(k+1)}$. The key property of the energy is

$$\mathcal{E}^{(k)} = p\,\mathcal{E}^{(k|A)} + (1-p)\,\mathcal{E}^{(k|R)}. \tag{3.5}$$

This, together with the decision rule, implies that $\mathcal{E}^{(k+1)} \le \mathcal{E}^{(k)}$ for $k = 0, 1, \dots, n-r-1$; therefore, the final energy is at most 1, and since it equals $\sum_j F_j(S^{(n-r)})$ for the (already fixed) sample $S^{(n-r)}$, this sample will satisfy the required conditions.

The evaluation of $\mathcal{E}_1^{(k)}$ and $\mathcal{E}_2^{(k)}$ is easy (at least with a limited but sufficient accuracy) and requires no more time than the other operations of the randomized algorithm itself. On the other hand, $\mathcal{E}_3^{(k)}$ appears much more demanding, and we cannot evaluate it exactly, so we use a suitable approximation instead, denoted by $\mathcal{AE}_3^{(k)}$. The sum $\mathcal{AE}^{(k)} = \mathcal{E}_1^{(k)} + \mathcal{E}_2^{(k)} + \mathcal{AE}_3^{(k)}$ will be called the *approximate energy*.

Here is a rough outline of our strategy. We shall be careful to define $\mathcal{AE}_3^{(k)}$ in such a way that it obeys an equation analogous to (3.5). Then we apply the Raghavan–Spencer method with the approximate energy instead of the actual energy, producing a sample $S^{(n-r)}$ for which the approximate energy does not exceed the initial approximate energy $\mathcal{AE}^{(0)}$. To make everything work, we show the following lemma.

LEMMA 3.6. *For every* $k = 0, 1, \dots, n-r$, $|\mathcal{E}_3^{(k)} - \mathcal{AE}_3^{(k)}| < 1/3$.

The lemma is proved in the next section, where we explain how to compute the approximate energy. Using this for $k = 0$, together with $\mathcal{E}^{(0)} < 1/3$ (which follows from the results of section 3.2), we see that the initial approximate energy is smaller than 2/3 and hence so is the final approximate energy. This in turn implies that the final energy $\mathcal{E}^{(n-r)}$ is less than 1 and hence that the sample $S^{(n-r)}$ satisfies conditions C1–C3.

3.4. Approximating the energy. In this section we define the approximate energy $\mathcal{AE}_3^{(k)}$ and establish Lemma 3.6, assuming the existence of a certain oracle. The implementation of the oracle is discussed in the next section.

We begin by setting the initial value $\mathcal{AE}_3^{(0)}$. Consider the expression for $\mathcal{E}_3 = \mathcal{E}_3^{(0)}$ in (3.3). We split the sum according to the simplices of $\mathcal{G}(R)$ containing the respective vertices, and we get

$$\mathcal{E}_3^{(0)} = \frac{1}{N} \sum_{s\in\mathcal{G}(R)} \sum_{v\in V(H)\cap s} p^{d_v}(1-p)^{n_v} n_v^c \,.$$

By a suitable general position assumption, we may suppose that a j-dimensional simplex $s \in \mathcal{G}(R)$ contains no vertices of $V(H)$ unless it is a part of a j-face of the polytope $R^{\cap}$; thus a vertex of $V(H)$ in such a j-simplex is contained in $d-j$ hyperplanes of R and j hyperplanes of $H\backslash R$, or in other words, $d_v = j = \dim s$. In this sense, all the vertices within s are of the same type, and we have $V(H)\cap s = V(H,s)$. This implies that

$$\mathcal{E}_3^{(0)} = \frac{1}{N} \sum_{s\in\mathcal{G}(R)} \sum_{v\in V(H,s)} p^{d_v}(1-p)^{n_v} n_v^c \,. \tag{3.6}$$

We describe an oracle, to be constructed later, that performs an approximate evaluation of the sums over a given simplex s. Let $\mathcal{O}$ be an oracle whose input is a j-simplex $s \in \mathcal{G}(R)$ and whose output is a number $\mathcal{O}(s)$, satisfying

$$\Big|\mathcal{O}(s) - \sum_{v\in V(H,s)} p^j(1-p)^{n_v} n_v^c\Big| \le E_s \stackrel{\text{def}}{=} \frac{1}{C^2 q_s^{\sqrt{c}}} n_s^c \,. \tag{3.7}$$

(Here is an attempt to give the reader some intuition about the choice of the error term E_s: the simplex s contains at most n_s^j vertices of $V(H)$, and for each vertex the summand is at most $p^j n_s^c$; thus the exact sum does not exceed $n_s^c q_s^d$. The approximation's relative accuracy is thus a suitable quasi-constant factor.)

We then define the initial approximate energy by

$$\mathcal{AE}_3^{(0)} \stackrel{\text{def}}{=} \frac{1}{N} \sum_{s\in\mathcal{G}(R)} \mathcal{O}(s) \,.$$

Assuming (3.6) and (3.7), we have

$$\Big|\mathcal{AE}_3^{(0)} - \mathcal{E}_3^{(0)}\Big| \le \frac{1}{N} \sum_{s\in\mathcal{G}(R)} E_s \le \frac{1}{C^3 r^{\lfloor d/2\rfloor}} \sum_{s\in\mathcal{G}(R)} q_s^{c-\sqrt{c}} = O(1/C^2) \tag{3.8}$$

by the semicutting property of R, which establishes Lemma 3.6 for $k=0$, provided C is big enough.

We proceed to the definition of $\mathcal{AE}_3^{(k)}$. For a vertex $v \in V(H) \cap (R \cup S^{(k)})^{\cap}$, let m_v denote the number of hyperplanes *among* $\{h_{k+1}, \ldots, h_{n-r}\}$ in its conflict list[2] (that is, not counting the rejected hyperplanes). We also let d_v be the number of

[2]To be formally consistent, we should also superscript m_v by (k), but this would overburden the notation.

hyperplanes among $\{h_{k+1}, \ldots, h_{n-r}\}$ passing through v. In a manner analogous to the above expression for $\mathcal{E}_3$, we can write

$$\mathcal{E}_3^{(k)} = \frac{1}{N} \sum_v p^{d_v} (1-p)^{m_v} n_v^c \,, \tag{3.9}$$

where the summation is taken over all vertices v of the arrangement of $R \cup S^{(k)} \cup \{h_{k+1}, \ldots, h_{n-r}\}$ lying in the (closed) polytope $(R \cup S^{(k)})^{\cap}$.

We describe an oracle $\mathcal{O}^{(k)}$, which can approximately evaluate a part of this sum over a suitable cell (the oracle $\mathcal{O}$ above can be seen as a weaker version of $\mathcal{O}^{(0)}$). The input of $\mathcal{O}^{(k)}$ is a j-dimensional cell σ. We assume that the affine span of σ is either $\mathbb{R}^d$ or contained in the intersection of hyperplanes of H (under a suitable general position assumption, other cells do not contain any relevant vertices) and that σ is completely contained in a single simplex $s \in \mathcal{G}(R)$ (this latter requirement is not so important for the current section, but it is needed in the construction of the oracle). The oracle returns a number $\mathcal{O}^{(k)}(\sigma)$ with

$$\left| \mathcal{O}^{(k)}(\sigma) - \sum_{v \in V(\{h_{k+1}, \ldots, h_{n-r}\}, \sigma)} p^j (1-p)^{m_v} n_v^c \right| \le E_s = \frac{1}{C^2 q_s^{\sqrt{c}}} n_s^c, \tag{3.10}$$

where E_s is defined as in (3.7).

It might now seem natural to evaluate the approximate energy $\mathcal{AE}_3^{(k)}$ as follows: keep the portions of the arrangement of $S^{(k)}$ within each simplex $s \in \mathcal{G}(R)$, and call the oracle $\mathcal{O}^{(k)}$ on each cell from the resulting arrangements. It turns out that the error introduced in this way would be too large. Instead we compute the approximate energy incrementally, using the oracle to approximate the *difference* in energy caused by adding or rejecting a hyperplane.

Let us look at what happens with the contribution of various vertices to the total energy $\mathcal{E}_3$ when a hyperplane h_{k+1} is accepted or rejected; we begin with the accepting case. The contribution of vertices strictly above h_{k+1} remains unchanged (we say "above" meaning "on the same side of h_{k+1} as the origin"). The contribution of all vertices strictly below h_{k+1} becomes zero and finally, for each vertex on h_{k+1}, d_v, decreases by one so that its contribution to the energy is multiplied by $1/p$. Denoting the contribution of the vertices on h_{k+1} to the sum (3.9) by $\mathcal{E}_{on}^{(k)}$ and the contribution of the vertices below by $\mathcal{E}_{below}^{(k)}$, we have

$$\mathcal{E}_3^{(k|A)} = \mathcal{E}_3^{(k)} - \mathcal{E}_{below}^{(k)} + \left(\frac{1}{p} - 1\right) \mathcal{E}_{on}^{(k)} \,.$$

Thus, an appropriate action after accepting h_{k+1} is the following: we let Σ_{on} be the set of all faces σ of the polytope $(R \cup S^{(k)})^{\cap} \cap h_{k+1}$ within s for all $s \in \mathcal{G}(R)$. We set

$$\mathcal{AE}_{on}^{(k)} \stackrel{\text{def}}{=} \frac{p}{N} \sum_{\sigma \in \Sigma_{on}} \mathcal{O}^{(k)}(\sigma) \,.$$

(Note that the oracle includes the $p^{\dim \sigma}$ multiplicative factor, while an appropriate factor for a vertex on h_{k+1} in $\mathcal{E}_{on}^{(k)}$ is $p^{\dim \sigma + 1}$; this is why the factor p appears in the definition.)

Then we gather the portion of $(R \cup S^{(k)})^{\cap}$ (strictly) below h_{k+1} inside each s, obtaining a set Σ_{below} of cells, and we set

$$\mathcal{AE}_{below}^{(k)} \stackrel{\text{def}}{=} \frac{1}{N} \sum_{\sigma \in \Sigma_{below}} \mathcal{O}^{(k)}(\sigma)\,.$$

We define

$$\mathcal{AE}_3^{(k+1)} \stackrel{\text{def}}{=} \mathcal{AE}_3^{(k|A)} = \mathcal{AE}_3^{(k)} - \mathcal{AE}_{below}^{(k)} + \left(\frac{1}{p} - 1\right) \mathcal{AE}_{on}^{(k)}\,.$$

The discussion of the case when h_{k+1} is rejected is similar. The contribution of all vertices lying on h_{k+1} to the energy vanishes, and the number m_v for all vertices below h_{k+1} decreases by one; thus their contribution to the energy is multiplied by $1/(1-p)$. Hence an appropriate incremental definition is

$$\mathcal{AE}_3^{(k+1)} \stackrel{\text{def}}{=} \mathcal{AE}_3^{(k|R)} = \mathcal{AE}_3^{(k)} + \left(\frac{1}{1-p} - 1\right) \mathcal{AE}_{below}^{(k)} - \mathcal{AE}_{on}^{(k)}\,.$$

From these definitions, the promised analogy to (3.5), namely

$$\mathcal{AE}_3^{(k)} = p\,\mathcal{AE}_3^{(k|A)} + (1-p)\,\mathcal{AE}_3^{(k|R)}\,, \tag{3.11}$$

follows immediately.

As usual, we accept h_{k+1} if $\mathcal{AE}^{(k|A)} < \mathcal{AE}^{(k|R)}$ (otherwise, we reject it). Let us remark that since we have already established $\mathcal{AE}_3^{(0)} \le 2/3$, we know that the final approximate energy $\mathcal{AE}^{(n-r)} < 2/3$ and, in particular, that conditions C1 and C2 hold for the final sample $S^{(n-r)}$. Thus, any intermediate sample $S^{(k)}$ satisfies $|S^{(k)}| \le r$ as well as condition C2. We are thus free to use these conditions further. From now on, the quantity r_s will be defined with respect to the final sample $S^{(n-r)}$ computed by the algorithm; that is, $r_s = |S^{(n-r)}_{|Os}| + 1$.

Proof of Lemma 3.6. Let us analyze the approximation error. We have

$$\begin{aligned}\left|\mathcal{E}_3^{(k)} - \mathcal{AE}_3^{(k)}\right| \le{}& \left|\mathcal{E}_3^{(0)} - \mathcal{AE}_3^{(0)}\right| \\ &+ \sum_{i;\ h_i \in S^{(k)}} \left(\left|\mathcal{E}_{below}^{(i-1)} - \mathcal{AE}_{below}^{(i-1)}\right| + \frac{1-p}{p}\left|\mathcal{E}_{on}^{(i-1)} - \mathcal{AE}_{on}^{(i-1)}\right|\right) \\ &+ \sum_{i \in \{1,\ldots,k\};\ h_i \notin S^{(k)}} \left(\frac{p}{1-p}\left|\mathcal{E}_{below}^{(i-1)} - \mathcal{AE}_{below}^{(i-1)}\right| + \left|\mathcal{E}_{on}^{(i-1)} - \mathcal{AE}_{on}^{(i-1)}\right|\right)\,.\end{aligned}$$

From (3.8), we know that the first term is $O(1/C)$. Let us consider the contribution of a single simplex $s \in \mathcal{G}(R)$ to the second and third terms; we consider the accepting and rejecting cases separately. The sets Σ_{on} and Σ_{below} have no more than $O(r_s^d)$ cells, and there are fewer than r_s accepted hyperplanes cutting s or separating it from O. For an accepted hyperplane h_i, the error of the oracle given by (3.10) is multiplied by $(1-p)/N \le 1/N$ for the cells in Σ_{on}, and by $1/N$ for the cells in Σ_{below}. The total error for the accepted hyperplanes is thus $O(r_s^{d+1}E_s/N)$.

For a rejected hyperplane h_i, the error for cells in Σ_{on} gets multiplied by p/N and by $p/(1-p)N \le 2p/N$ for cells in Σ_{below}. Thus, the contribution to the error is at most $O(n_s r_s^d p E_s/N) = O(q_s r_s^d E_s/N)$. Substituting for E_s and N, we get that the total contribution to the error for s does not exceed $O(q_s^{c-\sqrt{c}+1} r_s^{d+1}/C^3 r^{\lfloor d/2 \rfloor})$.

We have bounds for the sums of cth moments of the r_s and of the q_s. In order to deal with the product of their powers, we use the inequality $xy \le x^u + y^v$, where the exponents satisfy $1/u + 1/v = 1$. In our case we have $x = r_s^{d+1}$, $y = q_s^{c-\sqrt{c}+1} \le q_s^{c-\sqrt{c}/2}$, $u = 2\sqrt{c}$, $v = 1/(1-1/u) = c/(c-\sqrt{c}/2)$. Then $r_s^{d+1} q_s^{c-\sqrt{c}+1} \le r_s^{2(d+1)\sqrt{c}} + q_s^c$. The total error over all simplices thus becomes

$$\frac{O(1)}{C^3 r^{\lfloor d/2 \rfloor}} \left(\sum_{s \in \mathcal{G}(R)} r_s^{2(d+1)\sqrt{c}} + \sum_{s \in \mathcal{G}(R)} q_s^c \right).$$

The first sum is less than $C^2 r^{\lfloor d/2 \rfloor}$ by condition C2, and the second sum is $O(Cr^{\lfloor d/2 \rfloor})$ by the semicutting property of the geode of R; hence the whole expression is $O(1/C)$. This proves Lemma 3.6. ∎

It now remains for us to implement the oracle.

3.5. Implementing the oracle.

LEMMA 3.7. *It is possible to maintain a data structure for each simplex $s \in \mathcal{G}(R)$ such that a call to the oracle $\mathcal{O}^{(k)}$ with a cell σ, as described in the previous sections, can be answered in $O_C(\mathrm{compl}(\sigma) q_s^{b\sqrt{c}})$ time for an absolute constant b, where $\mathrm{compl}(\sigma)$ denotes the combinatorial complexity of σ. The total time needed for updating the data structure for s during the round is bounded by $O_C(n_s r_s^d q_s^{b\sqrt{c}})$.*

Proof. The proof follows a similar construction in [7]. We define another quasi-constant quantity

$$\rho_s \stackrel{\text{def}}{=} C^{3d+6} q_s^{3d\sqrt{c}}.$$

Let $H_s^{(k)}$ denote the set of yet unprocessed hyperplanes in the conflict list of s; that is, $H_s^{(k)} = \{h_{k+1}, \ldots, h_{n-r}\}_{|Os}$. Whenever we want to call the oracle $\mathcal{O}^{(k)}$, we make sure we have an ε-approximation $A_s^{(k)}$ for the set $H_s^{(k)}$ (with ranges defined by segments) available, where ε is such that the absolute error of the approximation does not exceed n_s/ρ_s, and $|A_s^{(k)}| = O(\rho_s^2 \log \rho_s)$. A simple way to maintain such an ε-approximation under the deletion of hyperplanes is to start with, say, a $(1/2\rho_s)$-approximation $A_s^{(0)}$, keep it unchanged for a while, and recompute a fresh $(1/2\rho_s)$-approximation after every $n_s/2\rho_s$ deletion. By the results of [20] (or by Theorem 2.1), each $(1/2\rho_s)$-approximation is computed in time conservatively estimated as $O(\rho_s^{2d+1} n_s)$, so the total time for the maintenance of the ε-approximations is $O_C(n_s q_s^{b\sqrt{c}})$ as claimed.

Suppose that we want to answer a call to the oracle $\mathcal{O}^{(k)}$ with a j-cell σ, that is, approximate the sum

$$\sum_{v \in V(H_s^{(k)}, \sigma)} p^j (1-p)^{m_v} n_v^c.$$

Let us set

$$\alpha_s \stackrel{\text{def}}{=} \frac{|H_s^{(k)}|}{|A_s^{(k)}|}.$$

We are ready to define the oracle value

$$\mathcal{O}^{(k)}(\sigma) \stackrel{\text{def}}{=} \sum_{v \in V(A_s^{(k)}, \sigma)} \alpha_s^j p^j (1-p)^{m_v} n_v^c. \tag{3.12}$$

Note that the quantities m_v and n_v can be computed exactly for all the vertices $v \in V(A_s^{(k)}, \sigma)$, in time conservatively estimated as $O_C(\rho_s^{3d} n_s)$ (we remark that we use them as well in (3.7), (3.10) for a vertex $v \in V(\{h_{k+1}, \ldots, h_{n-r}\}, \sigma)$, but these quantities are never actually computed). Up to easy details, we have thus completely described the algorithm implementing the oracle, and the times it takes to answer a call or maintain the data structure can easily be shown to stay within the claimed bounds. It remains to establish the bound on the accuracy.

In order to have a notationally simpler proof, we only deal with the case $k = 0$ (then $n_v = m_v$, and we can omit all the (k) superscripts). The case of a general k is entirely similar and is treated in detail in [5].

Our procedure replaces a summation over a discrete but large set of vertices by summation over a smaller suitably chosen set, and this very much resembles a numerical integration procedure. The notation is chosen to stress this analogy, and also the reasoning in the proof resembles simple estimates for the error of numerical integration. From a result of [7], we know that the number of vertices of the ε-approximation within any simplex multiplied by a suitable scaling factor approximates the number of vertices of H within the simplex with relative accuracy ε. Theorem 2.8 shows that this is even valid for cells of arrangements (since they are convex). Our strategy is to subdivide σ into small enough cells so that the variation of the summand within each cell is small but at the same time the number of small cells is not too large. Within each small cell, we treat the summand as essentially constant and use the vertex number approximation bound.

Here is a more detailed treatment. Our specific function f to be integrated (over the discrete set of vertices) is

$$f(t) = p^j (1-p)^t t^c \,.$$

We let $M_{f,T}$ stand for its *modulus of continuity* $M_{f,T}$ over a set T (as is done in a somewhat different context in [22]):

$$M_{f,T}(h) = \sup_{\substack{t_1, t_2 \in T \\ |t_2 - t_1| \le h}} |f(t_2) - f(t_1)| \,.$$

For a j-dimensional cell $\xi \subseteq \sigma$, let us denote

$$\Sigma_\xi f = \sum_{v \in V(H,\xi)} f(n_v) \tag{3.13}$$

and

$$\tilde{\Sigma}_\xi f = \sum_{v \in V(A_s,\xi)} \alpha_s^j f(n_v) \,. \tag{3.14}$$

We want to obtain a bound for the difference $|\Sigma_\xi f - \tilde{\Sigma}_\xi f|$. Let $T_\xi = \{n_v : v \in V(H,\xi)\}$, $f_\xi^{min} = \min_{t \in T_\xi} f(t)$, $f_\xi^{max} = \max_{t \in T_\xi} f(t)$, and $\Delta_\xi(f) = f_\xi^{max} - f_\xi^{min}$. We have

$$|V(H,\xi)| f_\xi^{min} \le \Sigma_\xi f \le |V(H,\xi)| f_\xi^{max} \,,$$

and similarly

$$\alpha_s^j |V(A_s,\xi)| f_\xi^{min} \le \tilde{\Sigma}_\xi f \le \alpha_s^j |V(A_s,\xi)| f_\xi^{max} \,.$$

From this we get

$$\left|\Sigma_\xi f - \tilde{\Sigma}_\xi f\right| \le \left|\alpha_s^j |V(A_s,\xi)| - |V(H,\xi)|\right| f_\xi^{max} + |V(H,\xi)|\Delta_\xi(f)\,. \tag{3.15}$$

Theorem 2.8 shows that, for any j-dimensional cell ξ within s, we have

$$\left|\alpha_s^j |V(A_s,\xi)| - |V(H,\xi)|\right| \le \frac{n_s^j}{\rho_s}\,. \tag{3.16}$$

To estimate the total error within σ, we subdivide σ into smaller cells. Namely, we fix a parameter

$$\nu_s \stackrel{\text{def}}{=} C^3 q_s^{2\sqrt{c}}\,,$$

and we choose a $(1/\nu_s)$-net $\mathcal{N} \subseteq H_{|Os}$ (with respect to ranges defined by segments) of size $O(\nu_s \log \nu_s)$. We let Ξ be the portion of the arrangement of $\mathcal{N}$ within σ (by a suitable general position assumption, we may consider only j-dimensional cells in Ξ). Ξ is thus a subdivision of σ into $|\Xi| = O((\nu_s \log \nu_s)^j) = O(\nu_s^{d+1})$ cells.

LEMMA 3.8. *For any $T \subseteq [0,\tau]$, $M_{f,T}(h) \le h\, p^j \tau^{c-1}(c - \log(1-p)\tau)$.*

Proof. We have $f'(t) = p^j(1-p)^t(ct^{c-1} + \log(1-p)t^c)$. By the mean value theorem, for any $t_1, t_2 \in T$,

$$|f(t_2) - f(t_1)| \le |t_2 - t_1| \sup_{t\in[t_1,t_2]} |f'(t)|,$$

from which the result follows. (The reader should keep in mind that $1-p<1$, hence the minus sign in front of the logarithm when taking the absolute values of f'.) ∎

It now suffices to note, by the $(1/\nu_s)$-net property and since for any $\xi \in \Xi$ we have $\sup T_\xi \le n_s$, that

$$\Delta_\xi(f) \le M_{f,T_\xi}\left(\frac{n_s}{\nu_s}\right) = O\left(\frac{n_s}{\nu_s} p^j (1 + \frac{r}{n} n_s) n_s^{c-1}\right) = O\left(\frac{q_s p^j}{\nu_s} n_s^c\right) \tag{3.17}$$

for any $\xi \in \Xi$, where we have taken into account the fact that $-\log(1-p) = O(r/n)$.

To obtain the total approximation error made by the oracle, we substitute (3.16), (3.17) into (3.15), and this yields

$$\begin{aligned}\left|\Sigma_\sigma f - \tilde{\Sigma}_\sigma f\right| &\le \sum_{\xi\in\Xi} \left|\Sigma_\xi f - \tilde{\Sigma}_\xi f\right| \le O(\nu_s^{d+1})\,\frac{n_s^j}{\rho_s} f_\sigma^{max} + O(n_s^j)\frac{q_s p^j}{\nu_s} n_s^c \\ &= O\left(\frac{\nu_s^{d+1} q_s^d}{\rho_s} + \frac{q_s^{d+1}}{\nu_s}\right) n_s^c = O(E_s/C)\,,\end{aligned}$$

which validates (3.7), (3.10). ∎

As mentioned in the lemma, the time needed to answer a call to the oracle within a cell σ also depends on the complexity $\mathrm{compl}(\sigma)$ of that cell. However, the total complexity of all the cells in the sets Σ_{below} and Σ_{on} introduced in processing a hyperplane h_i is easily seen to be in $O(r_s^d)$; hence, the total time spent by the oracle when processing a hyperplane is $O_C(r_s^d q_s^{b\sqrt{c}})$.

With this lemma we can finish the time analysis of the whole algorithm. We have already seen at the end of section 3.2 that the running time of the deterministic algorithm is no more than the expected running time of the randomized algorithm

plus the cost of computing the good sample and that of the oracle calls. The total time spent for computing local arrangements and testing each hyperplane during the round is easily shown to be $O_C(nr^{\lfloor d/2\rfloor-1})$ using the fact that the geode of R defines a semicutting. This does not account for the oracle costs. There are at most n_s hyperplanes to process within the simplex s during the computation, so the total time needed will be at most proportional to

$$\sum_{s\in\mathcal{G}(R)} n_s r_s^d q_s^{b\sqrt{c}} \le \frac{n}{r}\left(\sum_{s\in\mathcal{G}(R)} r_s^{2d} + \sum_{s\in\mathcal{G}(R)} q_s^{2b\sqrt{c}+2}\right),$$

which is $O_C(nr^{\lfloor d/2\rfloor-1})$, by a calculation similar to that given at the end of the proof of Lemma 3.6. This is also the total running time for one round, which is then bounded by the expected running time of one round for the randomized algorithm. To summarize see the following theorem.

THEOREM 3.9. *The algorithm presented above computes the convex hull of n points deterministically in time $O(n\log n + n^{\lfloor d/2\rfloor})$ for any fixed dimension $d \ge 2$.*

One final note is needed for the model of computation. The algorithm given here works in the so-called real-RAM model [1], where elementary arithmetic operations take unit time regardless of the size of the numbers. This is the traditional model used in computational geometry, to be contrasted with the so-called bit-model [1], where the size of the numbers also contributes to the time complexity. Note that exponentially large quantities are needed in the course of the algorithm. Nevertheless, because the algorithm runs in polynomial time, it is possible, while computing over logarithmic-size words, to approximate any intermediate number in our algorithm with a relative error smaller than an arbitrarily small constant. It is easily seen that such errors are too small to be of consequence in the derandomization technique we use.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *Data Structures and Algorithms*, Addison-Wesley, Reading, MA, 1983.

[2] N. ALON AND J. SPENCER, *The Probabilistic Method*, John Wiley & Sons, New York, 1992.

[3] N. AMATO, M. GOODRICH, AND E. RAMOS, *Parallel algorithms for higher-dimensional convex hulls*, in Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci., Santa Fe, NM, 1994, pp. 683–694.

[4] N. AMATO, M. GOODRICH, AND E. RAMOS, *Computing faces in segment and simplex arrangements*, in Proc. 27th ACM Symp. Theor. Comput., Las Vegas, NV, 1995, pp. 672–682.

[5] H. BRÖNNIMANN, *Derandomization of Geometric Algorithms*, Ph.D. thesis, Dept. of Comput. Sci., Princeton University, Princeton, NJ, 1995.

[6] B. CHAZELLE, *Cutting hyperplanes for divide-and-conquer*, Discrete Comput. Geom., 9 (1993), pp. 145–158.

[7] B. CHAZELLE, *An optimal convex hull algorithm in any fixed dimension*, Discrete Comput. Geom., 10 (1993), pp. 377–409.

[8] B. CHAZELLE, H. EDELSBRUNNER, M. GRIGNI, L. J. GUIBAS, AND M. SHARIR, *Improved bounds on weak ε-nets for convex sets*, Discrete Comput. Geom., 13, (1995), pp. 1–15.

[9] B. CHAZELLE AND J. FRIEDMAN, *A deterministic view of random sampling and its use in geometry*, Combinatorica, 10 (1990), pp. 229–249.

[10] B. CHAZELLE AND J. MATOUŠEK *On linear-time deterministic algorithms for optimization problems in fixed dimension*, J. Algorithms, 21 (1996), pp. 579–597.

[11] K. L. CLARKSON, *A randomized algorithm for closest-point queries*, SIAM J. Comput., 17 (1988), pp. 830–847.

[12] K. L. CLARKSON, *Randomized Geometric Algorithms*, in Computing in Euclidean Geometry, Lecture Notes Series Comput. 1, D.-Z. Du, F. K. Kwang, eds.,World Scientific, River Edge, NJ, 1992, pp. 117–162.

[13] K. L. Clarkson and P. W. Shor, *Applications of random sampling in computational geometry*, II, Discrete Comput. Geom., 4 (1989), pp. 387–421.
[14] R. M. Dudley and R. S. Wenocur, *Some special Vapnik-Chervonenkis classes*, Discrete Math., 33 (1981), pp. 313–318.
[15] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, New York, 1987.
[16] H. Edelsbrunner and E. P. Mücke, *Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms*, ACM Trans. Graph., 2 (1990), pp. 66–104.
[17] D. Haussler and E. Welzl, *ε-nets and simplex range queries*, Discrete Comput. Geom., 2 (1987), pp. 127–151.
[18] J. Matoušek, *Construction of ε-nets*, Discrete Comput. Geom., 5 (1990), pp. 427–448.
[19] J. Matoušek, *Cutting hyperplane arrangements*, Discrete Comput. Geom., 6 (1991), pp. 385–406.
[20] J. Matoušek, *Approximations and optimal geometric divide-and-conquer*, J. Comput. System Sci., 50 (1995), pp. 203–208.
[21] J. Matoušek, *Efficient partition trees*, Discrete Comput. Geom., 8 (1992), pp. 315–334.
[22] H. Niederreiter, *Random Number Generation and Quasi Monte-Carlo Methods*, CBMS-NSF 63, SIAM, Philadelphia, PA, 1992.
[23] P. Raghavan, *Probabilistic construction of deterministic algorithms: Approximating packing integer programs*, J. Comput. System Sci., 37 (1988), pp. 130–143.
[24] J. Spencer, *Ten Lectures on the Probabilistic Method*, CBMS-NSF, SIAM, Philadelphia, PA, 1987.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1576–1587

AUTOMATIC NONZERO STRUCTURE ANALYSIS*

AART J. C. BIK† AND HARRY A. G. WIJSHOFF‡

Abstract. The efficiency of sparse codes heavily depends on the size and structure of the input data. Peculiarities of the nonzero structure of each sparse matrix must be accounted for to avoid unsatisfying performance. Therefore, it is important to have an efficient analyzer that automatically determines characteristics of nonzero structures. In this paper, some efficient algorithms are presented that automatically detect particular nonzero structures.

Key words. nonzero structures, sparse computations, sparse matrices

AMS subject classifications. 68-04, 65F50

PII. S009753979529595X

1. Introduction.

Many methods have been developed that exploit the sparsity of matrices to reduce the storage requirements and computational time of particular applications (see, e.g., [9, 12, 15, 17, 20, 23]). The efficiency of each individual method, however, heavily depends on the specific characteristics of the nonzero structure of each sparse matrix. For example, although a particular band method may perform extremely well when actually applied to matrices with a small bandwidth, the explicit storage and manipulation of all elements within the band makes this method infeasible for sparse matrices in which the nonzero elements are scattered over the entire matrix. Because peculiarities of nonzero structures must be accounted for to obtain satisfactory performance, it is important to have an efficient analyzer that automatically determines certain characteristics of nonzero structures.

Such a nonzero structure analyzer can be used in a number of different fashions. First, if sparse applications are explicitly coded by hand, a nonzero structure analyzer can provide a programmer with useful insights about the characteristics of the matrices for which an application must be developed in case a representative set of sparse matrices is available beforehand. Although in this situation the efficiency of the analyzer is less important, excessively long running times would disable the analysis of large sets of matrices.

Second, in the past we have proposed a completely different approach to the development of sparse codes [1]. Rather than explicitly dealing with the sparsity of matrices at programming level, as done traditionally, this sparsity is dealt with at compilation level by a special kind of restructuring compiler, referred to as a "sparse compiler." We must refer to [1, 3, 4, 5] for a detailed presentation of this approach and some preliminary experiments with a prototype sparse compiler. It is obvious, however, that the automatically generated sparse code becomes more efficient if the sparse compiler can account for characteristics of the nonzero structures. For this,

*Received by the editors December 13, 1995; accepted for publication (in revised form) July 1, 1997; published electronically April 27, 1999. This research was supported by the Foundation for Computer Science (SION) of the Dutch Organization for Scientific Research (NWO) and the EC Esprit Agency DG XIII under grant APPARC 6634 BRA III. A preliminary version of this paper appears as "Nonzero structure analysis," in *ICS '94. Proc. 8th International Conference on Supercomputing*, July 11–15, 1994, Manchester, UK, ACM, New York, 1994, pp. 226–235.

http://www.siam.org/journals/sicomp/28-5/29595.html

†Intel Corp., 2200 Mission College Blvd., SC12-303, Santa Clara, CA 95052 (aart.bik@intel.com).

‡Computer Science Department, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands (harryw@cs.leidenuniv.nl).

the compiler requires an automatic nonzero structure analyzer. Since in this approach analysis time contributes to compile time, it is again desirable to keep analysis time limited.

In most practical cases, however, sparse matrices are not all available beforehand. A possible approach to deal with this situation is to generate multiple versions of an application (either explicitly by hand or automatically by means of a sparse compiler), each of which has been optimized specifically for a particular class of nonzero structures. At run-time, the analyzer is invoked to determine which version is probably the most efficient. This version is subsequently executed. Using run-time analysis has the major advantage that nonzero structures do not have to be known in advance. In this case, however, the analyzer must be very efficient to avoid the situation in which savings in execution time using an optimized version are outweighed by analysis time. In general, it is desirable to keep analysis time proportional to the number of nonzero elements and order of a sparse matrix [11].

In this paper, some efficient algorithms are presented that can be used by an analyzer to automatically detect particular nonzero structures of square sparse matrices. The presented algorithms examine each matrix as it is; i.e., no attempts are made to permute the matrix into a particular form (as is frequently done in the context of LU-factorization, for example, to confine fill-in to certain regions in the matrix [6, 7, 8, 10, 12, 14, 15, 16, 17, 22]). Even if such a permutation is applied to the matrix, however, the analyzer can be used afterwards to determine whether an unforeseen nonzero structure arises (information about the specific form for which the permutation is intended is usually obtained as a side effect of computing the permutation).

2. Nonzero structures. We can distinguish between general sparse matrices and sparse matrices having a particular nonzero structure. In this section, some important nonzero structures of square matrices are identified [12, 18, 19, 20, 21].

2.1. Band forms. The *lower* and *upper semibandwidth* of an $N \times N$ matrix A are defined as the smallest integers $b_l \geq 0$ and $b_u \geq 0$, respectively, for which $(a_{ij} \neq 0) \Rightarrow (-b_u \leq i - j \leq b_l)$ still holds. Minimum values reveal the most information about the nonzero structure, because this constraint is trivially satisfied for semibandwidths $N - 1$. Allowing for negative semibandwidths would enable the specification of an arbitrary band in which the main diagonal is not necessarily included. However, in this paper we will assume that all matrices have a full transversal (i.e., all elements on the main diagonal are nonzero).

If the semibandwidths are relatively small, we say that the matrix is in *band form*, which means that all nonzero elements are confined to a small band. We define the shape count of a band form as the total number of elements that reside within the band:

$$N \cdot (b_l + b_u + 1) - (b_l^2 + b_l)/2 - (b_u^2 + b_u)/2.$$

Note that this number is likely to exceed the total number of nonzero elements, because the band is not necessarily full.

2.2. Block forms. Consider a block partition of a square matrix A into submatrices A_{ij}:

$$A = \begin{pmatrix} A_{11} & \dots & A_{1p} \\ \vdots & \ddots & \\ A_{p1} & & A_{pp} \end{pmatrix}.$$

Each submatrix A_{ii}, referred to as a *diagonal block*, is a square $n_i \times n_i$ submatrix. Hence, each submatrix A_{ij} with $i \neq j$, referred to as an *off-diagonal block*, is an $n_i \times n_j$ submatrix. The off-diagonal blocks A_{pi} and A_{ip} for $1 \leq i < p$ are referred to as the *lower border* and *upper border*, respectively. If a block A_{ij} contains at least one nonzero element, the block is called a *nonzero block*, denoted by $A_{ij} \neq 0$.

If $(A_{ij} \neq 0) \Rightarrow (i = j)$, then the matrix is in *block diagonal form*. Likewise, if $(A_{ij} \neq 0) \Rightarrow (i \geq j)$, or if $(A_{ij} \neq 0) \Rightarrow (i \leq j)$, then the matrix is in *block lower triangular form* or *block upper triangular form*, respectively. If a matrix is in block diagonal form except for some nonzero blocks in the borders, then the matrix is in *doubly bordered block diagonal form*. Matrices in *singly bordered block lower triangular form* or *singly bordered block upper triangular form* are defined likewise. Finally, if $p = 2$, $A_{21} \neq 0$, $A_{12} \neq 0$, and A_{11} is in band form, we say that the matrix is in *doubly bordered band form*.

For these forms, the shape count is defined as the total number of elements in the diagonal blocks (but only counting elements in the band of A_{11} for a bordered band form), *all* border blocks for the bordered forms, and, for the triangular forms, *all* remaining off-diagonal blocks in the lower or upper triangular part, even if not all these blocks are nonzero. Again, this number is likely to exceed the total number of nonzero elements, since the blocks are not necessarily full (and some of them may even be zero).

Although, depending on which blocks are nonzero, a particular form of a matrix is defined once a block partition of that matrix is given, it is possible that *different* block partitions into one particular form differ in the accuracy of describing the nonzero structure. In Figure 2.1, for example, two different block partitions of a matrix into block diagonal form are shown with shape counts 15 and 25, respectively. Therefore, we say the most accurate description for a particular form is defined by a *minimum block partition into that form*, which means that there are no other block partitions of the matrix into the same form with a smaller shape count.

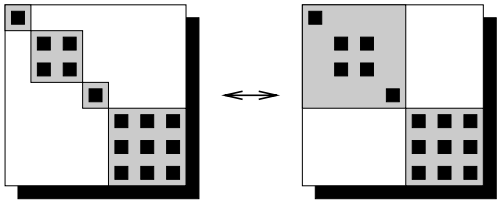


FIG. 2.1. *Two different block partitions into BDF.*

A square matrix has a unique minimum block partition into block diagonal form, which satisfies the following property (similar statements hold for block lower or upper triangular forms).

PROPOSITION 2.1. *A block partition of a square matrix into block diagonal form is minimum if and only if there is no diagonal block with a nontrivial block partition into block diagonal form.*

We will see that a matrix can have several minimum block partitions into a bordered band or block form.

3. Automatic nonzero structure analysis. In this section, efficient algorithms are presented that detect (bordered) band and (bordered) block forms. We assume that the nonzero structure of each $N \times N$ sparse matrix A to be analyzed is available on file in coordinate scheme. In this scheme, the file consists of the order N and an integer nnz that indicates the number of nonzero elements, followed by an unordered set of nnz triples (i, j, a_{ij}), indicating the row index, column index, and value of each individual nonzero element.

First, sky-line information is computed. Thereafter, this information is used to detect particular nonzero structures.

3.1. Preparatory analysis. Sky-line information, i.e., the lower and upper semibandwidth for each single row and column, respectively, can be obtained in a single pass over a file by executing the following procedure. In this fragment, the lower and upper sky-line are computed in the arrays `lsky` and `usky`, respectively:

```
procedure comp_skylines()
begin
  read(N, nnz);

  allocate lsky[1:N] and usky[1:N]
  for i := 1, N do
    lsky[i] := 0;
    usky[i] := 0;
  endfor

  for k := 1, nnz do
     read(i, j, aij);
     lsky[i] := max(lsky[i], (i-j));
     usky[j] := max(usky[j], (j-i));
  endfor
end
```

Sky-lines are computed under the assumption that the matrix has a full transversal, so that all elements of the arrays `lsky` and `usky` can be initialized to zero. All information requires $\Theta(N)$ storage and can be obtained in $\Theta(nnz + N)$ time.

Example. The following lower and upper sky-lines are obtained for the 15×15 sparse matrix that is depicted in Figure 3.1:

	1														15
lsky	0	0	1	0	4	0	1	0	1	0	0	0	0	0	3
usky	0	0	2	0	1	0	0	0	3	0	0	0	0	3	2

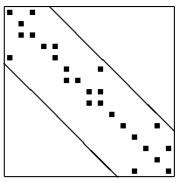


FIG. 3.1. *Band form.*

3.2. Band forms. Once the lower and upper sky-lines of a matrix have been computed, the lower and upper semibandwidths of this matrix are determined in $\Theta(N)$ time as follows:

```
procedure comp_bandform()
begin
  b_l := 0; b_u := 0;
  for i := 1, N do
    b_l := max(b_l, lksy[i]);
    b_u := max(b_u, uksy[i]);
  endfor
end
```

These semibandwidths directly determine the band form of a matrix.

Example. For the sparse matrix of Figure 3.1, the semibandwidths $b_l = 4$ and $b_u = 3$ result. This gives rise to the band form with shape count 104 that is shown in the same figure.

3.3. Bordered band forms. By slightly extending the previous procedure, an algorithm is derived that constructs a minimum block partition into bordered band form in $\Theta(N)$ time:

```
procedure comp_bord_bandform()
begin
  b_l := 0; b_u := 0;
  tsz := N*N;

  for i := 1, N do
    b_l := max(b_l, lksy[i]);
    b_u := max(b_u, uksy[i]);
    if (bf_size(b_l, b_u, N-i) <= tsz) then
      t_l := b_l; t_u := b_u; b := N-i;
      tsz := bf_size(b_l, b_u, N-i);
    endif
  endfor
end
```

In this algorithm, the auxiliary function `bf_size` computes the shape count of a bordered band matrix with border size `b` and semibandwidths `bl` and `bu`:

```
integer function bf_size(bl, bu, b)
begin
  return ( (N-b) * (bl+bu+1) - (bl*bl+bl) / 2
           - (bu*bu+bu) / 2 + 2*N*b - b*b );
end
```

After the semibandwidths are updated in each step `i`, we test whether the shape count of a bordered band form with border size `N-i` is less than or equal to the best shape count seen so far. If this is true, we record this shape count and corresponding border size and semibandwidths. Consequently, after applying the algorithm, variables `b`, `t_l`, and `t_u` contain the border size and semibandwidths of a *minimum* block partition into block band form. If `b=0` holds, then effectively a band form results. For example, applying `comp_bord_bandform()` to the matrix of Figure 3.1 yields exactly the same band form as computed by `comp_bandform()`.

Example. In Figure 3.2, we present the resulting bordered band forms for some matrices, with shape counts 125, 119, 153, and 131, respectively. The last example shows that, although a minimum block partition into a *doubly* bordered band form is constructed, it is possible that only a single border block is actually nonzero.

Example. A matrix may have different minimum block partitions into bordered band form, as illustrated in Figure 3.3, where the shape count of all forms is 93. Our algorithm solves such ties in favor of the smallest border size.

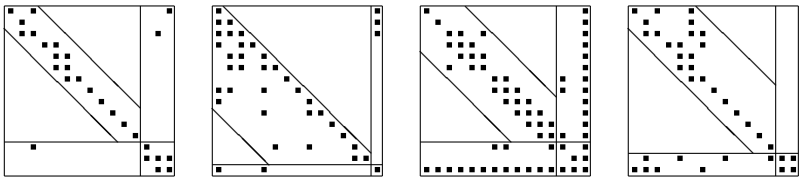


FIG. 3.2. *Bordered band forms.*

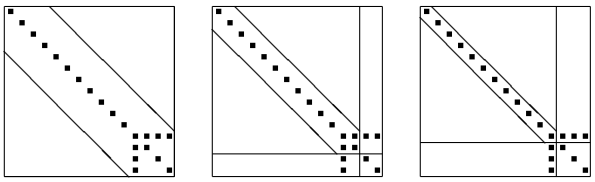


FIG. 3.3. *Different minimum block partitions into bordered band forms.*

3.4. Block forms. The following procedure constructs a block partition into diagonal block form in $\Theta(N)$ time by determining the size `k` of each next diagonal block with the lower right corner at row and column index `B` during a backward scan over the sky-lines:

```
procedure comp_blockdiag()
begin
  p := 0; k := 1; B := N;
  for i := N, 1, -1 do
S1: k := max(k, max(lsky[i],usky[i])+B-i+1);
    if (i = B-k+1) then
       p := p + 1;  part[p] := i;  /* Record Block */
       B := i - 1;        k := 1:  /* Next Block   */
    endif
  endfor
end
```

After application of this algorithm, `p` contains the number of diagonal blocks. The row (or column) indices of the upper left corners of all diagonal blocks of the block partition are recorded in reverse order in the first `p` locations of array `part`.

The following proposition states that the *minimum* block partition into block diagonal form is found. Likewise, if only the value `lsky[i]` or `usky[i]` is used in statement `S1`, then the minimum block partition into block lower, or block upper triangular form, respectively, is obtained in $\Theta(N)$ time.

PROPOSITION 3.1. *Application of* `comp_blockdiag()` *to the lower and upper sky-line of a matrix yields the minimum block partition into block diagonal form.*

Proof. By construction, each nonzero element is incorporated in a diagonal block. Now assume that the resulting block partition is not a minimum block partition into diagonal form. Then Proposition 2.1 implies that there is a certain $k \times k$ diagonal block with the lower right corner at a row and column index B that has a nontrivial

block partition into block diagonal form; i.e., there is $1 \leq k' < k$ such that $\forall B - k' < i \leq B : \max(l_i, u_i) + (B - i) < k'$. Since no diagonal block is recorded during iterations $\texttt{i} = B \ldots B - k' + 1$, during at least one of these iterations a value is assigned to `k` that is greater than k'. However, this can only occur if $\max(l_i, u_i) + B - i + 1 > k'$ for some $B - k' < i \leq B$. This contradicts the assumption.

Example. Application of the different versions of this algorithm to the matrix of Figure 3.1 yields the block diagonal, block lower, and upper triangular forms shown in Figure 3.4, with shape counts 67, 140, and 138, respectively. Note that although in the block triangular forms many off-diagonal blocks are zero, the elements of these blocks are also included in the shape count.

The contents of array `part` for the first block partition are shown below:

part	11	10	6	1

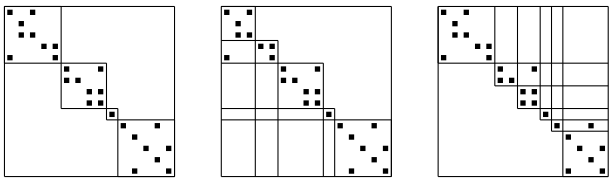


FIG. 3.4. *Block forms.*

3.5. Bordered block forms. Let $E(b)$ denote the shape count of a bordered block diagonal form with border size $b \in [0, N]$ arising from the minimum block partition of the remaining $(N - b) \times (N - b)$ matrix into block diagonal form. We define the improvement of using border size b' instead of b as $I(b', b) = E(b) - E(b')$, satisfying the following property.

PROPOSITION 3.2. *For $b, b', b'' \in [0, N]$, we have $I(b', b'') = I(b', b) + I(b, b'')$.*

Proof. $I(b', b'') = E(b) - E(b') + E(b'') - E(b) = I(b', b) + I(b, b'')$.

Suppose that for a given border size $b \in [0, N]$, we construct the minimum block partition of the remaining $(N - b) \times (N - b)$ submatrix into block diagonal form using the procedure `comp_diagblock()`. At any iteration $\texttt{i} = i$, the block partition found so far may be discarded and the algorithm may be restarted with $B = i - 1$ and $k = 1$ for a new border size $b' = N - i + 1$.

Obviously, selection of this border is only profitable if eventually we are able to determine that $I(b', b) > 0$. However, rather than constructing both block partitions completely, we are already able to compute the improvement during an iteration $\texttt{i} = i'$ in which the last diagonal block of the new block partition that overlaps with the diagonal block that was assumed during iteration $\texttt{i} = i$ has been found. This is *because the block partition of the remaining part of the matrix will be identical for both block partitions.* This new diagonal block may be contained in the old diagonal block (which occurs if the value of `k` would not have been incremented while constructing the old block partition), or these blocks may partially overlap.

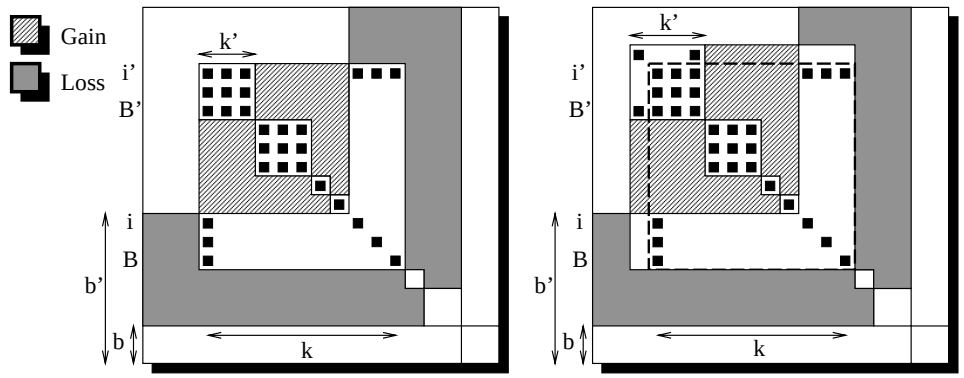


FIG. 3.5. *Gain and loss for border.*

Both cases are illustrated in Figure 3.5. In any case, the improvement is equal to the difference of the number of elements included in the border (*loss*) and the number of elements that do not have to be included in a diagonal block (*gain*). Let B, k and B', k' denote the value of `B` belonging to the iterations `i` $= i$ and `i` $= i'$, respectively. Furthermore, let Z and Z' denote the number of elements in the off-diagonal zero blocks of the old and new block partition below rows B and B', respectively. Then the improvement of using a new border size b' with respect to the old border size b is equal to the difference between the gain and the loss:

$$I(b', b) \quad = \quad Z' - Z - 2 \cdot (B' - k')(B - B').$$

If the gain exceeds the loss, i.e., $I(b', b) > 0$, then it is profitable to continue with the new block partition and border size b'. Moreover, border size b may be discarded from further consideration, since Proposition 3.2 implies that $I(b', b'') > I(b, b'')$ for all $b'' \in [0, N]$. If no improvement has been obtained, i.e., $I(b', b) \leq 0$, then the block partition corresponding to border size b must be restored and the algorithm can proceed with the search for the next diagonal block (which has at least size $\max(k, B - B' + k')$). In that case, we may discard border size b' from further consideration, since Proposition 3.2 implies that $I(b', b'') \leq I(b, b'')$ for all $b'' \in [0, N]$.

These observations enable us to construct a minimum block partition into block diagonal form in one pass over the sky-lines. At each step in which no diagonal block is recorded, the current status is saved on a stack, and a new border size is tried. If a diagonal block is recorded, no improvement can be obtained by trying a new border size. Instead, previously constructed block partitions belonging to smaller border sizes that can be verified are restored if an improvement is obtained (which is simply done by restoring the value of `p`), or discarded otherwise. The following slightly more complex version of procedure `comp_blockdiag()` results:

```
procedure comp_bord_blockdiag()
begin
  Z := 0; b := 0; s := 0;
  p := 0; k := 1; B := N;
  for i := N, 1,  -1 do
S2: k := max(k, max(lsky[i],usky[i])+B-i+1);
    if (i = B-k+1) then
       /* Last Overlapping Block? */
       while ( (s > 0) && (i == stackB[s]-new_k()+1) ) do /* Conditional AND */
          /* Improvement? */
          if (I() > 0) then
            s := s - 1;            /* Discard         */
          else
            pop_restore();         /* Restore         */
          endif
       endfor
       Z := Z + 2 * k * (B-k);
       p := p + 1;  part[p] := i;  /* Record Block    */
       B := i - 1;        k := 1:  /* Next Block      */
    else
       push();                     /* Save State      */
       Z := 0; B := i - 1;         /* New Search      */
       k := 1; b := N - i + 1;
    endif
  endfor
end
```

In this algorithm, the following auxiliary procedures are used to implement stack-like operations that save and restore states:

```
procedure push()
begin
  s := s + 1;
  stackk[s] := k;
  stackZ[s] := Z;
  stackB[s] := B;
  stackp[s] := p;
  stackb[s] := b;
end
```

```
procedure pop_restore()
begin
  k := new_k();
  Z := stackZ[s];
  B := stackB[s];
  p := stackp[s];
  b := stackb[s];
  s := s - 1;
end
```

The following auxiliary functions are used to compute the improvement and the new value of `k` for the block partition on top of the stack:

```
integer function I()
begin
 I := Z - stackZ[s] - 2 *
      (B-k) * (stackB[s]-B);
end
```

```
integer function new_k()
begin
  new_k := max(stackk[s],
               stackB[s]-B+k);
end
```

Although a while-loop occurs inside the `i`-loop, this algorithm still runs in $\Theta(N)$ time because each border size can only be pushed and popped from the stack once. Because the algorithm simply applies `comp_blockdiag()` to the submatrix that remains for the most profitable border size, it is clear that this extended algorithm constructs a *minimum* block partition into bordered block diagonal form.

After application of this algorithm, the scalar `b` contains the selected border size

(and hence the size of the last diagonal block). The first `p` locations of array `part` represent the block partition into block diagonal form of the remaining submatrix. If a zero border size is selected, the last diagonal block is empty and, effectively, a block partition into block diagonal form results.

If only the value `lsky[i]` or `usky[i]` is used in statement `S2`, then a minimum block partition into, respectively, singly bordered block lower or upper triangular form is obtained. In these cases, the constant 2 must be removed from the assignment to `Z` and the computation in function `I()` to compute the appropriate improvement.

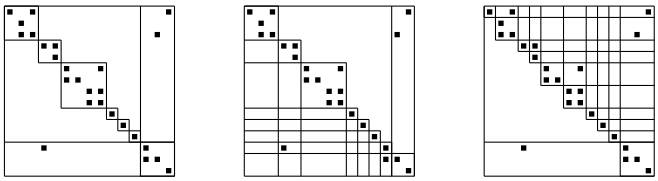


FIG. 3.6. *Bordered block forms.*

Example. In Figure 3.6, the bordered block forms that result for a matrix are shown, having shape counts $225 - 112 = 113$, $225 - 68 = 157$, and $176 - 14 = 162$, respectively. The contents of array `part` for the bordered block diagonal form having `b=3` are shown below:

part	12	11	10	6	4	1

Example. Applying the version operating only on `lsky[i]` to the matrix with 22 nonzero elements of Figure 3.7 yields a minimum partition into bordered block upper triangular form with border size 1 and shape count 166. However, the shape counts of similar forms with border sizes 2 and 3 are also 166. This example illustrates that a matrix may have different minimum block partitions into a particular bordered block form. Because a border is denied for a zero improvement (viz., $I(3,2) = 0$ and $I(2,1) = 0$ in the example), ties are solved in favor of the smallest border size.

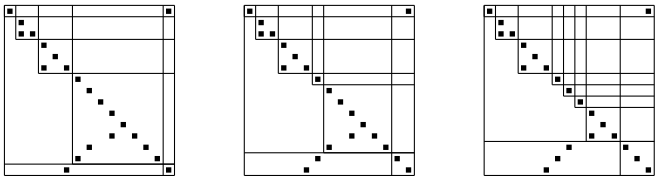


FIG. 3.7. *Different minimum block partitions into BBUTF.*

3.6. Classification. After the minimum block partitions into (doubly bordered) band form, (doubly bordered) block diagonal form, and (singly bordered) block lower and upper triangular forms have been constructed, shape counts can be used to determine which form most accurately describes the nonzero structure of a matrix. Moreover, the density of this form (i.e., *nnz* divided by the shape count) can be compared with a user-defined threshold to decide whether this nonzero structure is

used for the classification of the sparse matrix or whether the matrix is classified as a general sparse matrix.

Example. For the 59×59 matrix "impcol_b" of the Harwell–Boeing sparse matrix collection [13] with 312 nonzero elements, for example, the shape counts of the different forms are 2620, 3461, 3206, and 2930, respectively. In Figure 3.8, we show the band form and bordered block upper triangular form. Here, we classify this matrix as a band matrix if 312/2620 exceeds the threshold, or as a general sparse matrix otherwise.

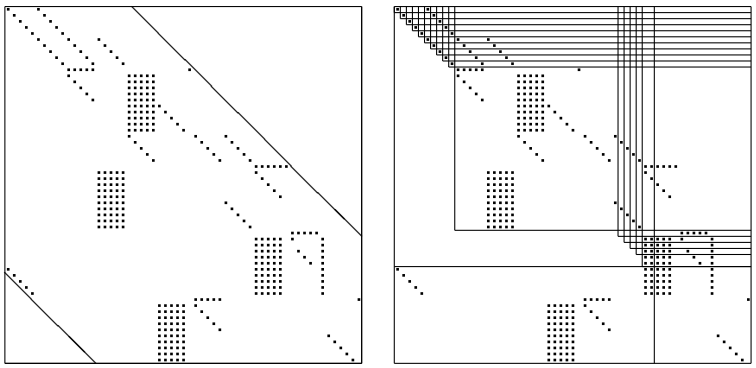


FIG. 3.8. *Classification of "impcol_b."*

4. Conclusions. In this paper, we have presented some efficient algorithms that automatically detect particular nonzero structures of sparse matrices. First, we have extended an algorithm that constructs a band form into an algorithm that constructs a minimum block partition into bordered band form (possibly yielding a band form as a special case). Likewise, an algorithm that constructs a minimum block partition into block diagonal or triangular form has been extended into an algorithm that constructs a minimum block partition into bordered block diagonal or triangular form (possibly yielding a block form with an empty border as a special case). All algorithms require only sky-line information, which can be obtained in $\Theta(N + nnz)$ time for an $N \times N$ sparse matrix with nnz nonzero elements, and have a running time of $\Theta(N)$.

REFERENCES

[1] A. J. C. BIK, *Compiler Support for Sparse Matrix Computations*, Ph.D. thesis, Department of Computer Science, Leiden University, Leiden, The Netherlands, 1996.

[2] A. J. C. BIK AND H. A. G. WIJSHOFF, *Nonzero structure analysis*, in ICS '94. Proc. 8th International Conference on Supercomputing, July 11–15, 1994, Manchester, UK, ACM, New York, 1994, pp. 226–235.

[3] A. J. C. BIK AND H. A. G. WIJSHOFF, *Advanced compiler optimizations for sparse computations*, J. Parallel Distrib. Comput., 31 (1995), pp. 14–24.

[4] A. J. C. BIK AND H. A. G. WIJSHOFF, *Automatic data structure selection and transformation for sparse matrix computations*, IEEE Trans. Parallel Distrib. Systems, 7 (1996), pp. 109–126.

[5] A. J. C. BIK AND H. A. G. WIJSHOFF, *The use of iteration space partitioning to construct representative simple sections*, J. Parallel Distrib. Comput., 34 (1996), pp. 95–110.

[6] T. F. COLEMAN, *Large Sparse Numerical Optimization*, Lecture Notes in Comput. Sci. 165, Springer-Verlag, Berlin, 1984.

[7] E. CUTHILL, *Several strategies for reducing the bandwidth of matrices*, in Sparse Matrices and Their Applications, D. J. Rose and R. A. Willoughby, eds., Plenum Press, New York, 1972, pp. 157–166.

[8] E. Cuthill and J. Mckee, *Reducing the bandwidth of sparse symmetric matrices*, in Proc. 24th National Conference of the ACM, 1969, pp. 157–172.
[9] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelphia, 1991.
[10] I. S. Duff, *A survey of sparse matrix research*, Proc. IEEE, (1977), pp. 500–535.
[11] I. S. Duff, *A sparse future*, in Sparse Matrices and Their Uses, I. S. Duff, ed., Academic Press, London, 1981, pp. 1–29.
[12] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford Science Publications, Oxford, 1990.
[13] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.
[14] A. George and J. W. H. Liu, *An implementation of a pseudoperipheral node finder*, ACM Trans. Math. Software, 5 (1979), pp. 284–295.
[15] A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[16] K. J. Mann, *Inversion of large sparse matrices: Direct methods*, in Numerical Solutions of Partial Differential Equations, J. Noye, ed., North-Holland, Amsterdam, 1982, pp. 313–366.
[17] S. Pissanetsky, *Sparse Matrix Technology*, Academic Press, London, 1984.
[18] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes*, Cambridge University Press, Cambridge, 1986.
[19] R. P. Tewarson, *Sorting and ordering sparse linear systems*, in Large Sparse Sets of Linear Equations, J. K. Reid, ed., Academic Press, London, 1971, pp. 151–167.
[20] R. P. Tewarson, *Sparse Matrices*, Academic Press, New York, 1973.
[21] M. Veldhorst, *An Analysis of Sparse Matrix Storage Schemes*, Ph.D. thesis, Mathematisch Centrum, Amsterdam, 1982.
[22] R. Wait, *The Numerical Solution of Algebraic Equations*, Wiley, Chichester, 1979.
[23] Z. Zlatev, *Computational Methods for General Sparse Matrices*, Kluwer, Dordrecht, 1991.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1588–1626

STACK AND QUEUE LAYOUTS OF DIRECTED ACYCLIC GRAPHS: PART II*

LENWOOD S. HEATH† AND SRIRAM V. PEMMARAJU‡

Abstract. Stack layouts and queue layouts of undirected graphs have been used to model problems in fault tolerant computing and in parallel process scheduling. However, problems in parallel process scheduling are more accurately modeled by stack and queue layouts of *directed acyclic graphs* (dags). A *stack layout* of a dag is similar to a stack layout of an undirected graph, with the additional requirement that the nodes of the dag be in some topological order. A *queue layout* is defined in an analogous manner. The *stacknumber* (*queuenumber*) of a dag is the smallest number of stacks (queues) required for its stack layout (queue layout). This paper presents algorithmic results—in particular, linear time algorithms for recognizing 1-stack dags and 1-queue dags, and proofs of NP-completeness for the problem of recognizing a 4-queue dag and the problem of recognizing a 6-stack dag. The companion paper (Part I [*SIAM J. Comput.,* 28 (1999), pp. 1510–1539.]) presents combinatorial results.

Key words. stack layout, queue layout, book embedding, graph embedding, directed acyclic graphs, dags, graph algorithms, leveled-planar graphs, NP-complete, posets

AMS subject classifications. 05C99, 68Q15, 68Q25, 68R10, 94C15

PII. S0097539795291550

Introduction. This is a companion paper to Heath, Pemmaraju, and Trenk [9]; we assume familiarity with the definitions, notation, and results in section 1 of that paper. There, we define stack and queue layouts of directed acyclic graphs (dags) and establish the stacknumber and queuenumber of path dags, cycle dags, tree dags, and unicyclic dags. We also provide a forbidden subgraph characterization of 1-queue tree dags and 1-queue cycle dags. In the current paper, we study stack and queue layouts of dags from an algorithmic point of view and present four algorithmic results.

First, we present a linear time algorithm that recognizes 1-stack dags. Recall that the class of 1-stack undirected graphs is equal to the class of outerplanar graphs (see Bernhart and Kainen [1]). Therefore, the linear time outerplanar graph recognition algorithm of Syslo and Iri [12] can be used to determine whether or not an undirected graph has a 1-stack layout. However, not all dags with outerplanar covering graphs are 1-stack dags and this makes it more difficult to recognize 1-stack dags (for an example, see Figure 1.1).

Second, we present a linear time algorithm to recognize 1-queue dags. Our algorithm uses a variation of the PQ-tree data structure introduced by Booth and Lueker [2]. This result solves an open problem of Heath, Pemmaraju, and Trenk [10] and brings out a contrast between 1-queue undirected graphs and 1-queue dags, because Heath and Rosenberg [11] have shown that the problem of recognizing 1-queue undirected graphs is NP-complete. The main step of our 1-queue dag recognition algorithm determines whether a given dag has a leveled-planar embedding. This result extends

*Received by the editors September 11, 1995; accepted for publication (in revised form) May 7, 1997; published electronically May 6, 1999. This research was supported in part by National Science Foundation grant CCR-9009953.

http://www.siam.org/journals/sicomp/28-5/29155.html

†Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061-0106 (heath@cs.vt.edu).

‡Department of Computer Science, University of Iowa, Iowa City, IA 52242-1316 (sriram@cs.uiowa.edu).

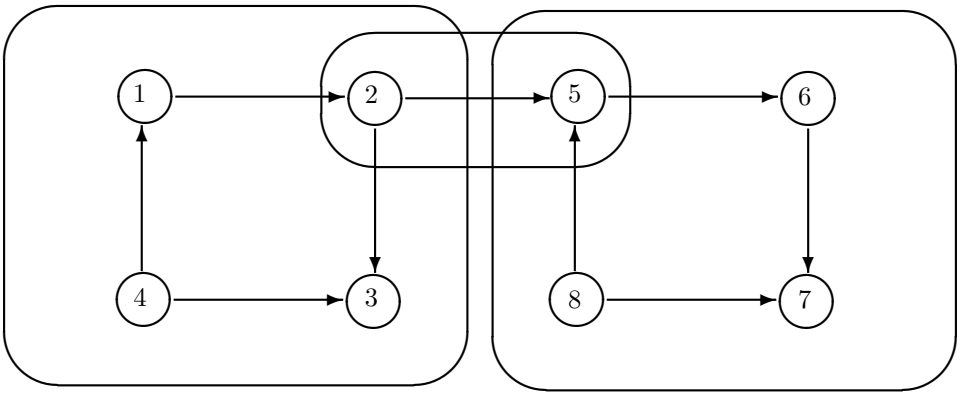


FIG. 1.1. *A* 2*-stack dag, each of whose biconnected components are* 1*-stack dags.*

the work of Di Battista and Nardelli [5] and Chandramouli and Diwan [3, 4]. These authors assume certain restrictions on the given dag. In particular, Di Battista and Nardelli present an algorithm to determine if a dag with a single source (node with in-degree 0) has a leveled-planar embedding. Chandramouli and Diwan present an algorithm to determine if a triconnected dag has a leveled-planar embedding.

Third, we prove that the problem of recognizing a 4-queue dag is NP-complete. In fact, our result is stronger: we show that recognizing a 4-queue poset (a 4-queue dag with certain restrictions) is NP-complete. Fourth and finally, we prove that recognizing a 6-stack dag is also NP-complete.

The organization of the rest of the paper is as follows. In section 1, we present a linear time algorithm for recognizing 1-stack dags. In section 2, we present a linear time algorithm for recognizing 1-queue dags. In section 3, we show that the problem of recognizing 4-queue dags is NP-complete, and finally, in section 4, we show that the problem of recognizing 6-stack dags is NP-complete.

1. Recognizing 1-stack dags. In this section, we present an $O(|V|)$ time algorithm for determining whether a dag $\vec{G} = (V, \vec{E})$ is a 1-stack dag. If $\vec{G}$ is a 1-stack dag, then the algorithm constructs a 1-stack layout of $\vec{G}$. A dag is a 1-stack dag if and only if each of its connected components is a 1-stack dag. Hence we assume, without loss of generality, that $\vec{G}$ is connected. Also, for $\vec{G}$ to be a 1-stack dag its covering graph $G = (V, E)$ has to be a 1-stack graph. By Bernhart and Kainen's characterization [1], this means that G has to be outerplanar, which in turn means that $|E| \leq 2|V| - 3$. Since $|\vec{E}| = |E|$, without loss of generality, we assume that $|\vec{E}| \leq 2|V| - 3$.

Bernhart and Kainen show that the stacknumber of an undirected graph is the maximum of the stacknumbers of its biconnected components. The analogous result does not hold for dags, as shown in Figure 1.1. The dag shown in this figure contains three biconnected components: one induced by the nodes $\{1, 2, 3, 4\}$, one induced by the nodes $\{2, 5\}$, and one induced by the nodes $\{5, 6, 7, 8\}$. For emphasis, each of the biconnected components is enclosed in an oval in Figure 1.1. Each of the biconnected components is a 1-stack dag, but we may verify that the dag itself cannot be laid out in one stack. Therefore, to verify that a dag is a 1-stack dag, it is not sufficient to check that each biconnected component is a 1-stack dag.

We organize our algorithm to recognize 1-stack dags in two steps. In the first step (section 1.1), we verify that each biconnected component of $\vec{G}$ is a 1-stack dag. If the

algorithm finds a biconnected component that is not a 1-stack dag, then it terminates immediately with failure. In the second step (section 1.2), we combine the 1-stack layouts of the biconnected components of $\vec{G}$ into a 1-stack layout of $\vec{G}$.

1.1. Biconnected dags. This step decomposes $\vec{G}$ into biconnected components and verifies that each biconnected component of $\vec{G}$ is a 1-stack dag. The verification depends on the following lemma.

LEMMA 1.1. *A biconnected dag $\vec{B} = (V, \vec{E})$ is a 1-stack dag if and only if $\vec{B}$ is an outerplanar dag and contains a directed Hamiltonian path obtained by traversing the outer face of an outerplanar embedding of $\vec{B}$.*

Proof. Suppose that $\vec{B}$ is an outerplanar dag containing a directed Hamiltonian path obtained by traversing the outer face of some outerplanar embedding of $\vec{B}$. This directed Hamiltonian path gives a unique topological order, say σ, of $\vec{B}$. It follows immediately from Bernhart and Kainen's characterization of 1-stack undirected graphs that σ yields a 1-stack layout of B. Since σ is a topological order of $\vec{B}$, σ also yields a 1-stack layout of $\vec{B}$.

To establish the converse, suppose that $\vec{B}$ is a biconnected 1-stack dag. Since its covering graph B is a 1-stack graph, B is outerplanar. Since B is biconnected and outerplanar, it contains a unique Hamiltonian cycle; call it C. Let $\vec{C}$ be the dag that is a subgraph of $\vec{B}$ and that has C as its covering graph. Clearly $\vec{C}$ is a 1-stack dag. By Lemma 2.2 of [9], $\vec{C}$ contains a unique directed Hamiltonian path; say it is given by $v_1, v_2, \ldots, v_n$. As a dag can have at most one Hamiltonian path, this is also the unique Hamiltonian path in $\vec{B}$. Then $\sigma = v_1, v_2, \ldots, v_n$ is the unique topological order of $\vec{B}$ that yields a 1-stack layout of $\vec{B}$. This 1-stack layout of $\vec{B}$ can be viewed, in a natural way, as an outerplanar embedding whose outer face can be traversed in the order given by σ. The lemma follows. □

Depth-first search on $\vec{G}$ can be used to decompose $\vec{G}$ into biconnected components. This requires $O(|V|)$ time, since depth-first search of G requires $O(|V| + |E|)$ time and $|E| \leq 2|V| - 3$. For each biconnected component $\vec{B}$ of $\vec{G}$, it is easy to combine a topological sort with a verification that $\vec{B}$ contains a directed Hamiltonian cycle and that the topological order yields a 1-stack layout of $\vec{B}$. The time complexity of the first step is $O(|V|)$.

1.2. General dags. This step combines the 1-stack layouts of the biconnected components of $\vec{G}$ into a 1-stack layout of $\vec{G}$. By Lemma 1.1, each of the 1-stack biconnected components of $\vec{G}$ has a unique source (node with in-degree 0), a unique sink (node with out-degree 0), and a unique 1-stack layout. For this step, we need the notion of a block-cutpoint tree, defined by Harary and Prins [6]. The *block-cutpoint tree* $T(\vec{G})$ of a dag $\vec{G}$ is the undirected graph with vertex set

$$\left\{\vec{B} \mid \vec{B} \text{ is a biconnected component of } \vec{G}\right\} \cup \left\{u \mid u \text{ is a cutpoint in } \vec{G}\right\}$$

and edge set

$$\left\{\{\vec{B}, u\} \mid u \text{ is a cutpoint in } \vec{B}\right\}.$$

Figure 1.2 shows the block-cutpoint tree of the dag in Figure 1.1. The larger circles, labeled $\vec{B}_1$, $\vec{B}_2$, and $\vec{B}_3$, represent the three biconnected components, while the smaller circles, labeled 2 and 5, represent cutpoints. Harary and Prins [6] show that $T(\vec{G})$ is a tree if and only if $\vec{G}$ is connected.

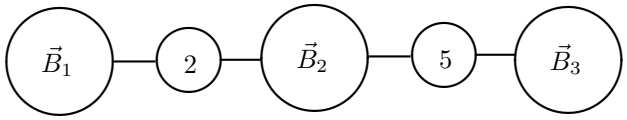


FIG. 1.2. *The block-cutpoint tree of the dag shown in Figure* 1.1.

An *intermediate node* in a biconnected component $\vec{B}$ is a node in $\vec{B}$ that is neither the source nor the sink in $\vec{B}$. In Figure 1.1, node 2 is an intermediate node in $\vec{B}_1$, and node 5 is an intermediate node in $\vec{B}_5$. For any biconnected component $\vec{B}$ in $\vec{G}$, we use $T(\vec{G}, \vec{B})$ to denote $T(\vec{G})$ rooted at $\vec{B}$. If x is a node in $\vec{B}$ that is a cutpoint of $\vec{G}$, then we use $D(\vec{B}, x)$ to denote the set of nodes in the subtree of $T(\vec{G}, \vec{B})$ rooted at x. For example, for the dag and its block-cutpoint tree shown in Figures 1.1 and 1.2, $D(\vec{B}_3, 5) = \{5, \vec{B}_2, 2, \vec{B}_1\}$ and $D(\vec{B}_1, 2) = \{2, \vec{B}_2, 5, \vec{B}_3\}$.

Two cutpoints u and v of $\vec{G}$ constitute a *conflicting pair of cutpoints* if u is an intermediate node in a biconnected component $\vec{B}_i$, v is an intermediate node in a biconnected component $\vec{B}_j$, $\vec{B}_i \in D(\vec{B}_j, v)$, and $\vec{B}_j \in D(\vec{B}_i, u)$. For example, the dag in Figure 1.1 contains the conflicting pair of cutpoints 2 and 5. This is because 2 is an intermediate node in $\vec{B}_1$, 5 is an intermediate node in $\vec{B}_3$, $\vec{B}_1 \in D(\vec{B}_3, 5)$, and $\vec{B}_3 \in D(\vec{B}_1, 2)$. Note that it is possible that u and v coincide.

The following lemma gives a partial characterization of 1-stack dags.

LEMMA 1.2. *Suppose the dag $\vec{G}$ contains a conflicting pair of cutpoints. Then $\vec{G}$ is not a* 1*-stack dag.*

Proof. Let u and v be a conflicting pair of cutpoints in $\vec{G}$. Let $\vec{B}_i$ and $\vec{B}_j$ be as in the definition of a conflicting pair. If either $\vec{B}_i$ or $\vec{B}_j$ is not a 1-stack dag, then $\vec{G}$ is not a 1-stack dag, and we are done. So assume that each of these biconnected component is a 1-stack dag. By Lemma 1.1, each biconnected component has a unique 1-stack layout, where the topological order is given by the unique Hamiltonian path in the biconnected component. Let $u_1, u_2, \ldots, u_t$ be that order on the nodes in $\vec{B}_i$ that yields a unique 1-stack layout of $\vec{B}_i$. By Lemma 1.1, (u_1, u_t) is an arc of $\vec{B}_i$. Since u is an intermediate node in $\vec{B}_i$, $u = u_k$, for some k, $1 < k < t$. Since $\vec{B}_i \in D(\vec{B}_j, v)$ and $\vec{B}_j \in D(\vec{B}_i, u)$, there is an undirected path P in G between cutpoints u and v such that no internal node in P belongs to either $\vec{B}_i$ or $\vec{B}_j$. Let σ be any topological ordering of $\vec{G}$. Clearly, $u_1, u_2, \ldots, u_t$ is a subsequence of σ. Based on where v occurs in σ relative to this subsequence, there are five cases:

1. $u = v$. Then $v = u_k$. Since v is an intermediate node in $\vec{B}_j$, there is a node v' immediately before v in the 1-stack layout of $\vec{B}_j$; moreover, v' is not in $\vec{B}_i$. Note that we may assume that $v' <_\sigma u_{k-1}$; if not, just switch the roles of $\vec{B}_i$ and $\vec{B}_j$. We consider two possibilities for the order of v' with respect to the nodes of $\vec{B}_i$. In the first case, suppose that the order σ satisfies

$$v' <_\sigma u_1 <_\sigma u_2 <_\sigma \cdots <_\sigma u_k = v <_\sigma u_{k+1} <_\sigma \cdots <_\sigma u_t.$$

In this case, the arc (u_1, u_t) crosses the arc (v', v), and σ does not yield a 1-stack layout of $\vec{G}$. In the second case, suppose that the order σ satisfies

$$u_m <_\sigma v' <_\sigma u_{m+1} <_\sigma \cdots <_\sigma u_k = v <_\sigma u_{k+1}$$

for some m with $1 \leq m < k-1$. In this case, the arc (u_m, u_{m+1}) crosses the arc (v', v), and σ does not yield a 1-stack layout of $\vec{G}$. Hence, in both cases, σ does not yield a 1-stack layout of $\vec{G}$.

2. $v <_\sigma u_1$ or $u_t <_\sigma v$. In either case, an edge in P crosses (u_1, u_t) and therefore σ does not yield a 1-stack layout of $\vec{G}$.
3. $u_m <_\sigma v <_\sigma u_{m+1}$, where $1 \leq m < k-1$ or $k+1 \leq m < t$. In either case, an edge in P crosses (u_m, u_{m+1}) and therefore σ does not yield a 1-stack layout $\vec{G}$.
4. $u_{k-1} <_\sigma v <_\sigma u_k$. Then, for any node w in $\vec{B}_j$, it is the case that $u_{k-1} <_\sigma w <_\sigma u_k$, because otherwise an arc in $\vec{B}_j$ would cross arc (u_{k-1}, u_k). But the fact that $u_{k-1} <_\sigma w <_\sigma u_k$ for all nodes w in $\vec{B}_j$ implies that an edge in P crosses arc (v', v''), where v' and v'' are the source and sink, respectively, of $\vec{B}_j$ (here we use the fact that v is an intermediate node in $\vec{B}_j$). Therefore, σ does not yield a 1-stack layout of $\vec{G}$.
5. $u_k <_\sigma v <_\sigma u_{k+1}$. An argument similar to the one employed in case 3 shows that σ does not yield a 1-stack layout of $\vec{G}$.

In all cases, σ does not yield a 1-stack layout of $\vec{G}$. But σ is an arbitrary topological ordering. Hence $\vec{G}$ is not a 1-stack dag, as desired. □

To combine the biconnected components of $\vec{G}$ into a 1-stack layout of $\vec{G}$, we first order the biconnected components of $\vec{G}$ via a breadth-first search of $T(\vec{G})$. Let an arbitrary biconnected component $\vec{B}_1$ be the root of the breadth-first search, and let $\vec{B}_1, \vec{B}_2, \ldots, \vec{B}_m$ be the order obtained for the biconnected components. Also, let T be the rooted version of $T(\vec{G})$ so obtained.

For each i, where $1 \leq i \leq m$, let $\vec{G}_i$ denote the subgraph of $\vec{G}$ induced by the biconnected components $\vec{B}_1, \vec{B}_2, \ldots, \vec{B}_i$. In particular, $\vec{G}_1 = \vec{B}_1$ and $\vec{G}_m = \vec{G}$. Let u be a cutpoint of $\vec{G}$, and let $\vec{B}_k$ be the parent of u in T. Then u is *restricted in* $\vec{G}_i$ if u belongs to $\vec{G}_i$ and some cutpoint $v \in D(\vec{B}_k, u)$ is an intermediate node in some biconnected component $\vec{B}_j$, where $j > i$ and $\vec{B}_j$ is a child of v. Note that it is possible that $v = u$ in which case $\vec{B}_j$ is a child of u. Consider Figure 1.3. Assume that v is an intermediate node in $\vec{B}_7$. Then the cutpoint u is restricted in both $\vec{G}_2$ and $\vec{G}_3$.

The following lemma places an upper bound on the number of cutpoints in $\vec{B}_i$ that can be restricted in $\vec{G}_i$.

LEMMA 1.3. *Suppose that $\vec{G}$ does not contain a conflicting pair of cutpoints. Then the number of cutpoints in $\vec{B}_i$ that are restricted in $\vec{G}_i$ is at most* 1. *Further suppose that $\vec{G}$ contains a restricted cutpoint u whose parent in T is $\vec{B}_k$. Then u is either the source or the sink of $\vec{B}_k$.*

Proof. Suppose that u is a restricted cutpoint in $\vec{G}_i$, and $\vec{B}_k$ is its parent in T. By definition, $k \leq i$ and there exists some cutpoint $v \in D(\vec{B}_k, u)$ that is an intermediate node in some biconnected component $\vec{B}_j$, where $j > i$ and $\vec{B}_j$ is a child of v. First suppose that u is an intermediate node of $\vec{B}_k$. Then $\vec{B}_i \in D(\vec{B}_j, v)$ and $\vec{B}_j \in D(\vec{B}_i, u)$, implying that u and v constitute a conflicting pair of cutpoints in $\vec{G}$, contrary to assumption. Hence only two nodes in $\vec{B}_k$, the source and the sink of $\vec{B}_k$, may be restricted in $\vec{G}_i$.

Now suppose that u and u' are distinct restricted cutpoints in $\vec{G}$. Suppose that $\vec{B}_k$ is the parent of u in T and $\vec{B}_{k'}$ the parent of u'. Note that it is possible that $k = k'$, in which case one of u and u' is the source and the other the sink of $\vec{B}_k$. By definition, $k \leq i$ and there exists some cutpoint $v \in D(\vec{B}_k, u)$ that is an intermediate

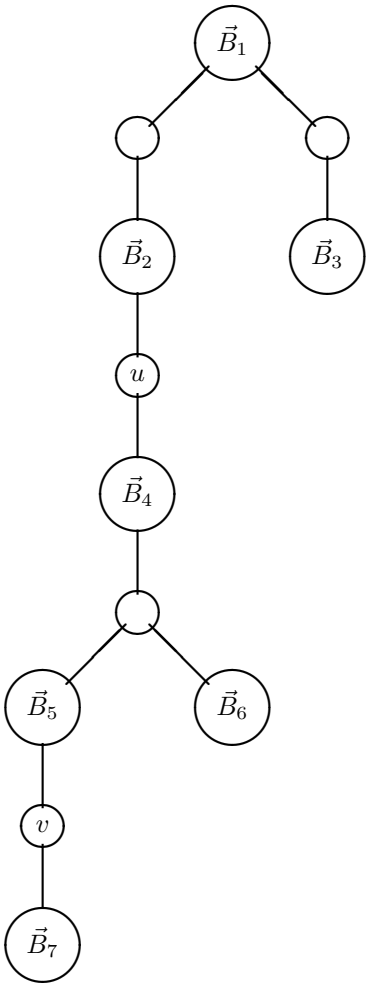


FIG. 1.3. *Node u is restricted in both $\vec{G}_2$ and $\vec{G}_3$.*

node in some block $\vec{B}_j \in D(\vec{B}_k, u)$, where $j > i$ and $\vec{B}_j$ is a child of v. Similarly, $k' \leq i$ and there exists some cutpoint $v' \in D(\vec{B}'_k, u')$ that is an intermediate node in some block $\vec{B}_{j'} \in D(\vec{B}'_k, u')$, where $j' > i$ and $\vec{B}_{j'}$ is a child of v'. It is easy to see that $v \neq v'$, that $v \in D(\vec{B}_{j'}, v')$, and that $v' \in D(\vec{B}_j, v)$. Hence, v and v' constitute a conflicting pair of cutpoints in $\vec{G}$, contrary to assumption.

We conclude that at most one cutpoint in $\vec{G}$ is restricted in $\vec{G}_i$. The lemma follows. □

The main result is now given by the following theorem.

THEOREM 1.4. *Let $\vec{G} = (V, E)$ be a dag that does not contain a pair of conflicting cutpoints and that does not contain a biconnected component with stacknumber exceeding* 1. *Then $\vec{G}$ is a* 1*-stack dag. Further, a* 1*-stack layout of $\vec{G}$ can be constructed in $O(|V| + |E|)$ time.*

Proof. The proof consists of an algorithm that actually constructs a 1-stack layout of $\vec{G}$. The algorithm first preprocesses the biconnected components of $\vec{G}$ in the order $\vec{B}_m, \vec{B}_{m-1}, \ldots, \vec{B}_1$ so as to compute information about any restricted node in each

$\vec{G}_i$. Clearly, $\vec{B}_m$ does not contain any nodes that are restricted in $\vec{G}_m$. For each i, $1 \leq i < m$, we can compute the set of nodes in $\vec{B}_i$ that are restricted in $\vec{G}_i$ as follows. Let u be a node in $\vec{B}_i$ that is a cutpoint of $\vec{G}$. Then, u is restricted in $\vec{G}_i$ if and only if u is shared by a biconnected component $\vec{B}_j$, $j > i$, and either u is an intermediate node in $\vec{B}_j$ or $\vec{B}_j$ contains nodes that are restricted in $\vec{G}_j$. Based on the above observation, for each u in $\vec{B}_i$ that is a cutpoint in $\vec{G}$, in the worst case, it takes time proportional to the number of children u has in T to determine whether u is restricted in $\vec{G}_i$. Thus, the entire preprocessing step takes time proportional to the number of edges in T. Since the number of nodes in T is at most $3|V| - 3$, the time complexity of the entire preprocessing step is $O(|V|)$.

After the preprocessing step, if we find that some $\vec{B}_i$ has two or more cutpoints that are restricted in $\vec{G}_i$, then (by Lemma 1.3) $\vec{G}$ contains a conflicting pair of cutpoints and (by Lemma 1.2) $\vec{G}$ does not have a 1-stack layout. So we may assume, without loss of generality, that after the preprocessing step each biconnected component $\vec{B}_i$ has at most one cutpoint that is restricted in $\vec{G}_i$.

A node u is said to be *exposed* in a topological order σ of $\vec{G}$ if there is no arc $(v, w) \in \vec{E}$ such that $v <_\sigma u <_\sigma w$. The algorithm now proceeds to process the biconnected components of $\vec{G}$ in forward order so as to maintain the following induction hypothesis.

INDUCTION HYPOTHESIS. *For each $i \geq 1$, we have that*

(a) *$\vec{G}_i$ has a 1-stack layout;*

(b) *if u is restricted in $\vec{G}_i$, then $\vec{G}_i$ has a 1-stack layout in which u is exposed.*

Note that Lemma 1.3 implies that there is at most one cutpoint that is restricted in any $\vec{G}_i$. We now prove the base case and the inductive step.

Base case. $\vec{B}_1$ has a unique 1-stack layout in which the source and the sink are exposed and the rest of the nodes are not. Thus item (a) of the induction hypothesis is satisfied. Lemma 1.3 implies that if a cutpoint in $\vec{B}_1$ is restricted in $\vec{G}_1$, then it is either the source or the sink of $\vec{B}_1$, both of which are exposed. Thus item (b) of the induction hypothesis is satisfied.

Inductive step. We now add the 1-stack layout of $\vec{B}_{i+1}$, as computed in section 1.1, to the 1-stack layout of $\vec{G}_i$ already constructed, and show that a 1-stack layout of $\vec{G}_{i+1}$ results that satisfies the induction hypothesis. Let $\sigma = u_1, u_2, \ldots, u_t$ be the unique order on the nodes in $\vec{B}_{i+1}$ that yields the 1-stack layout. Let u_m, $1 \leq m \leq t$, be the cutpoint of $\vec{G}$ through which $\vec{B}_{i+1}$ is connected to $\vec{G}_i$. There are two cases depending on whether or not $\vec{B}_{i+1}$ contains a cutpoint that is restricted in $\vec{G}_{i+1}$:

1. *$\vec{B}_{i+1}$ contains a cutpoint v that is restricted in $\vec{G}_{i+1}$.* This implies that the cutpoint u_m is restricted in $\vec{G}_i$ because either $v = u_m$ or u_m has a child $\vec{B}_{i+1}$ in T and $\vec{B}_{i+1}$ has a child in T, namely v, that is restricted in $\vec{G}_{i+1}$. Therefore, by the induction hypothesis item (b), u_m is exposed in the 1-stack layout of $\vec{G}_i$ constructed so far. Place the nodes $u_1, u_2, \ldots, u_{m-1}$ in that order to the left of the nodes in $\vec{G}_i$ and place the nodes $u_{m+1}, u_{m+2}, \ldots, u_t$ in that order to the right of the nodes in $\vec{G}_i$. Since u_m is exposed in the layout of $\vec{G}_i$, we obtain a 1-stack layout of $\vec{G}_{i+1}$, thus satisfying item (a) in the induction hypothesis.

   By Lemma 1.3, v is either a source ($v = u_1$) or v is a sink ($v = u_t$) in $\vec{B}_{i+1}$. Since both the source and the sink are exposed in the 1-stack layout of $\vec{G}_{i+1}$ constructed, item (b) of the induction hypothesis is satisfied.
2. *$\vec{B}_{i+1}$ does not contain a cutpoint that is restricted in $\vec{G}_{i+1}$.* To obtain a 1-stack

layout of $\vec{G}_{i+1}$, we place the nodes $u_1, \ldots, u_{m-1}$ in that order immediately to the left of u_m and the nodes $u_{m+1}, \ldots, u_t$ in that order immediately to the right of u_t. To show that this placement of the nodes in $\vec{B}_{i+1}$ yields a 1-stack layout of $\vec{G}_{i+1}$, we need to consider two possibilities with respect to cutpoint u_m. Either u_m is restricted in $\vec{G}_i$ or u_m is not restricted in $\vec{G}_i$.

If u_m is restricted in $\vec{G}_i$, then by induction hypothesis item (b), it is exposed in the 1-stack layout of $\vec{G}_i$ constructed so far. Hence, placing the nodes in $\vec{B}_{i+1}$ as described above yields a 1-stack layout of $\vec{G}_{i+1}$, thus satisfying induction hypothesis item (a). If u_m is restricted in $\vec{G}_{i+1}$ also, then by Lemma 1.3, u_m is either a source ($m = 1$) or a sink ($m = t$) of $\vec{B}_{i+1}$ and remains exposed in the layout of $\vec{G}_{i+1}$.

If u_m is not restricted in $\vec{G}_i$, then u_m cannot be an intermediate node in $\vec{B}_{i+1}$. Hence u_m is either the source ($m = 1$) or the sink ($m = t$) of $\vec{B}_{i+1}$. Again, placing the nodes in $\vec{B}_{i+1}$ as described above yields a 1-stack layout of $\vec{G}_{i+1}$. Item (b) of the induction hypothesis is satisfied because the cutpoints that are restricted in $\vec{G}_{i+1}$ are the same as the cutpoints that are restricted in $\vec{G}_i$ and all nodes that are exposed in the 1-stack layout of $\vec{G}_i$ remain exposed in the 1-stack layout of $\vec{G}_{i+1}$.

By induction, a 1-stack layout of $\vec{G}$ results.

Because the 1-stack layout of $\vec{B}_{i+1}$ is constructed in linear time in section 1.1, it takes $O(|\vec{B}_{i+1}|)$ time to extend the 1-stack layout of $\vec{G}_i$ to the 1-stack layout of $\vec{G}_{i+1}$. Thus constructing the 1-stack layout of $\vec{G}_m$ takes a total of $O(m)$ time. Since $m \leq |\vec{E}| \leq 2|V| - 3$, the time complexity of the algorithm is $O(|V|)$. □

In contrast to the ease of recognizing 1-stack dags, witness the NP-completeness of recognizing 6-stack dags shown in Theorem 4.1.

2. Recognizing 1-queue dags. In this section, we present an $O(|V|)$ time algorithm to determine whether a dag $\vec{G} = (V, \vec{E})$ is a 1-queue dag and, if so, to construct a 1-queue layout of $\vec{G}$. Heath and Rosenberg [11] characterize 1-queue undirected graphs as arched leveled-planar graphs and show that the problem of recognizing arched leveled-planar graphs is NP-complete. Analogously, Heath, Pemmaraju, and Trenk [9] characterize 1-queue dags as arched leveled-planar dags. In this section, we present a linear time algorithm for recognizing arched leveled-planar dags. Thus, recognizing 1-queue dags turns out to be significantly simpler than recognizing 1-queue undirected graphs. The development of our algorithm requires two steps, which are outlined here and subsequently elaborated upon:

1. *Recognize leveled-planar dags.* The first step is an algorithm that takes a leveled dag as input and determines whether or not the dag has a leveled-planar embedding. This is the more complicated of the two steps. Note the contrast with Heath and Rosenberg's result [11] in which they show that the problem of recognizing leveled-planar undirected graphs is NP-complete.
2. *Recognize* 1*-queue dags.* The second step is an extension of the first step and is the algorithm we seek. Given a dag as input, it first finds a maximal leveled subgraph of a particular kind by removing certain nonleveled arcs. It then applies the algorithm from the first step to determine whether the subgraph is a leveled-planar dag, while modifying the algorithm to take into account the nonleveled arcs.

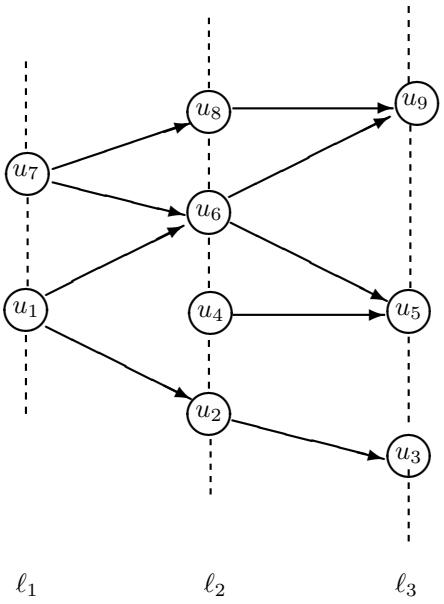


FIG. 2.1. *A leveled-planar dag.*

The first step of our algorithm, the one that recognizes leveled-planar dags, was sketched in [7] and extends the work of Di Battista and Nardelli [5] and Chandramouli and Diwan [3, 4]. These authors assume certain restrictions on the given dag. In particular, Di Battista and Nardelli [5] present a linear time algorithm to determine if a given *hierarchy* has a leveled-planar embedding. A hierarchy is a leveled dag with a single source. Hierarchies are widely used in many fields of social and mathematical sciences, and a common procedure for improving the readability of a drawing of a hierarchy is to minimize the number of edge crossings. Chandramouli and Diwan [3] present a linear time algorithm to determine if a given triconnected dag has a leveled-planar embedding. They point out that their algorithm can be used to solve a problem related to grid intersection graphs. These authors leave open the problem of determining whether or not an arbitrary dag is leveled-planar.

We proceed as follows. Section 2.1 discusses the problem of recognizing leveled-planar dags. Section 2.2 defines the data structures (PQ-trees and collections) that we need to represent sets of permutations of nodes in a particular level. We outline the algorithm to recognize a leveled-planar dag in section 2.3. Section 2.4 defines the operations we use to restrict or combine sets of permutations. Section 2.5 presents the details of our linear time algorithm for recognizing leveled-planar dags and proves its correctness and time complexity. Section 2.6 extends that algorithm to a linear time algorithm for recognizing 1-queue dags. The reader should review the definitions and notation from section 1 of [9], especially topological order, leveling, directed leveled-planar embedding, and directed arched leveled-planar embedding. Figure 2.1 shows a directed leveled-planar embedding of a dag.

2.1. The problem of recognizing leveled-planar dags. We concentrate first on the problem of recognizing whether a dag $\vec{G} = (V, \vec{E})$ is a leveled-planar dag. It is easy to check in linear time, using a graph traversal technique such as depth-first search, whether a dag $\vec{G} = (V, \vec{E})$ is leveled. If it is, then fix a leveling lev :

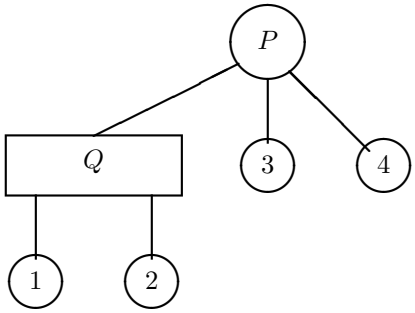


FIG. 2.2. *A PQ-tree.*

$V \to \{1, 2, \ldots, m\}$ of $\vec{G}$, for some integer m. Define $V_i = lev^{-1}(i)$. Without loss of generality, we may assume that $V_i \neq \emptyset$ for every i with $1 \leq i \leq m$. We write $\vec{G} = (V_1, V_2, \ldots, V_m; \vec{E})$ and henceforth assume that $\vec{G}$ is a connected, leveled dag.

The problem remaining is to determine whether $\vec{G}$ has a directed leveled-planar embedding.

Suppose $\vec{G}$ has a directed leveled-planar embedding $\mathcal{E}$. As $\vec{G}$ is connected, without loss of generality, we may assume that the leveling induced by $\mathcal{E}$ is *lev*. For each j, where $1 \leq j \leq m$, $\mathcal{E}$ determines a total order $\leq_j$ on V_j given by the bottom-to-top order of the nodes on ℓ_j. Conversely, if a total order $\leq_j$ on V_j is given for each j, then it is easy to check whether these total orders yield a directed leveled-planar embedding of $\vec{G}$. It suffices to check that there are no two arcs (u, v) and (x, y) such that $lev(u) = lev(x) = j$, $u <_j x$, and $y <_{j+1} v$. In Figure 2.1, the total orders are given by $u_1 <_1 u_7$, $u_2 <_2 u_4 <_2 u_6 <_2 u_8$, and $u_3 <_3 u_5 <_3 u_9$.

The problem of recognizing whether a connected leveled dag $\vec{G}$ is a leveled-planar dag is then equivalent to determining whether there are total orders on all m levels that together yield a leveled-planar embedding of $\vec{G}$. Each total order is a permutation of the nodes in that level. As our algorithm needs to represent many such permutations for each level, we introduce suitable data structures in the next section.

2.2. PQ-trees and collections. A classical data structure used to represent sets of permutations is the PQ-tree of Booth and Lueker [2]. A PQ-tree T for a set S is a rooted tree that contains three types of nodes: leaves, P-nodes (each drawn as a circle), and Q-nodes (each drawn as a box). The leaves in T are in one-to-one correspondence with the elements of S. The set S is called the *yield* of T, denoted YIELD(T). It is clear that we can iterate over the set YIELD(T) in time Θ(YIELD(T)) by simply traversing the tree in, say, preorder. The PQ-tree T represents permutations of YIELD(T) according to the following rules:

- The children of a P-node may be permuted arbitrarily.
- The children of a Q-node must occur in the given order or in the reverse order.

As a special case, the empty PQ-tree ϵ represents the empty set of permutations. For example, the PQ-tree shown in Figure 2.2 represents these 12 permutations

$$\begin{array}{llllll} 1,2,3,4 & 1,2,4,3 & 4,1,2,3 & 3,1,2,4 & 3,4,1,2 & 4,3,1,2 \\ 2,1,3,4 & 2,1,4,3 & 4,2,1,3 & 3,2,1,4 & 3,4,2,1 & 4,3,2,1 \end{array}$$

of $S = \{1, 2, 3, 4\}$. The set of permutations represented by a PQ-tree T is denoted by PERM(T). The *yield* YIELD(r) of a node r in T is the yield of the subtree rooted

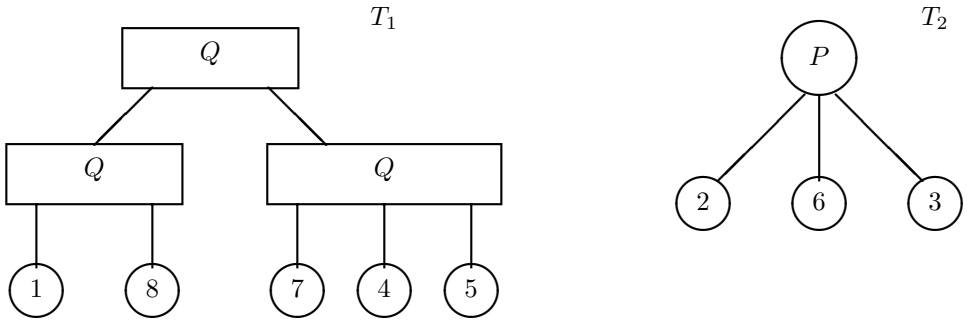


FIG. 2.3. *A collection.*

at r. Without loss of generality, we may assume that every P-node has three or more children and that every Q-node has two or more children.

A *collection* C is a finite set of PQ-trees with pairwise disjoint yields. The *yield* of C, denoted YIELD(C), is the union of the yields of its constituent trees. PERM(C), the set of permutations represented by C, consists of those permutations π of YIELD(C) such that, for each $T \in C$, π restricted to YIELD(T) is in PERM(T). An example of a collection containing two PQ-trees is shown in Figure 2.3. The PQ-tree T_1 represents these eight permutations

$$\begin{array}{cccc} 1,8,7,4,5 & 1,8,5,4,7 & 8,1,7,4,5 & 8,1,5,4,7 \\ 7,4,5,1,8 & 5,4,7,1,8 & 7,4,5,8,1 & 5,4,7,8,1 \end{array}$$

of $\{1,4,5,7,8\}$. The PQ-tree T_2 represents all six permutations of $\{2,3,6\}$. The collection C represents all $8 \cdot 6 \cdot (\frac{8!}{5! \cdot 3!}) = 2688$ permutations of $\{1,2,3,4,5,6,7,8\}$ that are consistent with both T_1 and T_2.

We wish to represent all possible leveled-planar embeddings of a connected, leveled dag by permutations of nodes in its "rightmost" level. To this end, we make the following definition: Suppose that $\vec{F}$ is a leveled, connected dag. Fix a leveling of $\vec{F}$. Suppose that level k is the largest nonempty level. We say that a PQ-tree T *mirrors* $\vec{F}$ if PERM(T) is the set of all permutations of the level-k nodes in $\vec{F}$ that witness some leveled-planar embedding of $\vec{F}$. If we take $\vec{F}$ to be the dag in Figure 2.1, then a PQ-tree consisting of a Q-node as root with children $3,5,9$ (in that order) mirrors $\vec{F}$.

For some $\vec{F}$, the PQ-tree $T[\vec{F}]$ that mirrors $\vec{F}$ is easily described. For example, suppose that $\vec{F}$ is a tree dag with a single source s and all arcs directed away from s. Such a dag is clearly a leveled-planar dag, with a unique leveling such that $lev(s) = 1$. Then the PQ-tree $T[\vec{F}]$ is essentially isomorphic to $\vec{F}$. To obtain $T[\vec{F}]$ from $\vec{F}$ simply replace each internal (nonleaf) node in $\vec{F}$ by a P-node. The root of $T[\vec{F}]$ is the node that replaced s. $T[\vec{F}]$ might contain P-nodes that have fewer than three children. To get rid of such nodes, iteratively replace every P-node, whose only child is a leaf, by that leaf. Then replace every P-node with two children by a Q-node.

This definition of mirroring does not extend to a leveled dag that is not connected. As an illustration, suppose $\vec{F}'$ is the leveled dag induced by the left two levels in Figure 2.1. In a leveled-planar embedding of $\vec{F}'$, the nodes 2, 6, and 8 must appear in that order or in the reverse order, but the node 4 may appear anywhere with respect to these nodes. It is clear that a single PQ-tree cannot represent the corresponding set of permutations, although a collection of two PQ-trees can.

2.3. Overview of the algorithm to recognize leveled-planar dags. In this section, we give an overview of our algorithm that determines whether a given connected, leveled dag $\vec{G} = (V_1, V_2, \ldots, V_m, \vec{E})$ is leveled-planar. For any j, $1 \leq j \leq m$, let $\vec{G}_j$ denote the subgraph of $\vec{G}$ induced by $V_1 \cup V_2 \cup \cdots \cup V_j$. Note that, unlike $\vec{G}$, the subgraph $\vec{G}_j$ is not necessarily connected. For each level V_j, we say that a permutation π of the nodes in V_j is a *witness* to a directed leveled-planar embedding $\mathcal{E}$ of $\vec{G}_j$ if the nodes in V_j appear in a bottom-to-top order on line ℓ_j according to π in $\mathcal{E}$. For each j, there is a set of permutations Π_j that contains all witnesses to directed leveled-planar embeddings of $\vec{G}_j$. So to recognize whether $\vec{G}$ is a leveled-planar dag, we need only compute Π_m and check that it is nonempty. Our basic approach to doing this efficiently is to perform a left-to-right sweep processing the levels in the order $V_1, V_2, \ldots, V_m$. For any dag $\vec{H}$, let $\mathrm{COMP}(\vec{H})$ be the set of connected components of $\vec{H}$. For each $\vec{F} \in \mathrm{COMP}(\vec{G}_j)$, we construct a PQ-tree $T[\vec{F}]$ that mirrors $\vec{F}$. We use a collection C_j to contain PQ-trees $T[\vec{F}]$ for all $\vec{F} \in \mathrm{COMP}(\vec{G}_j)$. Clearly, $\mathrm{YIELD}(C_j) = V_j$. Furthermore, any permutation of V_j that is a witness to a directed leveled-planar embedding of $\vec{G}_j$ is in $\mathrm{PERM}(C_j)$. However, if $\vec{G}_j$ is not connected, then there may be a permutation in $\mathrm{PERM}(C_j)$ that is not a witness to any directed leveled-planar embedding of $\vec{G}_j$. The algorithm then processes V_{j+1} and derives the collection C_{j+1} from the collection C_{j+1}. The collections maintained by our algorithm satisfy the following invariant.

Collection invariant. *For each j, where $1 \leq j \leq m$, and for each $\vec{F} \in \mathrm{COMP}(\vec{G}_j)$, there is a corresponding PQ-tree $T[\vec{F}]$ in C_j that mirrors $\vec{F}$.*

Since $\vec{G} = \vec{G}_m$ is connected, the collection invariant implies that C_m contains a single PQ-tree $T[\vec{G}]$ that represents $\vec{G}$. So C_m contains a nonempty PQ-tree if and only if $\vec{G}$ has a directed leveled-planar embedding. Thus the goal of our algorithm is to compute C_m.

The evolution of C_{j+1} from C_j requires that some information be maintained in each nonleaf node of a PQ-tree and one additional piece of information be maintained at the root. Let $\vec{F}$ be any connected component of $\vec{G}_j$. By the collection invariant, $T[\vec{F}]$ is the PQ-tree in C_j that mirrors $\vec{F}$. For any subset S of the set of nodes in V_j that belongs to $\vec{F}$, define $\mathrm{MEETLEVEL}(S)$ to be the greatest $d \leq j$ such that $V_d, \ldots, V_j$ induces a dag in which all nodes of S occur in the same connected component. For example, in Figure 2.1, $\mathrm{MEETLEVEL}(\{u_3, u_5\}) = 1$ and $\mathrm{MEETLEVEL}(\{u_5, u_9\}) = 2$. Note that if $|S| > 1$, then $\mathrm{MEETLEVEL}(S) < j$. For a Q-node q in $T[\vec{F}]$ with ordered children $r_1, r_2, \ldots, r_t$, maintain in node q, integers denoted $\mathrm{ML}(r_i, r_{i+1})$, where $1 \leq i < t$, that satisfy

$$\mathrm{ML}(r_i, r_{i+1}) = \mathrm{MEETLEVEL}(\mathrm{YIELD}(r_i) \cup \mathrm{YIELD}(r_{i+1})).$$

For a P-node p in $T[\vec{F}]$, maintain in node p a single integer denoted $\mathrm{ML}(p)$ that satisfies

$$\mathrm{ML}(p) = \mathrm{MEETLEVEL}(\mathrm{YIELD}(p)).$$

Let S be the set of nodes in V_j that belong to $\vec{F}$. Define $\mathrm{LEFTLEVEL}(S)$ to be the smallest d such that $\vec{F}$ contains a node in V_d. We always have

$$\mathrm{LEFTLEVEL}(S) \leq \mathrm{MEETLEVEL}(S),$$

and inequality is possible. At the root of $T[\vec{F}]$, maintain a single integer denoted $\mathrm{LL}(T[\vec{F}])$ satisfying

$$\mathrm{LL}(T[\vec{F}]) = \mathrm{LEFTLEVEL}(\mathrm{YIELD}(T[\vec{F}])).$$

When our algorithm computes the collection C_{j+1} from C_j, it also maintains the values of ML and LL in the PQ-trees in C_{j+1}. Note that since every PQ-tree in C_1 is a leaf, ML values are not defined, while $\mathrm{LL}(T) = 1$ for each tree $T \in C_1$.

It is easy to see that the ML values satisfy the following two propositions.

PROPOSITION 2.1. *Suppose that u is the least common ancestor of a pair of leaves v and w in a PQ-tree. If u is a P-node, then*

$$\mathrm{MEETLEVEL}(\{v, w\}) = \mathrm{ML}(u).$$

PROPOSITION 2.2. *Suppose that u is the least common ancestor of a pair of leaves v and w in a PQ-tree. Further suppose that u is a Q-node with ordered children $u_1, u_2, \ldots, u_t$ such that $v \in \mathrm{YIELD}(u_p)$ and $w \in \mathrm{YIELD}(u_q)$, where $1 \le p < q \le t$. Then*

$$\mathrm{MEETLEVEL}(\{v, w\}) = \min\{\mathrm{ML}(u_i, u_{i+1}) \mid p \le i < q\}.$$

The following proposition formalizes the notion that, as we follow a path in a PQ-tree from a leaf to the root, the ML values we encounter are nonincreasing.

PROPOSITION 2.3. *Suppose that node u is the parent of a nonleaf node v in a PQ-tree. Define x as follows:*

$$x = \begin{cases} \mathrm{ML}(u) & \text{if } u \text{ is a P-node;} \\ \max\{\mathrm{ML}(v, w) \mid w \text{ is a child of } u \text{ adjacent to } v\} & \text{if } u \text{ is a Q-node.} \end{cases}$$

Define y as follows:

$$y = \begin{cases} \mathrm{ML}(v) & \text{if } v \text{ is a P-node;} \\ \min\{\mathrm{ML}(v_i, v_{i+1}) \mid 1 \le i < t\} & \text{if } u \text{ is a Q-node with ordered} \\ & \text{children } v_1, v_2, \ldots, v_t. \end{cases}$$

Then $x \le y$.

2.4. Operations. We now describe two operations on a PQ-tree that serve as building blocks of the algorithm that constructs C_{j+1} from C_j. In each operation, the PQ-tree is transformed so that the set of permutations represented is restricted to be a potentially smaller set. In the case of the second operation, the yield of the PQ-tree is slightly modified. The LL value of the PQ-tree remains unchanged. The ML values are updated appropriately. We first describe what the operations are, then we describe how they are implemented:

1. ISOLATE(T, x), where T is a PQ-tree and $x \in \mathrm{YIELD}(T)$. This operation transforms T so that $\mathrm{PERM}(T)$ is restricted to permutations in which x is either the first or the last element. If there is no permutation in $\mathrm{PERM}(T)$ that has x as its first or last element, then T becomes ϵ.
2. IDENTIFY(T, x, y, z), where T is a PQ-tree, $x, y \in \mathrm{YIELD}(T)$, $x \ne y$, and z is a new node not in $\mathrm{YIELD}(T)$. Let P be the subset of permutations in $\mathrm{PERM}(T)$ in which x and y appear consecutively. Let P' be obtained from P as follows: If P contains the permutation $a, \ldots, b, x, y, c, \ldots, d$, then put in P' the permutation $a, \ldots, b, z, c, \ldots, d$, obtained by replacing x, y by z. The operation IDENTIFY(T, x, y, z) transforms T so that $\mathrm{PERM}(T) = P'$. Note that P' may be empty, in which case $T = \epsilon$.

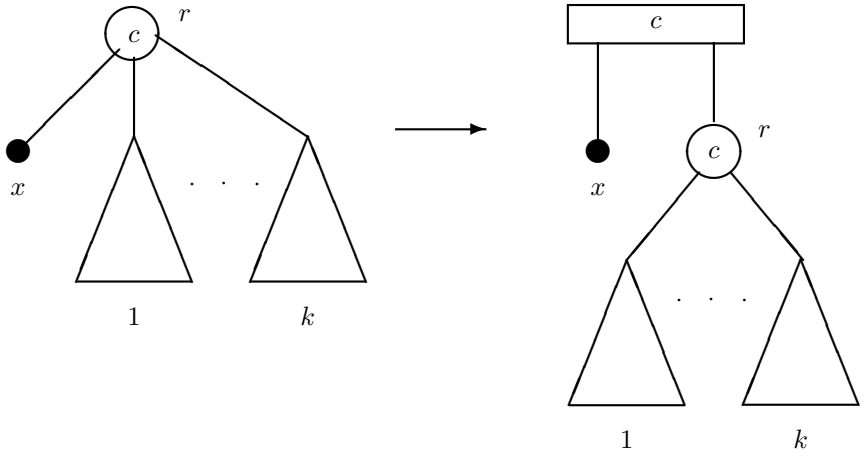


FIG. 2.4. *The transformation of T in the first case of* ISOLATE(T, x).

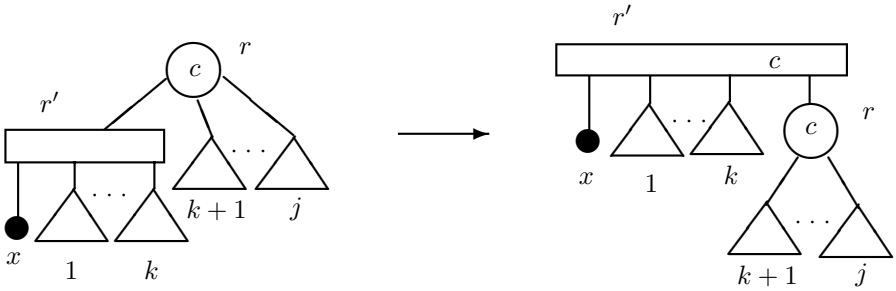


FIG. 2.5. *The transformation of T in the second case of* ISOLATE(T, x) *when r is a P-node.*

We first describe an implementation of ISOLATE(T, x). Let r be the root of T. If $x = r$, then ISOLATE(T, x) simply returns T. Otherwise, there are two cases based on whether or not x is a child of r.

1. x is a child of r. If r is a Q-node and x is not its first or last child, then there are no permutations in PERM(T) with x at the end or at the beginning, so the operation returns ϵ. If r is a Q-node and x is either the first or the last child of r, then T is unchanged. If r is a P-node, then T is transformed as shown in Figure 2.4. Before the operation, the tree consists of the root r, the child x, and subtrees labeled 1 through k. The ML(r) value is c. After the operation, a Q-node has been added as the root of T, x has been moved to be the first child of that Q-node, and the P-node r becomes the second and last child of that Q-node. The ML values are set as indicated.
2. x is not a child of r. Let T' be the subtree rooted at a child of r whose yield contains x. Perform ISOLATE(T', x). If $T' = \epsilon$, then ISOLATE(T, x) results in $T = \epsilon$. Otherwise, the root r' of T' is a Q-node with x as either its first or its last child. Without loss of generality, we may assume that x is the first child of r' and that the remaining subtrees attached to r' are labeled 1 through k. If r is a P-node, perform the transformation on T shown in Figure 2.5. The subtrees attached to r are T' together with subtrees labeled $k + 1$ through j. The ML(r) value is c. In this transformation, r' is rotated up to

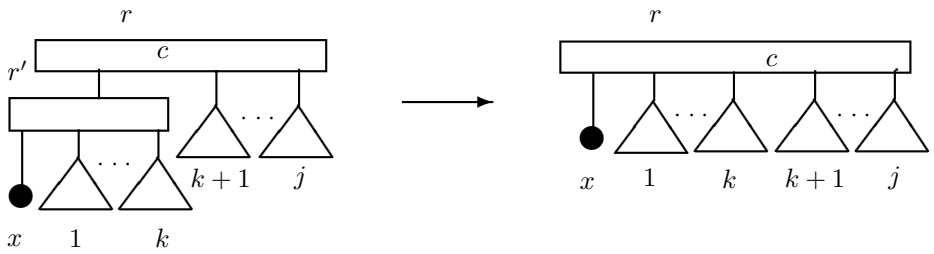


FIG. 2.6. *The transformation of T in the second case of* ISOLATE(T, x) *when r is a Q-node.*

be the root of T, while r becomes the last child of r'. The ML values are set as indicated. If r is a Q-node and r' is not the first or the last child of r, then set $T = \epsilon$. Otherwise, we may assume, without loss of generality, that r' is the first child of r. Perform the transformation on T shown in Figure 2.6. The subtrees attached to r are T' together with subtrees labeled $k + 1$ through j. In this transformation, r' is rotated up to be the root of T, while the subtrees $k + 1$ through j become the remaining subtrees attached to r'. The ML values are set as indicated.

The running time of ISOLATE(T, x) is proportional to the depth of x in T.

The operation IDENTIFY(T, x, y, z) can be implemented in the following four steps.

Step 1. Locate r, the node in T that is the least common ancestor of x and y.

Step 2. Let T_1 and T_2 be the subtrees of T rooted at children of r such that $x \in$ YIELD(T_1) and $y \in$ YIELD(T_2). Perform ISOLATE(T_1, x) and ISOLATE(T_2, y). If either $T_1 = \epsilon$ or $T_2 = \epsilon$, then set $T = \epsilon$. Otherwise, let r_1 be the root of T_1 and r_2 the root of T_2. Then r_1 is a Q-node with x being its first or last child, while r_2 is a Q-node with y being its first or last child.

Step 3. This step brings x and y together, if possible. The details depend on whether r is a P-node or a Q-node.

- *r is a P-node.* T is transformed as shown in Figure 2.7. The two Q-nodes r_1 and r_2 are merged so as to make x and y adjacent. The ML values c_1, c_2, and c_3 are repositioned as shown.
- *r is a Q-node.* If the subtrees T_1 and T_2 are not adjacent children of r, then set $T = \epsilon$. Otherwise, T is transformed as shown in Figure 2.8. The three Q-nodes r, r_1, and r_2 are merged so as to make x and y adjacent. The ML values c_1, c_2, and c_3 are repositioned as shown.

Step 4. Leaf z replaces leaves x and y.

The running time of IDENTIFY(T, x, y, z) is proportional to the sum of the depths of x and y in T.

In the transformations described for ISOLATE and IDENTIFY, we have ignored several special cases arising from the possibility that a transformation might lead to the birth of a P-node with two children. Instead of dealing with these cases separately, we simply note that whenever this happens, the P-node is replaced by a Q-node.

2.5. Recognizing leveled-planar dags. We are now ready to present our algorithm for determining whether the connected, leveled dag $\vec{G} = (V_1, \ldots, V_m; \vec{E})$ is a leveled-planar dag. In parallel, we develop the proof of correctness for the algorithm. The overall structure of the algorithm is an iteration that builds the collection C_{j+1} from the collection C_j for each j, $1 \leq j < m$. Hence, the proof of correctness is by

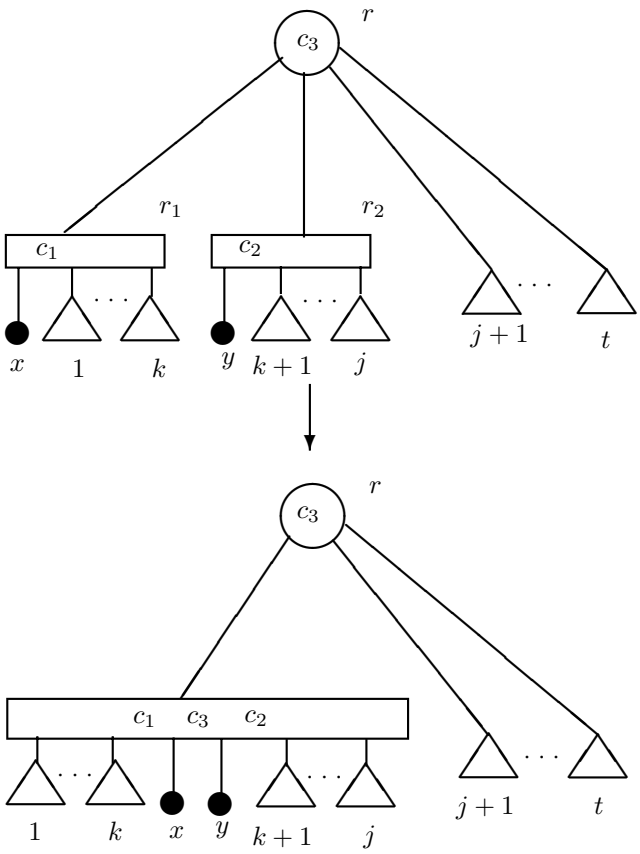


FIG. 2.7. *The transformation of T in* IDENTIFY(T, x, y, z) *when r is a P-node.*

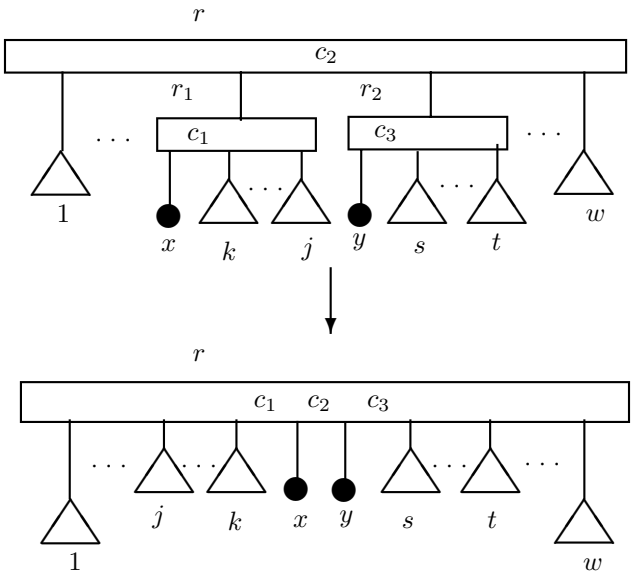


FIG. 2.8. *The transformation of T in* IDENTIFY(T, x, y, z) *when r is a Q-node.*

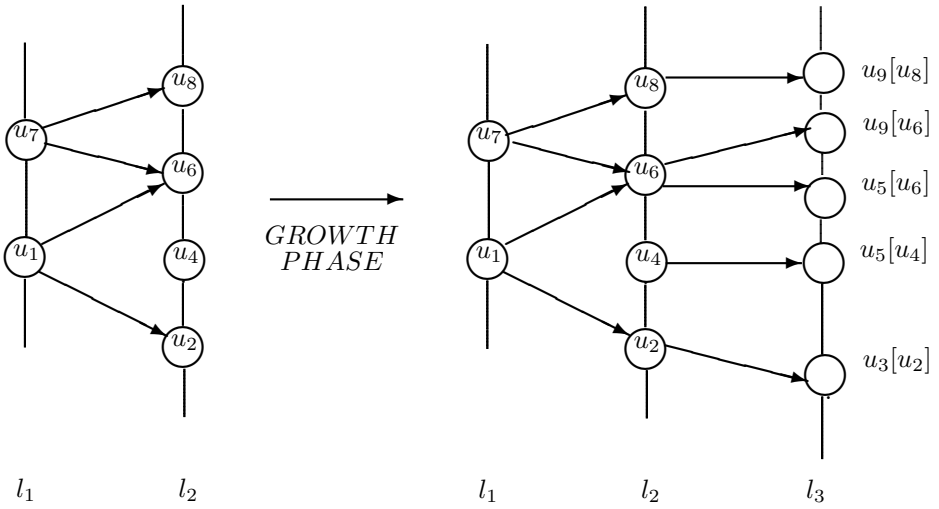


FIG. 2.9. *Illustration of the growth phase.*

induction on j with the collection invariant being the inductive hypothesis. The base case is $j = 1$. In this case, $\vec{G}_1 = (V_1, \emptyset)$ and C_1 contains, for each $v \in V_1$, one PQ-tree that contains only the node v.

When $1 \leq j < m$, the iterative step extends $\vec{G}_j$ to $\vec{G}_{j+1}$, while at the same time extending C_j to C_{j+1}. Let $\vec{E}_j$ be the set of *level-j arcs* from nodes in V_j to nodes in V_{j+1}. To extend $\vec{G}_j$ to $\vec{G}_{j+1}$, we must add the arcs in $\vec{E}_j$ and the nodes in V_{j+1} to $\vec{G}_j$. In describing the iterative step, it will be helpful to imagine that the arcs in $\vec{E}_j$ are added first and that the nodes in V_{j+1} are subsequently identified in a series of substeps. In particular, the iterative step can be thought of as working in four distinct phases: (i) GROWTH PHASE, (ii) FIRST MERGE PHASE, (iii) SECOND MERGE PHASE, and (iv) CLEANUP PHASE. We now describe each of the phases separately.

GROWTH PHASE. For a node v, let $\text{IN}(v)$ be the set of in-neighbors of v, and let $\text{OUT}(v)$ be the set of out-neighbors of v. For each $v \in V_{j+1}$ and for each $u \in \text{IN}(v)$, let $v[u]$ be a new node called a *copy* of v. In the growth phase, to $\vec{G}_j$, we add arcs $(u, v[u])$, for every node $v \in V_{j+1}$ and every $u \in \text{IN}(v)$. Thus, every connected component in $\vec{G}_j$ grows by a level and in this manner $\vec{G}_j$ is transformed into a leveled-planar dag $\vec{H}$ that has $j + 1$ levels. Note that $\vec{H}$ approximates $\vec{G}_{j+1}$ in the sense that every arc in $\vec{G}_{j+1}$ is represented in $\vec{H}$. An illustration of the growth phase can be found in Figure 2.9, where the first two levels of the dag of Figure 2.1 are extended to the third level. Note the two copies of the node u_9, namely, $u_9[u_8]$ and $u_9[u_6]$, and the two copies of the node u_5, namely, $u_5[u_6]$ and $u_5[u_4]$, that are created in this phase.

FIRST MERGE PHASE. In the two merge phases, all nodes of the form $v[u]$ that are created in the growth phase are identified to obtain the node v. In the first merge phase nodes belonging to the same connected component are identified, while in the second merge phase nodes belonging to distinct connected components are identified, as a result merging two connected components into one. If $v \in V_{j+1}$ and $X \subseteq \text{IN}(v)$, then let $v[X]$ be a new node that we think of as the result of identifying

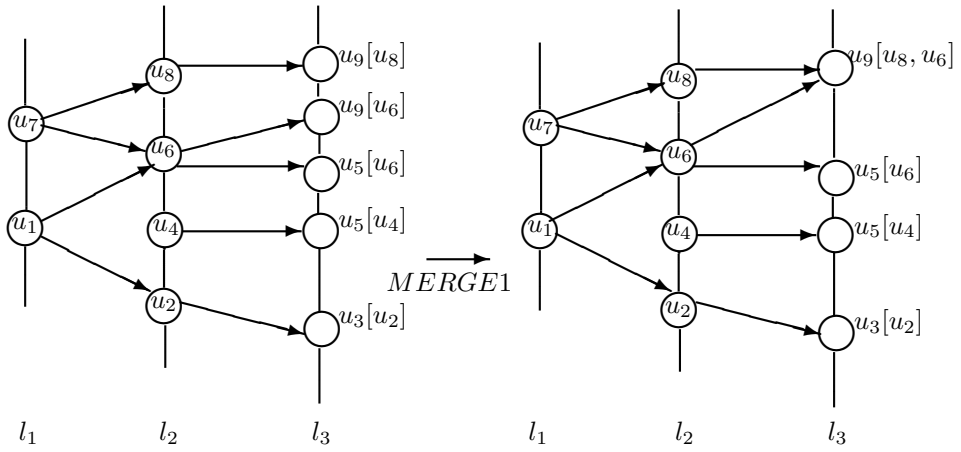


FIG. 2.10. *Illustration of the first merge phase.*

all nodes of the form $v[u]$, $u \in X$. (We think of the X in this notation as being an unordered list of the elements of X. Then we can write $v[X, Y]$ for $v[X \cup Y]$ and even $v[u, X]$ for $v[\{u\} \cup X]$.)

Consider any connected component $\vec{F}$ in $\vec{H}$. Recall that $\vec{F}$ was created in the growth phase by extending a connected component in $\vec{G}_j$ to $j+1$ levels. Also recall that each level-$(j+1)$ node in $\vec{F}$ is of the form $v[u]$, where $v \in V_{j+1}$ and $u \in V_j$. For each $v \in V_{j+1}$, let S_v be the set of all nodes u in $\vec{F}$ such that $v[u]$ is a node in $\vec{F}$. In the first merge phase all copies, $v[u]$, of v are identified into one node $v[S_v]$. This identification is done pairwise and in a certain order. For any permutation $\pi \in \text{PERM}(T[\vec{F}])$ of the nodes in $\text{YIELD}(T[\vec{F}])$, let

$$v[u_1], v[u_2], \ldots, v[u_t]$$

be the subsequence of π containing all copies of v in $\vec{F}$. The identification of the copies of v is done in the order prescribed by the above sequence. More precisely, for each i, $1 \leq i < t$, the nodes $v[u_1, u_2, \ldots, u_i]$ and $v[u_{i+1}]$ are identified into a single node $v[u_1, u_2, \ldots, u_{i+1}]$. The reason for respecting the above order is motivated by the following example. Suppose that $v[u_1]$, $v[u_2]$, and $v[u_3]$ are the only copies of v in $\vec{F}$. Further suppose that in every directed leveled-planar embedding of $\vec{F}$, $v[u_2]$ occurs between $v[u_1]$ and $v[u_3]$ on line ℓ_{j+1}. This implies that even though it may be possible to identify the nodes $v[u_1]$, $v[u_2]$, and $v[u_3]$ into a single node $v[u_1, u_2, u_3]$, it is not possible to do this identification pairwise starting with the pair $v[u_1]$ and $v[u_3]$ because $v[u_2]$ is an obstacle. The solution to this problem is to respect an order on the nodes $v[u_1], v[u_2], \ldots, v[u_t]$ that is a subsequence of some witness to a directed leveled-planar embedding of $\vec{F}$.

At the end of this phase, each connected component in $\vec{H}$ contains at most one copy of any node $v \in V_{j+1}$. The first merge phase is illustrated in Figure 2.10.

SECOND MERGE PHASE. In this phase duplicate copies of any level-$(j+1)$ node that occur in different connected components are identified. Thus, this phase eliminates any remaining duplicate copies of level-$(j+1)$ nodes by completing the identification that was started in the first merge phase. Suppose that connected components $\vec{F}_1$ and $\vec{F}_2$ of $\vec{H}$ share copies of a level-$(j+1)$ node. Note that, because

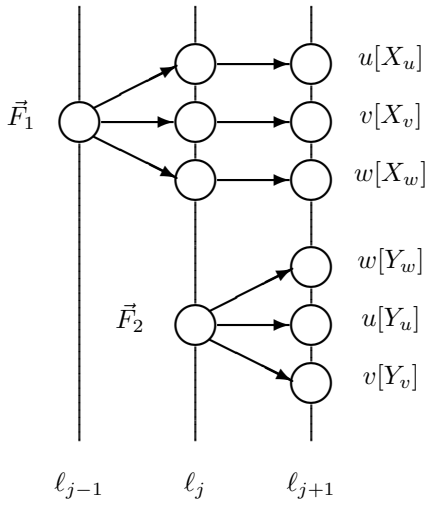


FIG. 2.11. *If $\vec{F}_1$ and $\vec{F}_2$ share copies of three level-$(j+1)$ nodes, then their merger cannot be leveled-planar.*

of the first merge phase, $\vec{F}_1$ and $\vec{F}_2$ each contain exactly one copy of any level-$(j+1)$ node. However, it is possible that they share copies of more than one level-$(j+1)$ node. Let U be the set of all $v \in V_{j+1}$ such that there is a copy $v[X]$ of v in $\vec{F}_1$ and a copy $v[Y]$ of v in $\vec{F}_2$. Let U be the set of all $v \in V_{j+1}$ such that there is a copy $v[X]$ of v in $\vec{F}_1$ and a copy $v[Y]$ of v in $\vec{F}_2$. If $|U| \geq 3$, then it is easy to see that $\vec{F}_3$ cannot be a leveled-planar dag. Figure 2.11 shows an example illustrating this fact for $|U| = 3$. Suppose $U = \{v_1, v_2\}$, where $v_1[X_1]$ (respectively, $v_2[X_2]$) is a copy of v_1 (respectively, v_2) in $\vec{F}_1$ and $v_1[Y_1]$ (respectively, $v_2[Y_2]$) is a copy of v_1 (respectively, v_2) in $\vec{F}_2$. Then, in the second merge phase, the connected components $\vec{F}_1$ and $\vec{F}_2$ are replaced by the connected component $\vec{F}$ that is obtained by identifying $v_1[X_1]$ and $v_1[Y_1]$ into a single node $v_1[X_1, Y_1]$. Now $\vec{F}$ contains two copies of v_2, namely, $v_2[X_2]$ and $v_2[Y_2]$. These are identified into a single node $v_2[X_2, Y_2]$. The second merge phase is illustrated in Figure 2.12.

CLEANUP PHASE. The dag $\vec{H}$ passes through the three earlier phases before it reaches the cleanup phase. The earlier phases ensure that for every node $v \in V_{j+1}$, there is exactly one copy, $v[\mathrm{IN}(v)]$, in $\vec{H}$. In this phase, each level-$(j+1)$ node in $\vec{H}$ with label $v[\mathrm{IN}(v)]$ is relabeled v so as to match its name in $\vec{G}_{j+1}$. For example, the dag obtained after the second merge phase in Figure 2.12 contains nodes $u_9[u_8, u_6]$, $u_5[u_4, u_6]$, and $u_3[u_2]$ which are relabeled u_9, u_5, and u_3 in the CLEANUP PHASE. More importantly, all level-$(j+1)$ sources in $\vec{G}$ are added to $\vec{H}$. This completes the transformation of $\vec{H}$ into $\vec{G}_{j+1}$.

Figure 2.13 shows the algorithm that transforms G_j into G_{j+1} using the four phases described above. In this algorithm, $\vec{H}$ is initialized to $\vec{G}_j$ and then transformed into $\vec{G}_{j+1}$ in four phases.

Having described the sequence of operations that transforms $\vec{G}_j$ into $\vec{G}_{j+1}$, we now describe the parallel sequence of operations that transforms C_j into C_{j+1}. We emphasize again that it is the transformations on the collections that are actually performed; the algorithm in Figure 2.13 merely gives a framework within which to

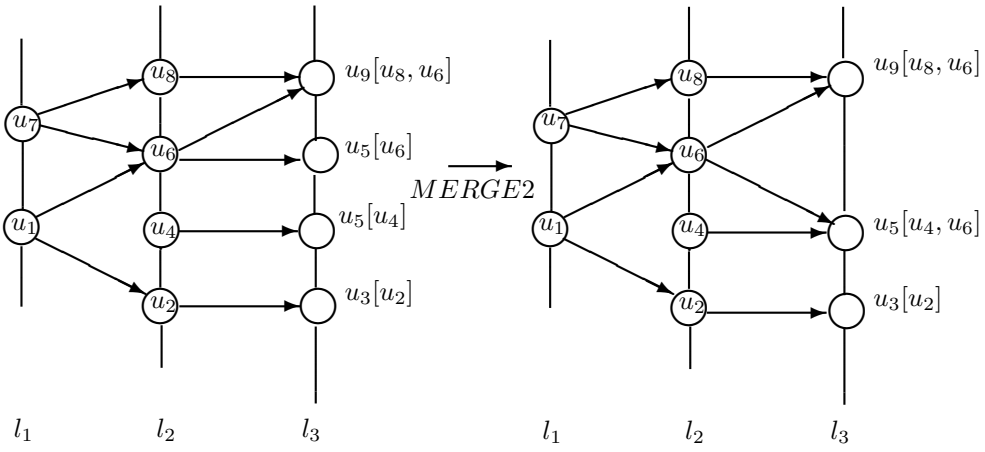


FIG. 2.12. *Illustration of the second merge phase.*

/* Algorithm for transforming G_j into G_{j+1} */

$\vec{H} \leftarrow \vec{G}_j$

/* GROWTH PHASE */
for $\vec{F} \in \text{COMP}(\vec{H})$ **do**
  for $u \in \text{YIELD}(T[\vec{F}])$ **do**
    for $v \in \text{OUT}(u)$ **do**
      (1) Add arc $(u, v[u])$ to $\vec{F}$

/* FIRST MERGE PHASE */
for $v \in V_{j+1}$ **do**
  for $\vec{F} \in \text{COMP}(\vec{H})$ **do**
    Let $v[u_1], v[u_2], \ldots, v[u_t]$ be a subsequence of some $\pi \in \text{PERM}(T[\vec{F}])$
    for $i = 1$ **to** $t - 1$ **do**
      (2) Identify $v[u_1, u_2, \ldots, u_i]$ and $v[u_{i+1}]$ into $v[u_1, u_2, \ldots, u_{i+1}]$

/* SECOND MERGE PHASE */
for $\vec{F}_1, \vec{F}_2 \in \text{COMP}(\vec{G})$ **do**
  Let $U = \{v \mid \vec{F}_1$ contains $v[X]$ and $\vec{F}_2$ contains $v[Y]\}$
    for $v \in U$ **do**
      (3) Identify $v[X]$ and $v[Y]$ into a single node $v[X, Y]$

/* CLEANUP PHASE */
Relabel each node $v[X]$ in $\vec{H}$ as v
Add all the level-$(j+1)$ sources in $\vec{G}$ to $\vec{H}$
$\vec{G}_{j+1} \leftarrow \vec{H}$

FIG. 2.13. *Transforming $\vec{G}_j$ into $\vec{G}_{j+1}$.*

explain the rationale for the transformations. In particular, for each of the statements marked (1), (2), and (3) in the algorithm in Figure 2.13, we perform corresponding operations on collections. So we now describe the transformations applied to collections during the four phases of the algorithm.

GROWTH PHASE. We need to describe the effect on the PQ-tree $T[\vec{F}]$ of adding the arcs of the form $(u, v[u])$ to the dag $\vec{F}$. Let $\vec{F}'$ be the dag that results from adding all these arcs to $\vec{F}$. All the outgoing arcs from u in $\vec{F}'$ must appear together in such an embedding. Conversely, as long as they appear together, the outgoing arcs from u in $\vec{F}'$ can appear in any order. We mirror this constraint in the PQ-tree $T[\vec{F}]$ as follows. Start with $T[\vec{F}'] = T[\vec{F}]$. Suppose $u \in \text{YIELD}(T[\vec{F}])$. If u has no out-neighbors, then delete the leaf u from $T[\vec{F}']$. If u has only one out-neighbor v, then replace the leaf u in $T[\vec{F}']$ with the leaf $v[u]$. If u has two out-neighbors v_1 and v_2, then replace the leaf u in $T[\vec{F}']$ with a Q-node having two children $v_1[u]$ and $v_2[u]$ that are leaves. If u has more than two out-neighbors $v_1, v_2, \ldots, v_t$, $t > 2$, then replace the leaf u in $T[\vec{F}']$ with a P-node having children that are leaves $v_1[u], v_2[u], \ldots, v_t[u]$. From the preceding discussion, it should be clear that the resulting $T[\vec{F}']$ mirrors $\vec{F}'$.

FIRST MERGE PHASE. In this phase, each dag $\vec{F} \in \text{COMP}(\vec{H})$ is transformed by repeatedly identifying level-$(j+1)$ nodes $v[X]$ and $v[Y]$ into a single node $v[X, Y]$. Identifying $v[X]$ and $v[Y]$ into a single node $v[X, Y]$ first forces the nodes $v[X]$ and $v[Y]$ to appear consecutively on line ℓ_{j+1} and then contracts these two nodes into a single node $v[X, Y]$. Thus, we are interested only in those permutations in $\text{PERM}(T[\vec{F}])$ in which $v[X]$ and $v[Y]$ appear together. In all such permutations, $v[X]$ and $v[Y]$ need to be replaced by $v[X, Y]$. The PQ-tree operation $\text{IDENTIFY}(T[\vec{F}], v[X], v[Y], v[X, Y])$ does precisely this. Hence repeated IDENTIFY operations on the PQ-trees in the collection result in a collection that mirrors the dag $\vec{H}$ obtained after this phase. Note that the order in which these IDENTIFY operations are applied to the collection has to be identical to the order in which pairs of nodes in $\vec{H}$ are identified.

SECOND MERGE PHASE. For this phase, we must describe how identifying $v[X]$ and $v[Y]$ belonging to different connected components $\vec{F}_1$ and $\vec{F}_2$ in $\vec{H}$ is reflected by a corresponding operation on the current collection. This operation is the most complicated PQ-tree operation and requires some initial intuition before we dive into the details. Let $T_1 = T[\vec{F}_1]$ and $T_2 = T[\vec{F}_2]$. One result of identifying $v[X]$ and $v[Y]$ is that the two connected components $\vec{F}_1$ and $\vec{F}_2$ are merged into one component, call it $\vec{F}_3$, hence decreasing the number of connected components of $\vec{H}$ by one. Similarly, the operation on the current collection results in a merger of T_1 and T_2 into a new PQ-tree $T_3 = T[\vec{F}_3]$.

The merging of T_1 and T_2 should mirror the merging of the directed leveled-planar embeddings of $\vec{F}_1$ and $\vec{F}_2$. When two such embeddings are joined at $v[X, Y]$, then one embedding must nestle inside the other embedding (unless both $v[X]$ and $v[Y]$ were at the ends of the embeddings of $\vec{F}_1$ and $\vec{F}_2$). For there to be room for this nestling, we must consider the structure of the dags at earlier levels. The values ML and LL stored at the nodes in T_1 and T_2 allow us to do so. We now describe the operation for this case in detail.

Without loss of generality, we may assume that $\text{LL}(T_1) \leq \text{LL}(T_2)$. The trees T_1 and T_2 are merged in two steps. In the first step, the PQ-tree T_2 is attached to T_1 at an appropriate location. We will call the resulting tree T_3'. The tree T_3' contains the leaves $v[X]$ and $v[Y]$. In the second step, these two leaves in T_3' are identified into

one leaf $v[X, Y]$ using the operation IDENTIFY. The resulting tree is called T_3. The two steps are discussed in detail below.

Step 1. *Attaching* T_2 *to* T_1. Start with the leaf $v[X]$ in T_1 and proceed upward in T_1 until a node r' and its parent r are encountered satisfying one of these five conditions:

1. r is a P-node with $\mathrm{ML}(r) < \mathrm{LL}(T_2)$. T_3' is obtained by attaching T_2 as a child of r in T_1.
2. r is a Q-node with ordered children $r_1, r_2, \ldots, r_t$, $r' = r_1$, and $\mathrm{ML}(r_1, r_2) < \mathrm{LL}(T_2)$. T_3' is obtained by replacing r_1 in T_1 with a Q-node q having two children, r_1 and the root of T_2. The case where $r' = r_t$ and $\mathrm{ML}(r_{t-1}, r_t) < \mathrm{LL}(T_2)$ is symmetric.
3. r is a Q-node with ordered children $r_1, r_2, \ldots, r_t$, $r' = r_i$, for some i, $1 < i < t$, and both $\mathrm{ML}(r_{i-1}, r_i) < \mathrm{LL}(T_2)$ and $\mathrm{ML}(r_i, r_{i+1}) < \mathrm{LL}(T_2)$. T_3' is obtained by replacing r_i in T_1 with a Q-node q having two children, r_i and the root of T_2.
4. r is a Q-node with children $r_1, r_2, \ldots, r_t$, $r' = r_i$, $1 < i < t$, and
$$\mathrm{ML}(r_{i-1}, r_i) < \mathrm{LL}(T_2) \leq \mathrm{ML}(r_i, r_{i+1}).$$
T_3' is obtained by attaching T_2 as a child of r between r_{i-1} and r_i. The case where
$$\mathrm{ML}(r_i, r_{i+1}) < \mathrm{LL}(T_2) \leq \mathrm{ML}(r_{i-1}, r_i)$$
is symmetric.
5. r' is the root of T_1. In this case, construct T_3' by making its root a Q-node q with two children, r' and the root of T_2.

Step 2. *Identifying* $v[X]$ *and* $v[Y]$ *into a single node* $v[X, Y]$. T_3 is obtained by performing $\mathrm{IDENTIFY}(T_3', v[X], v[Y], v[X, Y])$.

Steps 1 and 2 also update ML and LL values; this update is a straightforward extension of the updates in the ISOLATE and IDENTIFY operations. The following lemma establishes the correctness of Steps 1 and 2.

LEMMA 2.4. T_3 *mirrors* $\vec{F}_3$.

To prove Lemma 2.4 we need two observations.

OBSERVATION 1. *Let* $\pi_1 \in \mathrm{PERM}(T_1)$ *be a permutation in which the nodes in* $\mathrm{YIELD}(r')$ *are immediately followed by a node* x *such that* $\mathrm{MEETLEVEL}(\mathrm{YIELD}(r') \cup \{x\}) < \mathrm{LL}(T_2)$. *For any* $\pi_2 \in \mathrm{PERM}(T_2)$, *there is a permutation* $\pi \in \mathrm{PERM}(T_3')$ *that is consistent with* π_1 *and* π_2 *and in which the nodes in* $\mathrm{YIELD}(T_2)$ *occur immediately after* $\mathrm{YIELD}(r')$ *and just before* x.

Proof of Observation 1. Permute the leaves in T_1 according to π_1 and the leaves in T_2 according to π_2. For each of the four ways of constructing T_3' from T_1 and T_2, explained in Step 2 above, we show that the leaves of T_3' can be permuted to obtain the desired permutation. Let s denote the root of T_2.

1. T_2 *is attached as a child of a P-node* r. In this case, permute the children of r so that s immediately follows r' and the remaining children stay in the same relative order. This places the nodes in $\mathrm{YIELD}(T_2)$ immediately after the nodes in $\mathrm{YIELD}(r')$, without changing the relative order of the nodes in $\mathrm{YIELD}(T_1)$ or $\mathrm{YIELD}(T_2)$.
2. T_2 *is attached as a child of a Q-node* q *that has one other child* r' *and parent* r. In this case, reverse the order of the children of q, if necessary, so as to have s follow r'. This places the nodes in $\mathrm{YIELD}(T_2)$ immediately after the nodes

in YIELD(r'), without changing the relative order of the nodes in YIELD(T_1) or YIELD(T_2).

3. *T_2 is attached between r_{j-1} and $r' = r_j$, where $1 < j < t$, as a child of a Q-node r.* Since x immediately follows nodes in YIELD(r') in π_1, we have $x \in$ YIELD(r_{j-1}) or $x \in$ YIELD(r_{j+1}). We now show that $x \notin$ YIELD(r_{j+1}). We know that ML(r_j, r_{j+1}) $\geq$ LL(T_2) and this implies that for all $w \in$ YIELD(r_{j+1}), we have MEETLEVEL(YIELD(r_j) $\cup \{w\}$) $\geq$ LL(T_2). Since we assumed that MEETLEVEL(YIELD(r') $\cup \{x\}$) $<$ LL(T_2), we have $x \notin$ YIELD(r_{j+1}). This implies that $x \in$ YIELD(r_{j-1}) and since x follows nodes in YIELD(r') in π_1, the children of r in T_1 are permuted so that r_{j-1} occurs after r_j. This means that s occurs immediately after r_j in T_3. As a result the nodes in YIELD(T_2) immediately follow after the nodes in YIELD(r'), without changing the relative order of the nodes in YIELD(T_1) or YIELD(T_2).
4. *T_2 is attached to a Q-node q that has two children, r' and s, and has no parent.* We show that this case is not possible. We know that if T_2 is being attached to T_1 as described above, then MEETLEVEL(YIELD(T_2)) $\geq$ LL(T_1). This implies that

$$\text{MEETLEVEL}(\text{YIELD}(r') \cup \{x\}) \geq \text{LL}(T_1),$$

a contradiction. □

OBSERVATION 2. *Let $\pi \in$ PERM(T_1) be a permutation in which the nodes in* YIELD(r') *occur at the end. Let $\pi_2 \in$ PERM(T_2) be an arbitrary permutation of* YIELD(T_2). *Then, there is a permutation $\pi \in$ PERM(T_3') that is consistent with π_1 and π_2 and in which the nodes in* YIELD(T_2) *occur after* YIELD(r').

Proof of Observation 2. Permute the leaves in T_1 according to π_1 and the leaves in T_2 according to π_2. For each of the four ways of constructing T_3 from T_1 and T_2, explained in Step 2 above, we show that the leaves of T_3 can be permuted to obtain the desired permutation. Let s denote the root of T_2.

1. *T_2 is attached as a child of a P-node r or T_2 is attached as a child of a Q-node q that has one other child r' and parent r.* As explained in the proof of Observation 1, in either of these cases the leaves of T_3 can be permuted so as to have nodes in YIELD(T_2) immediately follow nodes in YIELD(r') without changing the relative order of the nodes in YIELD(T_1) and in YIELD(T_2).
2. *T_2 is attached between r_{j-1} and $r' = r_j$, where $1 < j < t$, as a child of a Q-node r.* This case is not possible since nodes in YIELD(r') cannot appear at the end of π_1 in this case.
3. *T_2 is attached to a Q-node q that has two children r' and s and has no parent.* Reverse the children of q, if necessary, so as to have s follow r'. □

Proof of Lemma 2.4. To show that T_3 mirrors $\vec{F}_3$, we need to show that PERM(T_3) is the set of all witnesses to directed leveled-planar embeddings of $\vec{F}_3$. We first show that any $\pi \in$ PERM(T_3) is a witness to some directed leveled-planar embedding $\mathcal{E}_3$ of $\vec{F}_3$. We then show that any permutation π that is a witness to some directed leveled-planar embedding $\mathcal{E}_3$ of $\vec{F}_3$ is in PERM(T_3).

If $\pi \in$ PERM(T_3), then π is a witness to some directed leveled-planar embedding $\mathcal{E}_3$ of $\vec{F}_3$. Recall that T_3 is the result of IDENTIFY($T_3', v[X], v[Y], v[X, Y]$) in Step 2. Since $\pi \in$ PERM(T_3), by the definition of the IDENTIFY operation, there is a permutation $\pi' \in$ PERM(T_3') in which $v[X]$ and $v[Y]$ occur together and replacing them by $v[X, Y]$ yields π. Clearly, in π' the elements in YIELD(T_2) occur contigu-

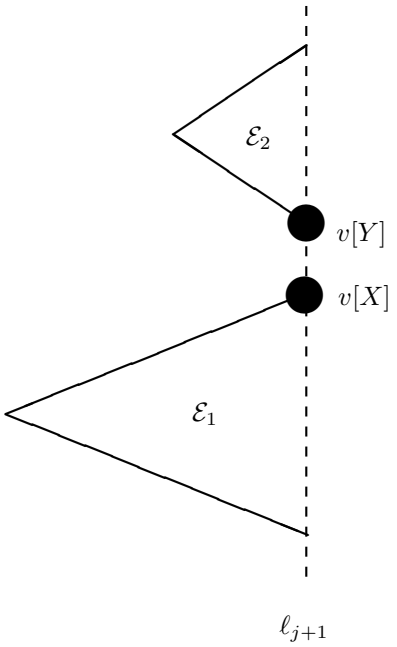


FIG. 2.14. *Construction of $\mathcal{E}_3$ from $\mathcal{E}_1$ and $\mathcal{E}_2$.*

ously and the elements in YIELD(r') occur contiguously. So without loss of generality, assume that in π' the nodes in YIELD(r') are immediately followed by the nodes in YIELD(T_2). Thus, among the nodes in YIELD(r'), $v[X]$ occurs last and among the nodes in YIELD(T_2), $v[Y]$ occurs first. Therefore π' can be written as $\pi_1\pi_2\pi_3$, where $\pi_2 \in \text{PERM}(T_2)$ and $\pi_1\pi_3 \in \text{PERM}(T_1)$. By the induction hypothesis, π_2 (respectively, $\pi_1\pi_3$) is a witness to a directed leveled-planar embedding $\mathcal{E}_2$ (respectively, $\mathcal{E}_1$) of $\vec{F}_2$ (respectively, $\vec{F}_1$). Two possible cases arise based on whether or not π_3 is empty:

(a) π_3 is empty. A planar embedding $\mathcal{E}_3$ of $\vec{F}_3$ can be constructed by placing $\mathcal{E}_2$ on "top" of $\mathcal{E}_1$ as shown in Figure 2.14 and then merging nodes $v[X]$ and $v[Y]$ into a single node $v[X, Y]$. Clearly, the level-$(j+1)$ nodes in $\vec{F}_3$ appear in bottom-to-top order on line ℓ_{j+1} according to π. Hence, π is a witness to the directed leveled-planar embedding $\mathcal{E}_3$ of $\vec{F}_3$.

(b) π_3 is nonempty. Suppose that $x \in \text{YIELD}(T_1)$ is the first element in π_3. Since $x \notin \text{YIELD}(r')$, the least common ancestor y of $v[X]$ and x is an ancestor (not necessarily proper) of r in T_1. Because of the way T_2 is attached to T_1, and due to Propositions 2.1, 2.2, and 2.3, we have

$$\text{MEETLEVEL}(\{v[X], x\}) < \text{LL}(T_2).$$

Hence, the embedding $\mathcal{E}_2$ of $\vec{F}_2$ can be "nested inside" the embedding $\mathcal{E}_1$ of $\vec{F}_1$ as shown in Figure 2.15. By merging nodes $v[X]$ and $v[Y]$ in this embedding, we get the leveled-planar embedding $\mathcal{E}_3$ of $\vec{F}_3$ in which the level-$(j+1)$ nodes appear in bottom-to-top order on ℓ_{j+1} according to π. Hence, π is a witness to the directed leveled-planar embedding $\mathcal{E}_3$ of $\vec{F}_3$.

If π is a witness to some directed leveled-planar embedding $\mathcal{E}_3$ of $\vec{F}_3$, then $\pi \in \text{PERM}(T_3)$. The level-$(j+1)$ nodes in $\vec{F}_3$ can be partitioned into three sets: S_1, the

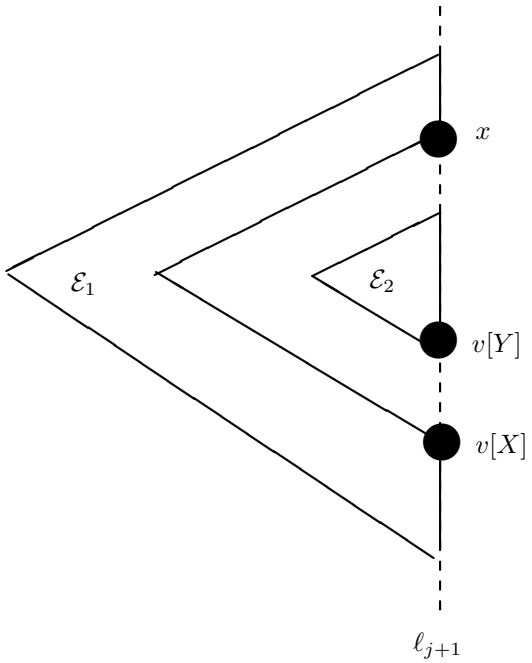


FIG. 2.15. *Construction of $\mathcal{E}_3$ from $\mathcal{E}_1$ and $\mathcal{E}_2$.*

set of all level-$(j+1)$ nodes in $\vec{F}_1$ except $v[X]$; S_2, the set of all level-$(j+1)$ nodes in $\vec{F}_2$ except $v[Y]$; and $\{v[X,Y]\}$. It is easy to see that in π, the nodes in S_2 appear consecutively, either immediately followed by or immediately preceded by $v[X,Y]$. Without loss of generality, we assume that in π, $v[X,Y]$ is immediately followed by nodes in S_2. Let $\vec{F}_3'$ be the dag that contains the two connected components $\vec{F}_1$ and $\vec{F}_2$. Replacing the node $v[X,Y]$ in π by node $v[X]$ followed by node $v[Y]$, we have a permutation π' that witnesses a directed leveled-planar embedding $\mathcal{E}_3'$ of $\vec{F}_3'$. Denote the subembedding of $\vec{F}_1$ in $\mathcal{E}_3'$ by $\mathcal{E}_1$ and the subembedding of $\vec{F}_2$ in $\mathcal{E}_3'$ by $\mathcal{E}_2$. Clearly, π' can be written as $\pi_1\pi_2\pi_3$, where π_1 ends with $v[X]$, π_2 begins with $v[Y]$ and is a witness to $\mathcal{E}_2$, and $\pi_1\pi_3$ is a witness to $\mathcal{E}_1$. By the induction hypothesis, $\pi_1\pi_3 \in \text{PERM}(T_1)$ and $\pi_2 \in \text{PERM}(T_2)$. We will now show that $\pi' \in \text{PERM}(T_3')$. This immediately implies that $\pi \in \text{PERM}(T_3)$. There are two cases depending on whether or not π_3 is empty:

(a) π_3 is nonempty. Suppose that the first element in π_3 is x. Clearly, since the nodes in $\text{YIELD}(T_2)$ occur consecutively between nodes $v[X]$ and x and since the embedding $\mathcal{E}_3'$ is leveled-planar, it has to be the case that

$$\text{MEETLEVEL}(\{u,x\}) < \text{LL}(T_2).$$

Let r be the node in T_1 that is a least common ancestor of $v[X]$ and x and let r' be a child of r such that $v[X] \in \text{YIELD}(r')$. Since $\text{MEETLEVEL}(\text{YIELD}(r') \cup \{x\}) \leq \text{MEETLEVEL}(\{v[X],x\})$, from Observation 1, it follows that $\pi' \in \text{PERM}(T_3)$.

(b) π_3 is empty. By Observation 2, it follows that $\pi' \in \text{PERM}(T_3)$. □

CLEANUP PHASE. Like the other phases, the cleanup phase applied to a collection mimics the cleanup phase applied to the corresponding dag. In this phase,

all the leaves of the PQ-trees in C_{j+1} are relabeled from $v[\mathrm{IN}(v)]$ to v. More significantly, any source s of $\vec{G}$ that is in V_{j+1} results in the addition of a PQ-tree $T[s]$ to C_{j+1} that consists of one leaf labeled s.

This completes the description of our algorithm. The following theorem summarizes what has been accomplished.

THEOREM 2.5. *A leveled-planar dag can be recognized in linear time. Moreover, a leveled-planar embedding for a leveled-planar dag can be constructed in linear time.*

Proof. The correctness of our algorithm follows from the proceeding discussion by induction. The linear time complexity of the algorithm follows from an amortized analysis given here. It suffices to show that the time complexity of all the identification operations together is linear. Each arc in the dag G is allocated three credits; since G is planar, the total number of credits is linear. In identifying two nodes belonging to the same connected component, there are two paths in the dag involved in forming a new face; since each arc is in two faces, two credits from each arc in the face pay for this identification. In identifying two nodes that belong to distinct connected components, the work is proportional to the height of the shorter dag; there is always a path in the dag that has a credit on each arc. We conclude that the allocated credits are sufficient and that the time complexity is linear.

Once our algorithm has recognized that a dag $\vec{G}$ is a leveled-planar dag, the actual construction of a leveled-planar embedding for $\vec{G}$ is straightforward. We outline how to construct an embedding for $\vec{G}$ from right to left. Choose any total order $\leq_m$ on V_m that is consistent with C_m. Choose any total order $\leq_{m+1}$ on V_{m-1} that is consistent with C_{m-1} and that, together with $\leq_m$, induces a leveled-planar embedding on the subgraph of $\vec{G}$ induced by $V_{m+1} \cup V_m$. Extend the construction one level at a time until a leveled-planar embedding of $\vec{G}$ results. Details are left to the reader. □

2.6. Recognizing 1-queue dags. The algorithm to recognize 1-queue dags builds on the one to recognize leveled-planar dags by first finding a maximal leveled subgraph, following the previous algorithm to obtain a leveled-planar embedding of that subgraph, and modifying the computation of C_i in the right-to-left sweep to accommodate the nonleveled arcs.

Let $\vec{G}$ be a connected dag. Let $\vec{H}$ be a spanning subgraph of $\vec{G}$. An *unnecessary* sink in $\vec{H}$ is one that is not also a sink in $\vec{G}$.

LEMMA 2.6. *For any connected dag $\vec{G}$, one may construct in linear time a spanning tree of G that has no unnecessary sinks.*

Proof. Give the connected dag $\vec{G}$ as input to the following program:

$T = (\emptyset, \emptyset)$;
WHILE T is not a spanning subgraph of $\vec{G}$ **DO**
  IF there exists an arc $(u, v) \in \vec{G}$ such that u is in T and v is not in T
    THEN $v_0 = v$;
    ELSE $v_0 =$ any source of $\vec{G}$ that is not in T;
  find a path $P = v_0, v_1, \ldots, v_j$ such that none of $v_0, \ldots, v_{j-1}$ is in T and such that either v_j is a sink in $\vec{G}$ or v_j is in T;
  add P to T;

Since $\vec{G}$ is connected, whenever T is not a spanning subgraph of $\vec{G}$, any node in $\vec{G}$ that is not in T must be reachable either by a path starting at a node of T or at a source of $\vec{G}$ that is not in T. Hence the program makes progress during each iteration of the while loop and terminates with T being a spanning subgraph of $\vec{G}$. By the way T is constructed, it is always acyclic. Hence the final T is a spanning tree of $\vec{G}$.

Linear time is easily accomplished by finding the paths P using depth-first search. Finally, every node added to T has at least one outgoing arc in T unless it is a sink of $\vec{G}$. Hence T has no unnecessary sinks. □

Now suppose $\vec{G}$ is a connected 1-queue dag. Use Lemma 2.6 to find a spanning tree T that has no unnecessary sinks. T also provides a leveling of $\vec{G}$. Add any arcs (u, v) such that $lev(v) = lev(u) + 1$ to obtain a maximal leveled subgraph $\vec{H}$ of $\vec{G}$. The following theorem tells us what additional arcs we must contend with.

THEOREM 2.7. *Let $\vec{G}$ be a connected 1-queue dag, and let $\vec{H}$ be a maximal leveled subgraph of $\vec{G}$ having no unnecessary sinks. Consider any arc (x, y) of $\vec{G}$ that is not in $\vec{H}$. Then either $lev(y) = lev(x)$ or $lev(y) = lev(x) + 2$. Furthermore, there cannot exist an arc (p, q) such that $lev(p) = lev(x)$; $lev(q) = lev(y)$; and $x \neq p$.*

Proof. To obtain a contradiction, we assume the existence of such an arc (p, q). Since neither x nor p is a sink in $\vec{G}$, neither is a sink in $\vec{H}$. Let (x, z) and (p, r) be arcs in $\vec{H}$ ($z = r$ is a possibility). Without loss of generality, we may assume that there is a 1-queue layout of $\vec{G}$ with x occurring before p (since $x \neq p$). Now we consider the two cases individually.

Case 1. The nodes of the independent arcs (x, z) and (p, q) must occur in the layout in the order x, p, q, z because x, p, and q are at level $lev(x)$ and z is at level $lev(x) + 1$. Hence these arcs nest and we have a contradiction.

Case 2. The nodes of the independent arcs (x, y) and (p, r) must occur in the layout in the order x, p, r, y because x and p are at level $lev(x)$, r is at level $lev(x) + 1$, and y is at level $lev(x) + 2$. Hence these arcs nest and we have a contradiction. □

The algorithm first computes the maximal leveled subgraph promised in Lemma 2.6. Then it runs the leveled-planar dag algorithm on the subgraph. The algorithm is modified appropriately to take into account the nonleveled arcs described in Theorem 2.7. For example, if (x, y) is an arc with $lev(x) = lev(y) = i$, then we force the tree T in D_i during the right-to-left sweep to place x and y on opposite ends of the level. This can be done by replacing T with a PQ-tree having a Q-node at the root with three children x, $T - \{x, y\}$, and y. The modifications do not add to the time complexity obtained for the previous algorithm. We obtain the desired linear time algorithm to recognize 1-queue dags, establishing the contrast with the NP-completeness result for recognizing 1-queue undirected graphs [11].

3. Recognizing 4-queue dags is NP-complete. In this section and the next, we show that the problem of determining whether a dag has a 4-queue layout and the problem of determining whether a dag has a 6-stack layout are both NP-complete. In fact we prove a stronger result for queue layouts. An arc (u, v) in a dag $\vec{G}$ is called a transitive arc if $\vec{G} - (u, v)$ contains a directed path from u to v. We show that even when a dag belongs to a restricted class of dags, namely, the class of dags without *transitive arcs*, the problem of determining whether the dag has a 4-queue layout is NP-complete. The motivation for restricting the class of dags to those without transitive arcs comes from our study of stack and queue layouts of posets in [8]. A *stack layout* (respectively, *queue layout)* of a poset P is a stack layout (respectively, queue layout) of its Hasse diagram $\vec{H}(P)$. Thus, our NP-completeness result related to queue layouts of dags shows that the problem of determining whether the queuenumber of a poset is 4 is NP-complete. Our results are in the spirit of the result of Yannakakis [13] who showed that the problem of determining whether or not the dimension of a poset is 3 is NP-complete. Define the decision problem, POSETQN, as follows:

POSETQN
INSTANCE: A poset P.
QUESTION: Can P be laid out in four queues?
Define the decision problem, DAGSN as follows:
DAGSN
INSTANCE: A dag $\vec{G} = (V, \vec{E})$.
QUESTION: Can $\vec{G}$ be laid out in six stacks?
This section proves that POSETQN is NP-complete. Section 4 proves that DAGSN is also NP-complete. The reduction in both proofs is from the NP-complete problem 3-SATISFIABILITY (3-SAT), defined below.
3-SATISFIABILITY
INSTANCE: Collection $C = \{c_1, c_2, \ldots, c_m\}$ of clauses on a set $X = \{x_1, x_2, \ldots, x_n\}$ of variables such that $|c_i| = 3$ for all i, $1 \leq i \leq m$.
QUESTION: Is there a truth assignment of X that satisfies all the clauses in C?

The remainder of this section is devoted to the proof of the following theorem.

THEOREM 3.1. POSETQN *is NP-complete.*

Proof. The queuenumber of a fixed layout of an undirected graph $G = (V, E)$ can be determined in $O(|E| \log\log |V|)$ time [11]. POSETQN is in NP because a nondeterministic Turing machine can guess an ordering of the nodes of the Hasse diagram $\vec{H}(P)$, check if the ordering is a topological ordering, and determine the queuenumber of the layout corresponding to that ordering in polynomial time.

POSETQN is shown to be NP-hard by reduction from 3-SAT. Let the collection of clauses $C = \{c_1, c_2, \ldots, c_m\}$ on the set of variables $X = \{x_1, x_2, \ldots, x_n\}$ be an instance of 3-SAT. The Hasse diagram $\vec{H}(P)$ of a poset P is constructed such that $\vec{H}(P)$ has a 4-queue layout if and only if there exists a truth assignment for the variables in X such that all clauses in C are satisfied. Corresponding to each clause c_i, $1 \leq i \leq m$, $\vec{H}(P)$ contains a subgraph called the *truth-setting dag* $\vec{TS}_i$. First, we describe the construction of $\vec{TS}_i$ and then show how the truth-setting dags are connected together to form $\vec{H}(P)$. The truth-setting dag $\vec{TS}_i$ can itself be thought of as consisting of four distinct subgraphs connected together. These are

1. *literal dag* $\vec{X}_i$,
2. *clause dag* $\vec{C}_i$,
3. *small enforcer dag* $\vec{F}_i$, and
4. *big enforcer dag* $\vec{F}'_i$.

We now describe the construction of $\vec{X}_i$, $\vec{C}_i$, $\vec{F}_i$, and $\vec{F}'_i$ and then show how they connect together to form $\vec{TS}_i$. We use $N(\vec{G})$ to denote the set of nodes of a dag $\vec{G}$ and $A(\vec{G})$ to denote its set of arcs.
Literal dag $\vec{X}_i$*:*

$$\begin{aligned} N(\vec{X}_i) = &\{x_{i,j} \mid 0 \leq j \leq n\} \\ &\cup \{\overline{x}_{i,j} \mid 0 \leq j \leq n\} \\ &\cup \{y_{i,j} \mid 1 \leq j \leq n\}, \\ A(\vec{X}_i) = &\{(y_{i,j}, x_{i,j}),\ (y_{i,j}, \overline{x}_{i,j}) \mid 1 \leq j \leq n\} \\ &\cup \{(x_{i,j}, y_{i,j+1}),\ (\overline{x}_{i,j}, y_{i,j+1}) \mid 0 \leq j \leq n-1\}. \end{aligned}$$

Figure 3.1 shows a literal dag $\vec{X}_i$. Note that there are no transitive arcs in $\vec{X}_i$. Any topological ordering of $\vec{X}_i$ contains the nodes $x_{i,j}$ and $\overline{x}_{i,j}$ followed by $y_{i,j+1}$ followed by $x_{i,j+1}$ and $\overline{x}_{i,j+1}$ for each j, $0 \leq j < n$. But, for each j, there is a choice in the

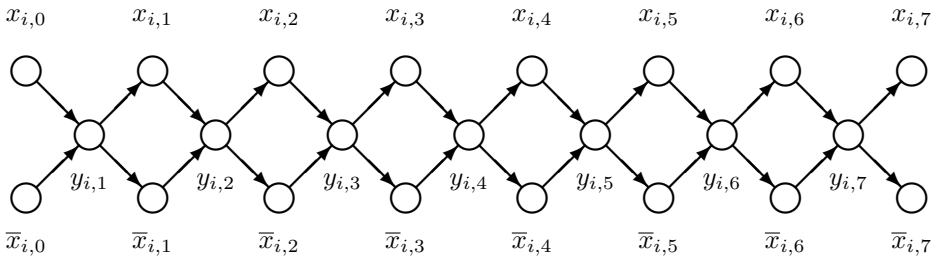


FIG. 3.1. *The literal dag* $\vec{X}_i$.

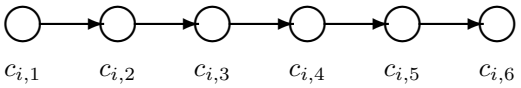


FIG. 3.2. *The clause dag* $\vec{C}_i$.

order in which nodes in the set $\{x_{i,j}, \overline{x}_{i,j}\}$ can appear. The choice of a particular order on the set $\{x_{i,j}, \overline{x}_{i,j}\}$ in the topological ordering of $\vec{H}(P)$ will be interpreted as a particular truth assignment to the variable x_j.

Clause dag $\vec{C}_i$*:*

$$N(\vec{C}_i) = \{c_{i,j} \mid 1 \le j \le 6\},$$
$$A(\vec{C}_i) = \{(c_{i,j}, c_{i,j+1}) \mid 1 \le j < 6\}.$$

The clause dag $\vec{C}_i$ is simply a directed path of length 5 and has a unique topological ordering. Clearly, $\vec{C}_i$ has no transitive arcs. Later we will show how nodes in $\vec{X}_i$ are connected to nodes in $\vec{C}_i$ so as to ensure that a smaller nesting is caused when there is at least one true literal in c_i, then when there is no true literal in c_i. Figure 3.2 shows a clause dag $\vec{C}_i$.

Enforcer dags $\vec{F}_i$ *and* $\vec{F}_i'$*:* The small enforcer dag $\vec{F}_i$ and the big enforcer dag $\vec{F}_i'$ are members of a class of dags

$$\{\vec{F}(q) \mid q \text{ is a positive integer}\}$$

defined as follows:

$$\begin{aligned} N(\vec{F}(q)) &= U(q) \cup V(q) \cup W(q), \\ U(q) &= \{u_j \mid 0 \le j \le q\}, \\ V(q) &= \{v_j \mid 0 \le j \le q\}, \\ W(q) &= \{w_j \mid 0 \le j \le q\}. \\ A(\vec{F}(q)) &= \{(u_j, u_{j-1}) \mid 0 < j \le q\} \\ &\quad \cup \{(v_j, v_{j+1}) \mid 0 \le j < q\} \\ &\quad \cup \{(u_j, w_j) \mid 0 \le j \le q\} \cup \{(w_j, v_j) \mid 0 \le j \le q\}. \end{aligned}$$

Figure 3.3 shows $\vec{F}(3)$. The small enforcer dag $\vec{F}_i$ is isomorphic to $\vec{F}(s)$ for some integer $s \ge 1$, whose value will be determined later. For each j, $0 \le j \le s$, node u_j in $\vec{F}(s)$ is mapped into node $u_{i,j}$ in $\vec{F}_i$; node v_j in $\vec{F}(s)$ is mapped into a node $v_{i,j}$ in $\vec{F}_i$;

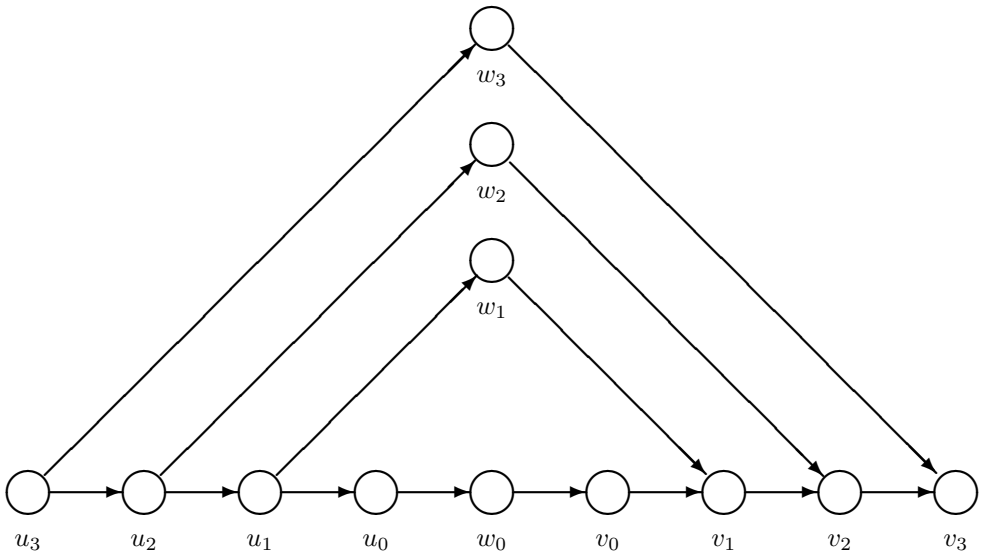


FIG. 3.3. *The dag* $\vec{F}(3)$.

and node w_j is mapped into a node $w_{i,j}$ in $\vec{F}_i$. The big enforcer dag $\vec{F}'_i$ is isomorphic to $\vec{F}(s+t+5)$ for some integer $t \geq 1$, whose value will also be determined later. For each j, $0 \leq j \leq s+t+5$, node u_j in $\vec{F}(s+t+5)$ is mapped into node $u'_{i,j}$ in $\vec{F}'_i$; node v_j in $\vec{F}(s+t+5)$ is mapped into node $v'_{i,j}$ in $\vec{F}'_i$; and node w_j in $\vec{F}(s+t+5)$ is mapped into node $w'_{i,j}$ in $\vec{F}'_i$.

Based on their role, the arcs of $\vec{F}(q)$ can be partitioned into two classes: *base arcs* and 2-*path arcs*. A *base arc* is an arc that belongs to the set

$$\{(u_j, u_{j-1}) \mid 1 \leq j \leq q\} \cup \{(v_j, v_{j+1}) \mid 0 \leq j \leq q-1\}.$$

Any arc that is not a base arc is a 2-*path arc*. Thus the set of 2-path arcs is

$$\{(u_j, w_j), (w_j, v_j) \mid 0 \leq j \leq q\}.$$

The base arcs in $\vec{F}(q)$ along with the 2-path arcs (u_0, w_0) and (w_0, v_0) enforce a unique ordering on the nodes in $U(q) \cup V(q) \cup \{w_0\}$. On the other hand, it is the 2-path arcs that mainly contribute to a rainbow in a layout of $\vec{F}(q)$. To be more precise, we provide the following definitions: Given a total ordering σ on the nodes of a dag $\vec{G}$ and a pair of 2-paths (a_1, b_1, c_1) and (a_2, b_2, c_2) in $\vec{G}$, we say that (a_1, b_1, c_1) and (a_2, b_2, c_2) *nest* in σ if the nodes in $\{a_1, c_1, a_2, c_2\}$ occur in the order a_1, a_2, c_2, c_1 or a_2, a_1, c_1, c_2 in σ. It is easy to see that if a pair of 2-paths nest in a layout, then the queuenumber of that layout is at least 2. Given an ordering σ on the nodes of $\vec{G}$ and a set S of 2-paths in $\vec{G}$, we say that S forms a 2-*path rainbow* in $\vec{G}$ if every pair of 2-paths in S nest in σ. Notice that in any layout of $\vec{F}(q)$ the set of 2-paths $\{(u_j, w_j, v_j) \mid 0 \leq j \leq q\}$ forms a 2-path rainbow of size $q+1$. It is this 2-path rainbow that forces $\vec{F}(q)$ to have a certain queuenumber. Given that we are constructing a dag with no transitive arcs, a simple way to force a certain queuenumber on the layout of the dag is to force a 2-path rainbow of a certain size. This is the key idea that is repeatedly used in our construction.

Heath and Pemmaraju [8] have determined nearly exact bounds on the queuenumber of $\vec{F}(q)$:

$$\lfloor \sqrt{q} \rfloor + 1 \leq QN(\vec{F}(q)) \leq \lfloor \sqrt{q} + 2 \rfloor .$$

In addition, we also know that

$$QN(\vec{F}(6)) = 3; \quad QN(\vec{F}(9)) = 4; \quad QN(\vec{F}(12)) = 4; \quad QN(\vec{F}(16)) = 5.$$

These results provide the basis for our choice of s and t. For notational convenience, denote $QN(\vec{F}(q))$ by $f(q)$. The constants s and t are chosen such that

$$\begin{aligned} f(s+5)+1 &= f(s+5+1) \\ &= f(s+5+t) \\ &= f(s+5+t+1) \\ &= f(s+5+t+2)-1. \end{aligned} \tag{3.1}$$

From the known values of $f(6)$, $f(9)$, $f(12)$, and $f(16)$ shown above it is easy to see that there exist s and t such that (3.1) is satisfied. More specifically, we can choose integers $s, t \geq 1$, such that

$$\begin{aligned} f(s+5) &= 3, \\ f(s+5)+1 &= f(s+5+1) \\ &= f(s+5+t) \\ &= f(s+5+t+1) \\ &= 4, \\ f(s+5+t+2) &= 5. \end{aligned}$$

Having described the four main components of a truth-setting dag $\vec{TS}_i$, we now describe the connections between them. The first set of connections simply ensures that the four dags appear in the order $\vec{X}_i, \vec{F}_i, \vec{C}_i, \vec{F}'_i$ in any topological ordering of $\vec{TS}_i$. These connections are

- $(x_{i,n}, u_{i,s}), (\overline{x}_{i,n}, u_{i,s})$ from $\vec{X}_i$ to $\vec{F}_i$.
- $(v_{i,s}, c_{i,1})$ from $\vec{F}_i$ to $\vec{C}_i$.
- $(c_{i,6}, u'_{i,s+t+5})$ from $\vec{C}_i$ to $\vec{F}'_i$.

The second set of connections depends on the literals that the clause c_i contains, and these are the connections which cause a 2-path rainbow of varying size depending on truth values of the literals in the clause. For these connections we need a set of additional nodes

$$Z_i = \{z_{i,j} \mid 1 \leq j \leq 6\}.$$

These nodes are connected to the clause dag $\vec{C}_i$ through the arcs

$$\{(z_{i,j}, c_{i,j}) \mid 1 \leq j \leq 6\}.$$

The connections between nodes in $\vec{X}_i$ and the nodes in Z_i depend on the literals in clause c_i. Let x_a, x_b, and x_c be the three variables that make up the literals in clause c_i. Without loss of generality, assume that $a < b < c$. If $x_a \in c_i$, then TS_i

contains the arcs $(x_{i,a}, z_{i,5})$ and $(\overline{x}_{i,a}, z_{i,6})$. Otherwise, if $\overline{x}_a \in c_i$, then TS_i contains the arcs $(x_{i,a}, z_{i,6})$ and $(\overline{x}_{i,a}, z_{i,5})$. Similarly, if $x_b \in c_i$, then TS_i contains the arcs $(x_{i,b}, z_{i,3})$ and $(\overline{x}_{i,b}, z_{i,4})$. Otherwise, if $\overline{x}_b \in c_i$, then TS_i contains the arcs $(x_{i,b}, z_{i,4})$ and $(\overline{x}_{i,b}, z_{i,3})$. Finally, if $x_c \in c_i$, then TS_i contains the arcs $(x_{i,c}, z_{i,1})$ and $(\overline{x}_{i,c}, z_{i,2})$. Otherwise, if $\overline{x}_c \in c_i$, then TS_i contains the arcs $(x_{i,c}, z_{i,2})$ and $(\overline{x}_{i,c}, z_{i,1})$. For example, suppose that $c_i = \{x_2, \overline{x}_4, x_7\}$. Then $\vec{TS}_i$ contains the arcs $(x_{i,2}, z_{i,5})$ and $(\overline{x}_{i,2}, z_{i,6})$, corresponding to the positive literal x_2; $(x_{i,4}, z_{i,4})$ and $(\overline{x}_{i,4}, z_{i,3})$ corresponding to the negative literal $\overline{x}_4$; and $(x_{i,7}, z_{i,1})$ and $(\overline{x}_{i,7}, z_{i,2})$ corresponding to the positive literal x_7. Note that the relative order of the pairs of nodes $(x_{i,2}, \overline{x}_{i,2})$, $(x_{i,4}, \overline{x}_{i,4})$, and $(x_{i,7}, \overline{x}_{i,7})$ determines the size of the 2-path rainbow between $\vec{X}_i$ and $\vec{C}_i$. This size could be as small as 3 if the pairs of nodes occurred in the order $(x_{i,2}, \overline{x}_{i,2})$, $(\overline{x}_{i,4}, x_{i,4})$, and $(x_{i,7}, \overline{x}_{i,7})$ and as large as 6 if the pairs of nodes occur in reverse order. This completes the description of a truth-setting dag $\vec{TS}_i$.

Now we describe how the truth-setting dags are connected together to form $\vec{H}(P)$. The arcs

$$(v_{i,s+t+5}, x_{i+1,0}), \qquad (v_{i,s+t+5}, \overline{x}_{i+1,0})$$

ensure that $\vec{TS}_{i+1}$ appears after $\vec{TS}_i$ for all i, $1 \leq i < m$, in any topological ordering of $\vec{H}(P)$. In addition, each truth-setting dag $\vec{TS}_i$ is connected to the truth-setting dag $\vec{TS}_{i+1}$ via 2-paths from nodes in $\vec{X}_i$ to nodes in $\vec{X}_{i+1}$. For these 2-paths we need for each i, $1 \leq i \leq m$, the additional nodes

$$R_i = \{r_{i,j}, \overline{r}_{i,j} \mid 0 \leq j \leq n\}.$$

For each i, $1 \leq i \leq m$, and for each j, $0 \leq j \leq n$, add arcs $(x_{i,j}, r_{i,j})$ and $(\overline{x}_{i,j}, \overline{r}_{i,j})$ to the dag. Complete the 2-paths by adding for each i, j, $1 \leq i < m$, $0 \leq j \leq n$, the arcs $(r_{i,j}, x_{i+1,j})$ and $(\overline{r}_{i,j}, \overline{x}_{i+1,j})$. In addition, to take care of the special case of $\vec{X}_m$, we introduce a new node x with incident arcs:

$$\{(r_{m,j}, x), (\overline{r}_{m,j}, x) \mid 0 \leq j \leq n\} \cup \{(v'_{m,s+t+5}, x)\}.$$

Thus, x is an additional node that occurs at the end of any layout of $\vec{H}(P)$ and to which there are 2-paths from nodes in $\vec{X}_m$. The 2-paths from $\vec{TS}_i$ to $\vec{TS}_{i+1}$ introduced above serve the purpose of causing a 2-path rainbow whose size depends on the order of the pair of nodes $(x_{i,j}, \overline{x}_{i,j})$ in the literal dag $\vec{X}_i$ relative to the order on the pair of nodes $(x_{i+1,j}, \overline{x}_{i+1,j})$ in the literal dag $\vec{X}_{i+1}$. If all the pairs of nodes occur in the same relative order, then the size of the 2-path rainbow is 1; otherwise it is 2. As we shall establish later, this relationship between the relative order of pairs of nodes in the literal dags $\vec{X}_i$ and $\vec{X}_{i+1}$ and the size of the 2-path nesting between $\vec{X}_i$ and $\vec{X}_{i+1}$ is responsible for truth-values "flowing" consistently from clause to clause.

This completes the description of $\vec{H}(P)$.

Let σ be a topological ordering of $\vec{H}(P)$. If $x_{i,j}$ occurs before $\overline{x}_{i,j}$ in σ, then we say that the node $x_{i,j}$ *is assigned true* in σ; otherwise, we say that $x_{i,j}$ *is assigned* $false$ in σ. If $x_{i,j}$ is assigned *true* in σ and $x_j \in c_i$ or if $x_{i,j}$ is assigned $false$ in σ and $\overline{x}_j \in c_i$, then we say that the subgraph C_i is assigned *true* in σ. A topological ordering σ of $\vec{H}(P)$ is said be *satisfiable* if and only if

1. every subgraph C_i is assigned *true* in σ, and
2. for each j, $1 \leq j \leq n$, either $x_{i,j}$ occurs before $\overline{x}_{i,j}$ for all i, $1 \leq i \leq m$, or $x_{i,j}$ occurs after $\overline{x}_{i,j}$ for all i, $1 \leq i \leq m$, in σ. In other words, for each j,

$1 \leq j \leq n$, the pair of nodes $x_{i,j}$ and $\overline{x}_{i,j}$ occur in the same relative order in all literal dags $\vec{X}_i$, $1 \leq i \leq m$, in σ.

Clearly, there exists a satisfiable topological ordering σ of $\vec{H}(P)$ if and only if the given instance of 3-SAT is satisfiable.

Finally, we show that there exists a satisfiable topological ordering of $\vec{H}(P)$ if and only if $QN(\vec{H}(P)) = f(s+t+6) = 4$. The two directions of the "if and only if" are proved separately.

If $QN(\vec{H}(P)) = 4$, then there exists a satisfiable topological ordering of $\vec{H}(P)$. Since $QN(\vec{H}(P)) = 4$, there exists a topological ordering σ of $\vec{H}(P)$ that yields a 4-queue layout of $\vec{H}(P)$. We will show that σ is satisfiable. To obtain a contradiction, assume that σ is not satisfiable. This implies that σ violates conditions 1 or 2 required of a satisfiable topological order. If σ violates condition 2, then there exist some $k, p, 1 \leq k \leq m, 1 \leq p \leq n$, such that the pairs of nodes $(x_{k,p}, \overline{x}_{k,p})$ and $(x_{k+1,p}, \overline{x}_{k+1,p})$ occur in reverse order relative to each other. In other words, $x_{k,p}$ precedes $\overline{x}_{k,p}$ in σ if and only if $x_{k+1,p}$ follows $\overline{x}_{k+1,p}$ in σ. Thus the set

$$\{(x_{k,p}, r_{k,p}, x_{k+1,p}), (\overline{x}_{k,p}, \overline{r}_{k,p}, \overline{x}_{k+1,p})\}$$

forms a 2-path rainbow. This 2-path rainbow nests over the big enforcer dag F'_k and this yields a 2-path rainbow of size $s+t+7$. Therefore the subgraph of $\vec{H}(P)$ induced by the nodes in $\vec{TS}_k$ and $\vec{TS}_{k+1}$ requires at least $f(s+t+7)$ queues to be laid out. But, by our choice of s and t, $f(s+t+7) = f(s+t+6)+1 = 5$. Thus σ yields a layout of $\vec{H}(P)$ that requires at least five queues—a contradiction.

We now show that if σ yields a queue layout of $\vec{H}(P)$ in $f(s+t+6) = 4$ queues, then σ satisfies condition 1. If σ yields an $f(s+t+6)$-queue layout of $\vec{H}(P)$, then all nodes in R_i for all $i, 1 \leq i \leq m$, have to appear between $u_{i,s+t+5}$ (the first node in the layout of $\vec{F}'_i$) and $w_{i,s+t+5}$ (the last node in the layout of $\vec{F}'_i$). This is because, for any i, $1 \leq i < m$, if a node $r \in R_i$ appears to the left of $u'_{i,s+t+5}$, then there is an arc from r to a node in $\vec{TS}_{i+1}$ that nests over $\vec{F}'_i$. If a node in R_m appears to the left of $u'_{m,s+t+5}$, then there is an arc from a node in R_m to x that nests over the layout of $\vec{F}'_m$. Similarly, for any i, $1 \leq i \leq m$, if a node $r \in R_i$ appears to the right of $w_{i,s+t+5}$, then there is an arc from a node in $\vec{TS}_i$ to r that nests over $\vec{F}'_i$. Since, the queuenumber of $\vec{F}'_i$ is $f(s+t+5) = 4$, the arc incident on r that nests over $\vec{F}'_i$ increases the nesting size to $f(s+t+5)+1 = f(s+t+6)+1 = 5$. Therefore, we can assume that in any 4-queue layout of $\vec{H}(q)$ all nodes in R occur between $u_{i,s+t+5}$ and $v_{i,s+t+5}$ in σ.

To obtain a contradiction, suppose that σ does not satisfy condition 1 that satisfiable topological orderings are required to satisfy. Then there exists a clause subgraph C_k that is assigned false. This implies that the 2-paths between the literal dag $\vec{X}_k$, the set of nodes Z_k, and the clause dag $\vec{C}_k$ form a 2-path rainbow of size 6. This 2-path rainbow of size 6 nests over the small enforcer dag $\vec{F}_i$ to yield a 2-path rainbow of total size $s+6$. Thus the subgraph induced by the nodes in $\vec{X}_k$, Z_k, $\vec{C}_k$, and F_k requires at least $f(s+6)$ queues in σ. In addition, the arc $(x_{k,0}, r_{k,0})$ nests over any layout of the subgraph described above to yield a total nesting of size $f(s+6)+1 = 5$. Thus σ requires at least five queues—a contradiction.

If σ is a satisfiable topological ordering, then σ yields a 4-queue layout of $\vec{H}(P)$. If σ is satisfiable, then the largest nesting of 2-paths between $\vec{X}_i$ and $\vec{C}_i$ for any $i, 1 \leq i \leq n$, is of size 5 and the largest nesting of 2-paths between $\vec{X}_i$ and $\vec{X}_{i+1}$ for

any $i, 1 \leq i \leq n-1$, is of size 1. The following is an assignment of arcs of $\vec{H}(P)$ to $f(s+t+6)$ queues such that if $\vec{H}(P)$ is laid out according to σ, then no two arcs assigned to the same queue nest. For some $i, 1 \leq i \leq n$, consider the subgraph of $\vec{H}(P)$ induced by the nodes in $\vec{X}_i$, $\vec{C}_i$, Z_i, and $\vec{F}_i$. The largest 2-path nesting in any layout of this subgraph is of size $s+5$ and hence this subgraph can be laid out in $f(s+5)$ queues. Now consider the subgraph induced by the 2-paths from $\vec{X}_i$ to $\vec{X}_{i+1}$ and the nodes in $\vec{F}'_i$. The largest 2-path nesting in any layout of this subgraph is of size $s+t+6$. Therefore, this subgraph can be laid out in $f(s+t+6)$ queues. Queues can be reused for the assignment of arcs in the two subgraphs to yield a layout for the whole dag in $f(s+t+6)$ queues.

This completes the reduction. Clearly, the reduction can be achieved in polynomial time, thereby showing that POSETQN is NP-complete. □

4. Recognizing 6-stack dags is NP-complete. In this section, we show that DAGSN is also NP-complete.

THEOREM 4.1. *DAGSN is NP-complete.*

Proof. DAGSN is in NP because a nondeterministic Turing machine can guess an ordering of the nodes in $\vec{G}$ and an assignment of the arcs in $\vec{G}$ to six stacks, check if the ordering is a topological ordering, and determine if any two arcs assigned to a stack cross, in polynomial time.

As with POSETQN, DAGSN is shown to be NP-hard by reduction from 3-SAT. Let the collection of clauses $C = \{c_1, c_2, \ldots, c_m\}$ on the set of variables $X = \{x_1, x_2, \ldots, x_n\}$ constitute an instance of 3-SAT. A dag $\vec{G}$ is constructed such that $\vec{G}$ has a 6-stack layout if and only if there exists a truth assignment for the variables in X such that all clauses in C are satisfied.

The construction of $\vec{G}$ from an instance of 3-SAT is much simpler than the corresponding construction of $\vec{H}(P)$ presented in the proof of Theorem 3.1. Corresponding to each clause c_i, $\vec{G}$ contains a *truth-setting dag* $\vec{TS}_i$. Each truth-setting dag $\vec{TS}_i$ consists of three subgraphs connected together:

1. *Literal dag* $\vec{X}_i$,
2. *clause dag* $\vec{C}_i$, and
3. *enforcer dag* E_i.

We will describe each of these three dags, one by one.

Literal dag. The literal dag $\vec{X}_i$ contains two subgraphs $\vec{A}_i$ and $\vec{B}_i$ connected together. The dag $\vec{A}_i$ is as follows:

$$N(\vec{A}_i) = \{x_{i,j} \mid 1 \leq j \leq n\} \cup \{\overline{x}_{i,j} \mid 1 \leq j \leq n\} \cup \{a_{i,1}a_{i,2}\},$$
$$V(\vec{A}_i) = \{(x_{i,j}, x_{i,j+1}), (x_{i,j}, \overline{x}_{i,j+1}), (\overline{x}_{i,j}, x_{i,j+1}), (\overline{x}_{i,j}, \overline{x}_{i,j+1})\}$$
$$\cup \{(a_{i,1}, x_{i,1}), (a_{i,1}, \overline{x}_{i,1}), (x_{i,n}, a_{i,2}), (\overline{x}_{i,n}, a_{i,2})\}.$$

Figure 4.1 shows $\vec{A}_i$. For simplicity we assume that $n = 4$. Any topological ordering of $\vec{A}_i$ contains the node $a_{i,1}$ followed by the nodes in the set $\{x_{i,j}, \overline{x}_{i,j} \mid 1 \leq i \leq n\}$, followed by the node $a_{i,2}$. The nodes in the set $\{x_{i,j}, \overline{x}_{i,j} \mid 1 \leq j \leq n\}$ occur in any order in which the nodes $x_{i,j}$ and $\overline{x}_{i,j}$ are followed by $x_{i,j+1}$ and $\overline{x}_{i,j+1}$ for all j, $1 \leq j < n$. However, for each j, $1 \leq j \leq n$, there is a choice in the order in which the pair of nodes $(x_{i,j}, \overline{x}_{i,j})$ occurs. It is this choice that we exploit to cause a twist of varying size depending on whether the given instance of 3-SAT is satisfiable. Recall that for any ordering σ of the nodes in a dag, a twist in σ is a set of arcs $\{(a_i, b_i) \mid 1 \leq i \leq p\}$ in the dag such that the end-points of the arcs occur in the order

$$a_1, a_2, \ldots, a_p, b_1, b_2, \ldots, b_p$$

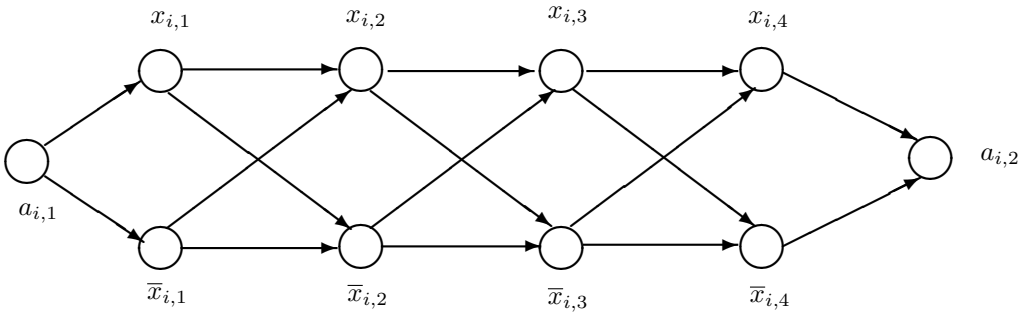


FIG. 4.1. *The dag $\vec{A}_i$ for $n = 4$.*

in σ.

To construct $\vec{B}_i$, first construct a dag $\vec{B}'_i$ that is isomorphic to $\vec{A}_i$. Each node $x_{i,j}$ in $\vec{A}_i$ is mapped into a node $y_{i,j}$ in $\vec{B}'_i$; each node $\overline{x}_{i,j}$ in $\vec{A}_i$ is mapped into a node $\overline{y}_{i,j}$ in $\vec{B}'_i$; each node $a_{i,j}$ in $\vec{A}_i$ is mapped into a node $b_{i,j}$ in $\vec{B}'_i$. To construct $\vec{B}_i$ from $\vec{B}'_i$, simply reverse the direction of all the arcs in $\vec{B}'_i$.

The literal dag $\vec{X}_i$ is constructed by connecting $\vec{A}_i$ and $\vec{B}_i$ using the following arcs:

$$\{(a_{i,1}, b_{i,1}), (a_{i,2}, b_{i,2})\} \cup \{(x_{i,j}, y_{i,j}) \mid 1 \le j \le n\} \cup \{(\overline{x}_{i,j}, \overline{y}_{i,j}) \mid 1 \le j \le n\}.$$

Thus, from each node in $\vec{A}_i$ there is an arc to the corresponding node in $\vec{B}_i$. Note that in any topological ordering of $\vec{G}$ in which the nodes in $\vec{B}_i$ occur in "reverse" order relative to the nodes in $\vec{A}_i$, all arcs connecting $\vec{A}_i$ to $\vec{B}_i$ nest and can be assigned to one stack.

Clause dag. The clause dag $\vec{C}_i$ is simply a directed path of length 5. More precisely,

$$N(\vec{C}_i) = \{c_{i,j} \mid 1 \le j \le 6\},$$
$$A(\vec{C}_i) = \{(c_{i,j}, c_{i,j+1}) \mid 1 \le j < 6\}.$$

Enforcer dag. The enforcer dag $\vec{E}_i$ contains two connected components, $\vec{E}_{i,1}$ and $\vec{E}_{i,2}$. Each of these is simply a directed path of length 4. More precisely,

$$N(\vec{E}_{i,1}) = \{e_{i,j} \mid 1 \le j \le 5\},$$
$$A(\vec{E}_{i,1}) = \{(e_{i,j}, e_{i,j+1}) \mid 1 \le j < 5\},$$

and

$$N(\vec{E}_{i,2}) = \{e_{i,j} \mid 6 \le j \le 10\},$$
$$A(\vec{E}_{i,2}) = \{(e_{i,j}, e_{i,j+1}) \mid 6 \le j < 10\}.$$

We now describe how $\vec{X}_i$, $\vec{C}_i$, and $\vec{E}_i$ are connected together to form the truth-setting dag $\vec{TS}_i$. The arcs $(a_{i,2}, c_{i,1})$, $(c_{i,6}, e_{i,1})$, $(e_{i,5}, b_{i,2})$, and $(b_{i,1}, e_{i,6})$ are added to $\vec{G}$ to force the subgraphs of $\vec{TS}_i$ to occur in the order

$$\vec{A}_i, \vec{C}_i, \vec{E}_{i,1}, \vec{B}_i, \vec{E}_{i,2}$$

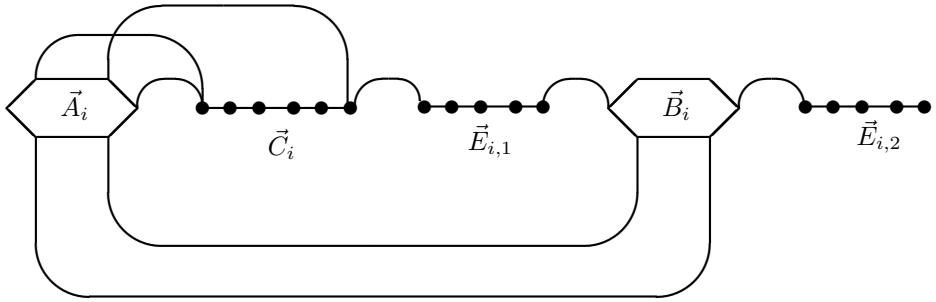


FIG. 4.2. *The overall structure of the truth-setting dag* $\vec{TS}_i$.

in any topological ordering of $\vec{G}$.

In addition we connect $\vec{A}_i$ to $\vec{C}_i$ by six arcs that depend on the literals in the clause c_i. Let x_a, x_b, and x_c be the three variables that make up the literals in clause c_i. Without loss of generality, assume that $a < b < c$. If $x_a \in c_i$, then TS_i contains the arcs $(x_{i,a}, c_{i,2})$ and $(\overline{x}_{i,a}, c_{i,1})$. Otherwise, if $\overline{x}_a \in c_i$, then TS_i contains the arcs $(x_{i,a}, c_{i,1})$ and $(\overline{x}_{i,a}, c_{i,2})$. Similarly, if $x_b \in c_i$, then TS_i contains the arcs $(x_{i,b}, c_{i,4})$ and $(\overline{x}_{i,b}, c_{i,3})$. Otherwise, if $\overline{x}_b \in c_i$, then TS_i contains the arcs $(x_{i,b}, c_{i,3})$ and $(\overline{x}_{i,b}, c_{i,4})$. Finally, if $x_c \in c_i$, then TS_i contains the arcs $(x_{i,c}, c_{i,6})$ and $(\overline{x}_{i,c}, c_{i,5})$. Otherwise, if $\overline{x}_c \in c_i$, then TS_i contains the arcs $(x_{i,c}, c_{i,5})$ and $(\overline{x}_{i,c}, c_{i,6})$. For example, suppose that $c_i = \{x_2, \overline{x}_4, x_7\}$. Then $\vec{TS}_i$ contains the arcs $(x_{i,2}, c_{i,2})$ and $(\overline{x}_{i,2}, c_{i,1})$ corresponding to the positive literal x_2; $(x_{i,4}, c_{i,3})$ and $(\overline{x}_{i,4}, c_{i,4})$ corresponding to the negative literal $\overline{x}_4$; and $(x_{i,7}, c_{i,6})$ and $(\overline{x}_{i,7}, c_{i,5})$ corresponding to the positive literal x_7. Depending on the order of the pairs of nodes $(x_{i,a}, \overline{x}_{i,a})$, $(x_{i,b}, \overline{x}_{i,b})$, and $(x_{i,c}, \overline{x}_{i,c})$ these arcs form a twist of size at least 3 and at most 6. In particular, we will try to ensure that these arcs form a twist of size 6 if and only if all literals in c_i are false.

This completes the description of the truth-setting dag $\vec{TS}_i$. The overall structure of $\vec{TS}_i$ is shown in Figure 4.2. We now describe how $\vec{TS}_i$ is connected to $\vec{TS}_{i+1}$ for each i, $1 \le i < m$. To ensure that $\vec{TS}_{i+1}$ occurs after $\vec{TS}_i$ in any topological ordering of $\vec{G}$, we add the arc $(e_{i,10}, a_{i+1,1})$. In addition we have the following arcs from $\vec{B}_i$ to $\vec{A}_{i+1}$:

$$\{(y_{i,j}, x_{i+1,j}) \mid 1 \le j \le n\} \cup \{(\overline{y}_{i,j}, \overline{x}_{i+1,j}) \mid 1 \le j \le n\}.$$

To complete the description of the dag $\vec{G}$ we need to describe a final component, called a *gate-keeper* dag, $\vec{GK}$, which is simply a directed path of length 5. More precisely, the gate-keeper dag is as follows:

$$N(\vec{GK}) = \{g_k \mid 1 \le k \le 5\},$$
$$A(\vec{GK}) = \{(g_k, g_{k+1}) \mid 1 \le k < 5\}.$$

An additional arc $(g_5, a_{1,1})$ ensures that the gate-keeper dag occurs before the rest of the dag in any topological ordering of $\vec{G}$. This completes the description of $\vec{G}$.

Let σ be a topological ordering of $\vec{G}$. If $x_{i,j}$ occurs before $\overline{x}_{i,j}$ in σ, then we say that the node $x_{i,j}$ *is assigned true* in σ; otherwise, we say that $x_{i,j}$ *is assigned false* in σ. If $x_{i,j}$ is assigned *true* in σ and $x_j \in c_i$ or if $x_{i,j}$ is assigned *false* in σ and $\overline{x}_j \in c_i$, then we say that the clause dag $\vec{C}_i$ is assigned *true* in σ; otherwise, we say

that the clause dag $\vec{C}_i$ is assigned *false* in σ. A topological ordering σ of $\vec{G}$ is said be *satisfiable* if and only if

1. every clause dag $\vec{C}_i$ is assigned *true* in σ, and
2. for each j, $1 \leq j \leq n$, either $x_{i,j}$ occurs before $\overline{x}_{i,j}$ for all i, $1 \leq i \leq m$, or $x_{i,j}$ occurs after $\overline{x}_{i,j}$ for all i, $1 \leq i \leq m$, in σ. In other words, for each j, $1 \leq j \leq n$, the pair of nodes $x_{i,j}$ and $\overline{x}_{i,j}$ occurs in the same relative order in all literal dags $\vec{X}_i$, $1 \leq i \leq m$, in σ.

Clearly, there exists a satisfiable topological ordering σ of $\vec{H}(P)$ if and only if the given instance of 3-SAT is satisfiable.

Finally, to complete our proof, we show that there exists a satisfiable topological ordering of $\vec{G}$ if and only if $SN(\vec{G}) = 6$. The two directions of the "if and only if" are proved separately.

If $SN(\vec{G}) = 6$, then there exists a satisfiable topological ordering of $\vec{G}$. Let σ be any topological ordering of $\vec{G}$ that yields a 6-stack layout of $\vec{G}$. We will show that σ is satisfiable. To obtain a contradiction assume that σ is not satisfiable. Therefore, σ violates one of the two conditions required of a satisfiable topological ordering. Suppose that σ violates condition 1. In particular, suppose that there exists a clause dag $\vec{C}_i$ that is not assigned *true* in σ. This implies that the set of six arcs from $\vec{A}_i$ to $\vec{C}_i$ forms a twist of size 6. The arc $(a_{i,2}, b_{i,2})$ adds to this twist to cause a twist of size 7. This implies that the layout of $\vec{G}$ according to σ requires at least seven stacks—a contradiction. Now suppose that σ violates condition 2. Thus there exist k, p, $1 \leq k \leq m$, $1 \leq p \leq n$, such that the pairs of nodes $(x_{k,p}, \overline{x}_{k,p})$ and $(x_{k+1,p}, \overline{x}_{k+1,p})$ do not occur in the same relative order. In other words, $x_{k,p}$ precedes $\overline{x}_{k,p}$ if and only if $x_{k+1,p}$ follows $\overline{x}_{k+1,p}$. Without loss of generality, suppose that $x_{k,p}$ precedes $\overline{x}_{k,p}$. This implies that $x_{k+1,p}$ follows $\overline{x}_{k+1,p}$. The pair of nodes $(y_{k,p}, \overline{y}_{k,p})$ can occur in one of two possible orders in σ. If $y_{k,p}$ precedes $\overline{y}_{k,p}$ in σ, then the arcs in the set

$$\{(x_{k,p}, y_{k,p}), (\overline{x}_{k,p}, \overline{y}_{k,p})\}$$

form a twist of size 2. The five arcs from the gate-keeper dag $\vec{GK}$ to the enforcer dag component $\vec{E}_{i,1}$

$$\{(g_j, e_{i,j}) \mid 1 \leq j \leq 5\}$$

add to the above twist to yield a twist of size 7. Therefore, the layout of $\vec{G}$ according to σ requires at least seven stacks—a contradiction. Similarly, if $y_{k,p}$ follows $\overline{y}_{k,p}$, it is easy to see that the arcs in the set

$$\{(y_{k,p}, x_{k+1,p}), (\overline{y}_{k,p}, \overline{x}_{k+1,p})\}$$

along with the five arcs from the gate-keeper dag to the second component of the enforcer dag $\vec{E}_{i,2}$ form a twist of size 7, yet again leading to a contradiction.

If there exists a satisfiable topological ordering of $\vec{G}$, then $SN(\vec{G}) = 6$. Let σ be a satisfiable topological ordering of $\vec{G}$. Without loss of generality, we assume that in σ for all i, $1 \leq i \leq m$, the nodes in $\vec{B}_i$ occur in a "reverse" order relative to the order of nodes in $\vec{A}_i$. This ensures that the arcs from $\vec{A}_i$ to $\vec{B}_i$ can be assigned to one stack and the arcs from $\vec{B}_i$ to $\vec{A}_{i+1}$ can also be assigned to one stack. We demonstrate an assignment of the arcs of $\vec{G}$ to six stacks so that when the nodes in $\vec{G}$ are laid out according to σ, no two arcs assigned to a stack cross.

It does not matter how we deal with arcs that connect nodes that are adjacent in σ. In particular, we will ignore the arcs in the gate-keeper dag $\vec{GK}$, the arcs in the

clause dag $\vec{C}_i$, $1 \leq i \leq m$, and the arcs in the enforcer dag $\vec{E}_i$, $1 \leq i \leq m$. We will also ignore some arcs that connect different subgraphs, namely, the arc $(g_5, a_{1,1})$, and for all i, $1 \leq i \leq m$, the arcs

$$(a_{i,2}, c_{i,1}), (c_{i,5}, e_{i,1}), (e_{i,5}, b_{i,2}), (b_{i,1}, e_{i,6}).$$

We will assign the rest of the arcs to six stacks, s_k, $1 \leq k \leq 6$. Start with the arcs incident on the gate-keeper dag. For each k, $1 \leq k \leq 4$, assign all arcs incident on g_k to stack s_k. Assign the arcs incident on g_5 to two stacks as follows. First assign the arcs from g_5 to the enforcer dag component $\vec{E}_{i,1}$ to s_5 for all i, $1 \leq i \leq m$. Then assign the arcs from g_5 to the enforcer dag component $\vec{E}_{i,2}$ to s_6 for all i, $1 \leq i \leq m$. This partial assignment of arcs of $\vec{G}$ to stacks fixes, to a large extent, the assignment of the rest of the arcs to stacks. In particular, the arcs from $\vec{A}_i$ to $\vec{B}_i$, for all i, $1 \leq i \leq m$, cross the arcs from $G\vec{K}$ to $\vec{E}_{i,1}$ and therefore have to be assigned to the stack s_6. Similarly, the arcs from $\vec{B}_i$ to $\vec{A}_{i+1}$, for all i, $1 \leq i < m$, cross the arcs from $G\vec{K}$ to $\vec{E}_{i,2}$ and therefore have to be assigned to the stack s_5. Since σ is a satisfiable topological ordering of $\vec{G}$, we know that the arcs from $\vec{A}_i$ to $\vec{C}_i$ form a twist of size at most 5. Since these arcs cross the arcs from $\vec{A}_i$ to $\vec{B}_i$, assign these arcs to stacks s_1 through s_5. It is easy to reuse the stacks s_1 through s_5 for the arcs in $\vec{A}_i$ and in $\vec{B}_i$.

Thus DAGSN is NP-complete. □

5. Conclusions. This paper presents fundamental algorithmic results concerning the computational complexity of determining the stacknumber and queuenumber of dags. For both stacks and queues, there remains a gap between the number of stacks or queues for which an NP-completeness result is known and the number for which a polynomial-time algorithm is known. We conjecture that recognition of both 2-stack and 2-queue dags is NP-complete. A more fruitful line of research is to identify classes of graphs for which the layout problem can be solved efficiently.

REFERENCES

[1] F. Bernhart and P. C. Kainen, *The book thickness of a graph*, J. Combin. Theory Ser. B, 27 (1979), pp. 320–331.

[2] K. S. Booth and G. S. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 335–379.

[3] M. Chandramouli and A. A. Diwan, *Upward numbering testing for triconnected graphs*, in Graph Drawing, GD '95, F. J. Brandenburg, ed., Lecture Notes in Comput. Sci. 1027, Springer-Verlag, Berlin, 1996, pp. 140–151.

[4] M. Chandramouli, *Upward Planar Graph Drawings*, Ph.D. thesis, IIT Bombay, 1994.

[5] G. Di Battista and E. Nardelli, *Hierarchies and planarity theory*, IEEE Trans. Systems Man Cybernet., 18 (1988), pp. 1035–1046.

[6] F. Harary and G. Prins, *The block-cutpoint-tree of a graph*, Publ. Math. Debrecen, 13 (1966), pp. 103–107.

[7] L. S. Heath and S. V. Pemmaraju, *Recognizing leveled-planar dags in linear time*, in Graph Drawing, GD '95, F. J. Brandenburg, ed., Lecture Notes in Comput. Sci. 1027, Springer-Verlag, Berlin, 1996, pp. 300–311.

[8] L. S. Heath and S. V. Pemmaraju, *Stack and queue layouts of posets*, SIAM J. Discrete Math., 10 (1997), pp. 599–625.

[9] L. S. Heath, S. V. Pemmaraju, and A. Trenk, *Stack and queue layouts of directed acyclic graphs: Part* I, SIAM J. Comput., (28 (1999), pp. 1510–1539.

[10] L. S. Heath, S. V. Pemmaraju, and A. Trenk, *Stack and queue layouts of directed acyclic graphs*, in Planar Graphs, W. T. Trotter, ed., AMS, Providence, RI, 1993, pp. 5–11.

[11] L. S. Heath and A. L. Rosenberg, *Laying out graphs using queues*, SIAM J. Comput., 21 (1992), pp. 927–958.
[12] M. M. Sysło and M. Iri, *Efficient outerplanarity testing*, Fund. Inform., 2 (1979), pp. 261–275.
[13] M. Yannakakis, *The complexity of the partial order dimension problem*, SIAM J. Alg. Discrete Methods, 3 (1982), pp. 351–358.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1627–1640

MEMBERSHIP IN CONSTANT TIME AND ALMOST-MINIMUM SPACE*

ANDREJ BRODNIK† AND J. IAN MUNRO‡

Abstract. This paper deals with the problem of storing a subset of elements from the bounded universe $\mathcal{M} = \{0, \ldots, M-1\}$ so that membership queries can be performed efficiently. In particular, we introduce a data structure to represent a subset of N elements of $\mathcal{M}$ in a number of bits close to the information-theoretic minimum, $B = \left\lceil \lg \binom{M}{N} \right\rceil$, and use the structure to answer membership queries in constant time.

Key words. information retrieval, search strategy, data structures, minimum space, dictionary problem, efficient algorithms hashing, lower bound

AMS subject classifications. 68P05, 68P10, 68Q20

PII. S0097539795294165

1. Introduction.

A basic problem in computing is to store a finite set of elements so that one can quickly determine whether or not a query element is a *member* of this set. In this paper we study a version of the problem in which elements are drawn from the bounded universe $\mathcal{M} = \{0, \ldots, M-1\}$ using an extended random access machine (RAM) model that permits constant-time arithmetic and Boolean bitwise operations on these elements. Such a very realistic model enables us to decrease the space needed to store a set of N elements almost to the information-theoretic minimum of $B = \lceil \lg \binom{M}{N} \rceil$ bits, while answering queries in constant time.

Fich and Miltersen [12] have shown that, under a RAM model whose instruction set does not include division, $\Omega(\log N)$ operations are necessary to answer a membership query if the size of a data structure is at most M/N^{ϵ} words of $\lceil \lg M \rceil$ bits each. Thus a sorted array is optimal in that context. Our model includes integer division along with the other standard operations in its instruction set. This permits us to use perfect hash tables (functions) and bitmaps, both of which have constant-time worst-case behavior. However, hash tables generally require that key values be stored explicitly, and so are succinct only when relatively few elements are present. On the other hand, a bitmap is succinct only if about half of the elements are present. In this paper we focus primarily on the range in which N is at least M^{ϵ}, but still $o(M)$, with the goal of introducing a data structure whose size is within a lower-order term of the minimum.

In general terms, our basic approach is to use either perfect hashing or a bitmap whenever one of them achieves the optimum space bound; otherwise we split the

*Received by the editors November 5, 1995; accepted for publication (in revised form) April 10, 1998; published electronically May 6, 1999. This work was supported in part by the Natural Science and Engineering Research Council of Canada under grant A-8237 and the Information Technology Research Centre of Ontario and was done while the first author was a graduate student at the University of Waterloo. Some of the results of this work were announced in preliminary form, in *Membership in constant time and minimum space*, in Proceedings, 2nd European Symposium on Algorithms, Lecture Notes in Comput. Sci. 855, Springer-Verlag, Berlin, New York, 1994, pp. 72–81.
http://www.siam.org/journals/sicomp/28-5/29416.html

†Department of Theoretical Computer Science, Institute of Mathematics, Physics, and Mechanics, University of Ljubljana, Jadranska 11, 1111 Ljubljana, Slovenia, and Department of Computer Science, Luleå Technical University, SE-971 87 Luleå, Sweden (andrej.Brodnik@IMFM.Uni-Lj.SI).

‡Department of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada (imunro@uwaterloo.ca).

universe into subranges of equal size. We discover that, after a couple of careful iterations of this splitting, the subranges are small enough so that succinct indices into a single table of all possible configurations of these subranges (*table of small ranges*) permit the encoding in the minimal space bound. This is an example of what we call *word-size truncated recursion* (cf. [15, 16]). That is, the recursion continues only to a level of "small enough" subproblems, at which point indexing into a table of all solutions suffices. We can do this because at this level a single word in the machine model is large enough to encode a complete solution to each of these small problems.

We proceed with definitions, notation, and background literature. In section 3 we present a constant-time solution with space bound within a small constant factor of the minimum required. The solution has the merit of providing a reasonably practical implementation, and can be tuned to specific problem sizes as is illustrated in giving the space requirements for two specific examples. In section 4 the solution is further tuned to achieve the asymptotic space bound of $B + o(B)$. The results of sections 3 and 4 are extended in section 5 to the dynamic case.

2. Notation, definitions, and background.

2.1. The problem. We use lg to denote the logarithm base 2 and ln to denote the natural logarithm. $\lg^{(i)}$ indicates lg applied i times and $\lg^*$ indicates the number of times lg can be applied before reducing the parameter to at most 1.

DEFINITION 2.1. *Given a universe $\mathcal{M} = \{0, 1, \ldots, M-1\}$ with an arbitrary subset $\mathcal{N} = \{e_0, e_1, \ldots, e_{N-1}\}$, where N and M are known, the* static membership problem *is to determine whether a query value $x \in \mathcal{M}$ is in $\mathcal{N}$.*

This problem has an obvious dynamic extension leading to the following definition.

DEFINITION 2.2. *The* dynamic membership problem *is the static membership problem extended by two operations: insertion of an element $x \in \mathcal{M}$ into $\mathcal{N}$ (if it is not already in $\mathcal{N}$) and deletion of x from a set $\mathcal{N}$ (if it is in $\mathcal{N}$).*

Since solving either problem for $\mathcal{N}$ trivially gives a solution for $\overline{\mathcal{N}}$, we assume $0 \le N \le M/2$.

Our model of computation is an extended version of the RAM machine model (cf. [1]; see also MBRAM in [9]). Memory consists of words of $m = \lceil \lg M \rceil$ bits, which means that one memory register (word) can be used to represent a single element of $\mathcal{M}$, specify an arbitrary subset of a set of m elements, refer to some portion of the data structure, or have some other role that is an m-bit blend of these. For convenience, we measure space in bits rather than in words. Our word size, then, matches the problem size, and so the model is *transdichotomous* in the sense of Fredman and Willard [14]. The usual operations, including integer multiplication, division, and bitwise Boolean operations, are performed in unit time.

We take as parameters of our problem M and N. Hence,

$$B = \left\lceil \lg \binom{M}{N} \right\rceil \tag{2.1}$$

is an information-theoretic lower bound on the number of bits required to describe any possible subset of N elements chosen from M elements. Since we are interested only in solutions that use $O(B)$ or $B + o(B)$ bits for a data structure, there is no need to pay attention to rounding errors, and so we can omit the ceiling and floor functions.

Using Stirling's approximation for the factorial function and Robbins' refinement for its error term (cf. [20, p. 184]), we compute from (2.1) a lower bound on the

number of bits required,

$$\begin{aligned} B &= \lg \binom{M}{N} = \lg M! - \lg N! - \lg(M-N)! \\ &\approx M \lg M - N \lg N - (M-N)\lg(M-N) \quad \text{error} \le \lg N + \mathrm{O}(1) \\ &= M \lg M - N \lg N \\ &\quad -(M-N)(\lg M + \lg(1 - N/M)) \\ &= N \lg(M/N) - (M-N)\lg(1-N/M). \end{aligned} \tag{2.2}$$

Defining the *relative sparseness* of the set $\mathcal{N}$ as

$$r = M/N \tag{2.3}$$

and observing that $2 \le r \le \infty$, we rewrite the second term of (2.2) as

$$N \le -N((r-1)\lg(1-r^{-1})) \le N/\ln 2 \approx 1.4427\ldots N. \tag{2.4}$$

Thus, for the purposes of much of this work, we can use

$$B \approx N \lg(M/N) \equiv N \lg r \tag{2.5}$$

with an error bounded of $\Theta(N)$ bits as given in (2.4). Note that the error term is positive and hence (2.5) is an underestimate.

An intuitive explanation of (2.5) is that $\mathcal{N}$ is fully described when each element in $\mathcal{N}$ "knows" its successor. Since there are N elements in $\mathcal{N}$, the average distance between them is $r = M/N$; to encode this distance we need $\lg r$ bits. Moreover, it is not hard to argue that the worst case, and indeed the average one, occurs when elements are fairly equally spaced. This is exactly what (2.5) says.

2.2. Some background literature. This paper deals with one of the most heavily studied problems in computing, in a context in which the exact model of computation is critical. Therefore, we suggest [9] and [18] as general background and focus on those papers that most heavily shaped the authors' approach. We address three aspects of the problem: the static and dynamic cases of storing a table with little auxiliary data and the information-theoretic trade-offs. In the first two cases, it is usually assumed that there is enough space to list those keys that are present (in a hash table or similar structure) or to list the answers to all queries (by using a bitmap). Here we deal with the situation in which we cannot always afford the space needed to use either structure directly. Nonetheless, we start with the idea of storing keys and little else.

Yao [24] extended the notion of an implicit data structure [21] to the domain of the bounded universe and addressed the problem of storing the value N and an array of N words, each containing a $\lg M$ bit data item. He showed that if no more information is stored, then there always exists some value of N and subset of size N that requires at least logarithmic search time. Adding almost any storage, however, changes the situation. For example, with one more register ($\lg M$ bits) Yao [24] showed that there exists a constant-time solution for $N \approx M$ or $N \le \frac{1}{4}\sqrt{\lg M}$, while Tarjan and Yao [23] presented a more general $\mathrm{O}(\lg M/\lg N)$ time, $\mathrm{O}(N \lg M)$ bit solution. Fredman, Komlós, and Szemerédi [13, sec. 4] developed a constant-time algorithm with a data structure of $N \lg M$ bits for the portion of the data, plus an additional $\mathrm{O}(N\sqrt{\lg N} + \lg^{(2)} M)$ bits. Fiat et al. [11] decreased the extra bits to

$6 \lg N + 3\lceil \lg^{(2)} M \rceil + O(1)$. Moreover, combining their result with Fiat and Naor's [10] construction of an implicit search scheme for $N = \Omega((\lg M)^p)$, they produced a scheme using fewer than $(1+p)\lceil \lg^{(2)} M \rceil + O(1)$ additional bits.

Mairson [17] took a different approach. He assumed all structures are implicit in Yao's sense and the additional storage represents the complexity of a searching program. Following a similar path, Schmidt and Siegel [22] proved a lower bound of $\Omega(N/(k^2 e^k) + \lg^{(2)} M)$ bits spatial complexity for k-probe oblivious hashing. In particular, for constant-time hashing this gives a spatial complexity of $\Theta(\lg^{(2)} M + N)$ bits.

For the dynamic case, Dietzfelbinger et al. [5] proved an $\Omega(\lg N)$ worst-case lower bound for a realistic class of hashing schemes. In the same paper they also presented a scheme which, using results of [13] and a standard doubling technique, achieved constant amortized expected time per operation. However, the worst-case time per operation (nonamortized) was $\Omega(N)$. Later Dietzfelbinger and Meyer auf der Heide [6] upgraded the scheme and achieved constant worst-case time per operation with a high probability. A similar result was also obtained by Dietzfelbinger et al. [4].

In the data compression technique described by Choueka et al. [3], a bit vector is hierarchically compressed. First, the binary representations of elements stored in the dictionary are split into pieces of equal size. Then the elements with the same value as the most significant piece are put in the same bucket and the technique is recursively applied within each bucket. When the number of elements which fall in the same bucket becomes sufficiently small, the data are stored in a compressed form. The authors experimentally tested their ideas but did not formally analyze them. They claim their result gives a relative improvement of about 40% over similar methods.

An information-theoretic approach was taken by Elias [7] in addressing a more general version of the static membership problem which involved several different types of queries. For these queries he discussed a trade-off between the size of the data structure and the average number of bit probes required to answer the queries. In particular, for the set membership problem he described a data structure of a size $N \lg(M/N) + O(N)$ (using (2.5), $B + o(B)$) bits, which required an average of $(1+\epsilon)\lg N + 2$ bit probes to answer a query. However, in the worst case the method required N bit probes. Elias and Flower [8] further generalized the notion of a query into a database. They defined the set of data and a set of queries and, in a general setting, studied the relation between the size of the data structure and the number of bits probed, given the set of all possible queries. Later, the same arrangement was more rigorously studied by Miltersen [19].

3. Static solution using $O(B)$ space. Our solution breaks down to a number of cases, based on the relative sparseness of $\mathcal{N}$. As noted earlier, we can assume that at most half the elements are present, since the complementary problem could otherwise be solved. We are left with four cases as r ranges between 2 and ∞ (cf. Table 3.1). The crucial dividing point between the sparse and dense cases comes when r is in the range $\Theta(\lg M)$. For purposes of tuning the method, we find it convenient to define this *separation point* in terms of a parameter $\lambda(>1)$, namely,

$$r_{sep} = \log_\lambda M, \tag{3.1}$$

or the size of sets

$$N_{sep} = M/r_{sep} = M/\log_\lambda M. \tag{3.2}$$

TABLE 3.1
Cases considered for the static version of the problem.

Sparseness	Range of r			Range of N			Section
Very sparse	∞	to	M^ϵ	0	to	$M^{1-\epsilon}$	3.1
Moderately sparse	M^ϵ	to	$\log_\lambda M$	$M^{1-\epsilon}$	to	$M/\log_\lambda M$	3.2
Moderately dense	$\log_\lambda M$	to	$1/\alpha$	$M/\log_\lambda M$	to	αM	3.3
Very dense	$1/\alpha$	to	2	αM	to	$M/2$	3.1

The very sparse and very dense cases are rather straightforward, though their boundaries with the more difficult moderately sparse and moderately dense cases are subject to tuning as well. After handling these easy cases, we address the moderately sparse case and subsequently extend its solution to handle the moderately dense.

3.1. Very sparse and very dense cases. When $\mathcal{N}$ is very dense, i.e., $N \geq \alpha M$ for $0 < \alpha \leq 1/2$, we can afford to use a bitmap of size $M = \Theta(B)$ to represent it. When $\mathcal{N}$ is very sparse, i.e., $N \leq M^{1-\epsilon}$ for $0 < \epsilon \leq 1$, we are allowed $\Theta(N \log M)$ bits which is enough to list all the elements of $\mathcal{N}$. For $N \leq c = \mathrm{O}(1)$ we simply list them. Beyond this we use a perfect hashing function of some form (cf. [10, 11, 13]). Note that all of these structures allow us to answer a membership query in constant time and are, indeed, reasonably practical methods.

3.2. Moderately sparse case—indexing. The range in which $r \approx r_{sep}$ typifies the case in which neither the straightforward listing of the elements nor a bitmap minimizes the storage requirements. In this range, the $N \lg M$ bits needed to list all elements is of the same order as the M for a bitmap, but $B = \Theta(N \lg^{(2)} M)$. Indeed, thoughts of this specific case lead not only to a solution to the entire moderately sparse range, but also to the first step in the solution for the moderately dense case.

LEMMA 3.1. *If $N \leq N_{sep} = M/\log_\lambda M$ for $\lambda > 1$, then there is an algorithm which answers a membership query in constant time using an $\mathrm{O}(B)$ bit data structure.*

Proof. The idea is to split the universe, $\mathcal{M}$, into p buckets, where p is as large as we can make it without exceeding our space constraints. The data falling into individual buckets are then organized using perfect hashing. The buckets cover contiguous ranges of equal sizes, $M_1 = \lfloor M/p \rfloor$, so that a key $x \in \mathcal{M}$ falls into bucket $\lfloor x/M_1 \rfloor$. To reach individual buckets, we index through an array of pointers.

Each pointer occupies $\lceil \lg M \rceil$ bits. Hence, the total size of the *index* (the array of pointers) is $p \lceil \lg M \rceil$ bits. We store all elements that fall in the same bucket in a perfect hash table [10, 11, 13] for that bucket. Since the ranges of all buckets are of equal size, the space required to describe each element in a hash table is $\lceil \lg(M/p) \rceil$ bits, and so to describe all elements in all buckets we require only $N \lceil \lg(M/p) \rceil$ bits. We also need some extra space to describe individual hash tables. If we use the method of Fiat et al. [11], the additional space for bucket i is bounded by $6\lceil \lg N_i \rceil + 3\lceil \lg^{(2)} M_1 \rceil + \mathrm{O}(1)$, where N_i is the number of elements in bucket i. Thus, the additional space to describe all hash functions is bounded by $p(6 \lg N + 3 \lg^{(2)} M + \mathrm{O}(1))$. Putting the pieces together, we get the expression for the size of the structure:

$$S = p \lg M + N \lg(M/p) + p(6 \lg N + 3 \lg^{(2)} M + \mathrm{O}(1)).$$

Choosing p to minimize this value leads to a rather complex formula. However, a simple approximation is adequate, and so we take

$$p = N/\lg M. \tag{3.3}$$

This gives

$$\begin{aligned} S &= N + N(\lg M/N + \lg^{(2)} M) + \\ &\quad N(6\lg N + 3\lg^{(2)} M + \mathrm{O}(1))/\lg M \qquad \text{by (2.3)} \\ &\le N\lg r + (N\lg r)(\lg^{(2)} M/\lg r) + N + \\ &\quad N(6\lg N + 3\lg^{(2)} M + \mathrm{O}(1))/\lg M \qquad \text{by (2.5)} \\ &= B + B\lg^{(2)} M/\lg r + \mathrm{o}(B). \end{aligned} \tag{3.4}$$

Hence, for a moderately sparse subset, i.e., $r \ge r_{sep}$, the size of the structure is $\mathrm{O}(B)$ bits. It is also easy to see that the structure permits constant-time search. □

Note that if $r_{sep} \ge \lg M$ (i.e., in (3.1) $\lambda < 2$), the lead term of (3.4) is less than $2B$.

3.3. Moderately dense case—word-size truncated recursion. In this section we consider sets of size N (or sparseness r) in the range

$$\begin{array}{rccccccc} N_{sep} = M/\log_\lambda M & \le & N & \le & \alpha M & \le & M/2, \\ r_{sep} = \log_\lambda M & \ge & r & \ge & 1/\alpha & \ge & 2. \end{array} \tag{3.5}$$

For such *moderately dense* $\mathcal{N}$ we apply the technique of Lemma 3.1—that is, split the universe $\mathcal{M}$ into equal-range buckets. However, this time the buckets remain too full to use hash tables and therefore we apply the splitting scheme again. In particular, we treat each bucket as a new, separate-but-smaller, universe. If its relative sparseness falls in the range defined by (3.5) (with respect to the size of its smaller universe) we recursively split it.

Such a straightforward strategy leads, in the worst case, to a $\Theta(\lg^* M)$ level structure and therefore to a $\Theta(\lg^* M)$ search time. However, we observe that at each level the number of buckets with the same range increases and ultimately there must be so many small subsets that not all can be different. Therefore we build a table of all possible subsets of universes of size up to a certain threshold. This *table of small ranges* (TSR) allows replacement of buckets in the main structure by pointers (indices) into the table. Although the approach is not new (cf. [15, 16]), it does not appear to have been given a name. We refer to the technique as *word-size truncated recursion*. In our structure the truncation occurs after two splittings. In fact, because all our second-level buckets have the same range, our TSR consists of all possible subsets of only a single small universe. In the rest of this section we give a detailed description of the structure and its analysis.

On the first split we partition the universe into

$$p = N_{sep}/\lg M = M/(\log_\lambda M \lg M) \tag{3.6}$$

buckets, each of which has a range $M_1 = M/p$. At the second level we have, again, relatively sparse and dense buckets which now separate at the relative sparseness

$$r'_{sep} = \log_\lambda M_1 = \log_\lambda(M/p) = \mathrm{O}(\lg^{(2)} M). \tag{3.7}$$

For sparse buckets we apply the solution of section 3.2, and for very dense ones with more than the fraction α of their elements present we use a bitmap. For the moderately dense buckets, with relative sparseness within the range defined in (3.5), we reapply the splitting. However, this time the number of buckets is (cf. (3.6))

$$p_1 = M_1/(r'_{sep}\lg M_1), \tag{3.8}$$

so that each of these smaller buckets has the same range,

$$M_2 = M_1/p_1 = \mathrm{O}((\lg^{(2)} M)^2), \tag{3.9}$$

because $\lg M_1 = \mathrm{O}(\lg^{(2)} M)$.

At this point we use the TSR. This table consists of bitmap representations of all subsets of the universe of size M_2. Thus we can replace buckets in the main structure with "indices" (pointers of varying sizes) into the table. We order the table first according to the number of elements in the subset and then lexicographically. We store a pointer in the TSR as a record consisting of two fields: ν, the number of elements in the bucket, which takes $\lceil \lg M_2 \rceil$ bits; and β, the rank of this bucket with respect to the lexicographic order among all buckets containing ν elements. To store β, by (2.1), takes $B_\nu = \lceil \lg \binom{M_2}{\nu} \rceil$ bits. The actual position (index) of the corresponding bitmap of the bucket in the TSR is thus

$$\sum_{i=1}^{\nu-1} \binom{M_2}{i} + \beta - 1. \tag{3.10}$$

The sum is found by table lookup and so a search is performed in constant time.

This concludes the description of the data structure also presented in Algorithm 3.1. As demonstrated in Algorithm 3.2, the data structure allows constant-time membership queries, but it remains to be seen how much space it occupies. The algorithm uses functions `LookUpBM`—look up bitmap; `FindOL`—find in ordered list; and `FindHT`—find in hash table, whose descriptions are omitted. However, their particular implementation suggests the constants c, ϵ, λ, and α used for the fine tuning of Algorithm 3.2.

ALGORITHM 3.1. DATA STRUCTURE FOR THE SOLUTION OF THE STATIC PROBLEM.

```
TYPE
tCases=
  (eEmpty,                                              (* N = 0 *)
   eVerySparse1,                                        (* 0 < N ≤ c *)
   eVerySparse2,                                        (* c < N ≤ M^(1-ε) *)
   eModeratelySparse,                                   (* M^(1-ε) < N ≤ M / log_λ M *)
   eModeratelyDense,                                    (* M / log_λ M < N ≤ α M *)
   eVeryDense);                                         (* α M < N < M/2 *)
tSet= RECORD CASE BOOLEAN OF
TRUE:                                                   (* current universe is at most M2 *)
  ν, β;
FALSE:                                                  (* general case *)
 N;                                                     (* size of the set *)
  CASE tCases OF
   eEmpty: ;                                            (* nothing *)
   eVerySparse1:                                        (* (un)ordered list *)
     list: ARRAY [] OF tElement;
   eVerySparse2:                                        (* hash table *)
     hashTable: tHashTable;
   eModeratelySparse:                                   (* indexing *)
     index: ARRAY [] OF ^tHashTable;
   eModeratelyDense:                                    (* (word-size truncated) recursion *)
     subset: ARRAY [] OF ^tSet;
   eVeryDense:                                          (* bit map *)
```

```
    bitmap: ARRAY [] OF BOOLEAN;
  END;
END;
```

ALGORITHM 3.2. MEMBERSHIP QUERY IF elt IS IN $\mathcal{N} \subseteq \mathcal{M}$, WHERE $|\mathcal{M}| = M$.

```
PROCEDURE Member (M, 𝒩, elt): BOOLEAN;
  IF M ≤ M₂ THEN
    pointer:= binomials[𝒩.ν] + 𝒩.β -1;                (* pointer by (3.10), *)
    RETURN LookUpBM (TSR[pointer], elt);             (* bit map from the TSR *)
  ELSE
    IF 𝒩.N ≥ M/2 THEN negate:= FALSE; N:= 𝒩.N
    ELSE negate:= TRUE; N:= M-𝒩.N END;
    CASE N OF
                                                     (* How sparse the set 𝒩 is: *)
    N = 0: answer:= FALSE                                    (* EMPTY SET; *)
    N ≤ c: answer:= FindOL (𝒩.list, elt);                (* VERY SPARSE SET; *)
    N ≤ M^(1-ε): answer:= FindHT (𝒩.hashTable, elt);   (* STILL VERY SPARSE SET;
*)
    N ≤ M/log_λ(M):                                      (* MODERATELY SPARSE SET: *)
      M₁:= Floor ((M/N)*lg(M));    (* split into buckets of range M₁ by (3.3), *)
      answer:= FindHT (𝒩.index[elt DIV M₁], elt MOD M₁);  (* search bucket; *)
    N ≤ α*M:                                             (* MODERATELY DENSE SET: *)
      M₁:= Floor (log_λ(M)*lg(M));          (* split into subuniverses of size M₁ by
(3.6), *)
      answer:=                                           (* and recursively search it; *)
        Member (M₁, 𝒩.subset[elt DIV M₁]^, elt MOD M₁)
                                                         (* VERY DENSE SET; *)
    ENDCASE;
    IF negate THEN RETURN NOT answer
    ELSE RETURN answer ENDIF;
  ENDIF;
END Member;
```

In analyzing the space requirements, we are interested only in moderately dense subsets, as otherwise we use the structures of sections 3.1 and 3.2. First we analyze the main structure, i.e., the data structure without a TSR, and begin with the following lemma.

LEMMA 3.2. *Suppose we are given a subset of N elements from the universe M, and B is as defined in* (2.1). *If this universe is split into p buckets of ranges of sizes M_i containing N_i elements, respectively (now, using* (2.1), $B_i = \lceil \lg \binom{M_i}{N_i} \rceil$ *for $1 < i \le p$), then $B + p > \sum_{i=1}^{p} B_i$.*

Proof. If $\sum_{i=1}^{p} M_i = M$ and $\sum_{i=1}^{p} N_i = N$, we know that $0 < \prod_{i=1}^{p} \binom{M_i}{N_i} \le \binom{M}{N}$ and therefore $\sum_{i=1}^{p} \lg \binom{M_i}{N_i} \le \lg \binom{M}{N}$. On the other hand, from (2.1) we have $B_i = \lceil \lg \binom{M_i}{N_i} \rceil$ and therefore $B_i - 1 < \lg \binom{M_i}{N_i} \le B_i$. This gives us $\sum_{i=1}^{p} (B_i - 1) < B$ and finally $B + p > \sum_{i=1}^{p} B_i$. □

In simpler terms, Lemma 3.2 proves that if subbuckets are encoded at close to the information-theoretic bound, then the complete bucket also uses an amount of space close to the information-theoretic minimum, provided that the number of buckets is small enough ($p = o(B)$) and that the index does not take too much space.

We analyze the main structure itself from the top to the bottom. The first-level index consists of p pointers of $\lg M$ bits each. Therefore, using (3.6) and (3.2), the size of that complete index is

$$p \lg M = M/\log_\lambda M = N_{sep} = o(B). \tag{3.11}$$

For the sparse buckets on the second level we use the solution presented in section 3.2. For the very dense buckets ($r \leq 1/\alpha$) we use a bitmap. Both of these structures guarantee space requirements within a constant factor of the information-theoretic bound on the number of bits. If the same also holds for the moderately dense buckets, then, using Lemma 3.2 and (3.11), the complete main structure uses $O(B)$ bits. Note that we can apply Lemma 3.2 freely because the number of buckets, p, is $o(B)$.

Next we determine the size of the encoding of the second-level moderately dense buckets, that is, the encoding of buckets with sparseness in the range of (3.5). For this purpose we first consider the size of bottom-level pointers (indices) into the TSR. As mentioned, the pointers are records consisting of two fields. The first field, ν (number of elements in the bucket), occupies $\lceil \lg M_2 \rceil$ bits, and the second field takes at most B_ν. Since $B_\nu \geq \lceil \lg M_2 \rceil$, the complete pointer[1] takes at most twice the information-theoretic bound on the number of bits, B_ν. On the other hand, the size of an index is bounded using an expression similar to (3.11). Subsequently, this, together with Lemma 3.2, also limits the size of space needed to store representation of moderately dense buckets on the second level to be within a constant factor of the information-theoretic bound. This, in turn, limits the size of the complete main structure to $O(B)$ bits.

It remains to compute the size of the TSR. There are 2^{M_2} entries in the table and each of the entries is M_2 bits wide. By (3.9) $M_2 = O((\lg^{(2)} M)^2)$. This gives us the total size of the table

$$\begin{aligned} M_2 2^{M_2} &= O((\lg^{(2)} M)^2 (\log M)^{\lg^{(2)} M}) \\ &= O((\lg \lg M \lg M)(\lg^{(2)} M (\lg M)^{1+\lg^{(2)} M})) \\ &= o(\lg r_{sep} M / r_{sep}) \qquad \text{by (3.1)} \\ &= o(N_{sep} \lg r_{sep}) = o(B). \end{aligned} \tag{3.12}$$

Finally, this also bounds the size of the whole structure to $O(B)$ bits and hence in conjunction with Lemma 3.1 proves the following theorem.

THEOREM 3.3. *There is an algorithm which solves the static membership problem in $O(1)$ time using a data structure of size $O(B)$ bits.*

Note the constants in order notation of Theorem 3.3 are relatively small. Algorithm 3.2 performs at most two recursive calls of `Member` and eight probes of the data structure:

- two probes in the first call of `Member`: one to get N and one to compute M_1;
- two probes in the second call of `Member`: same as above; and
- four probes in the last call of `Member`: two probes to get the number of elements in the bucket, ν, and the lexicographic order of the bucket, β; the next probe to get the sum in (3.10) by lookup in table `binomials`; and the final probe into the `TSR`.

[1] Note that the size of a pointer depends on the number of elements that fall into the bucket.

TABLE 3.2
Space usage for sets of primes and SINs for various data structures.

Example	M	N	B	ours	hash	bit map
Primes	$1.0 \cdot 2^{32}$	$1.4 \cdot 2^{27}$	$1.6 \cdot 2^{29}$	$1.9 \cdot 2^{30}$	$1.4 \cdot 2^{32}$	$1.0 \cdot 2^{32}$
SINs	$1.9 \cdot 2^{29}$	$1.7 \cdot 2^{24}$	$1.1 \cdot 2^{27}$	$1.2 \cdot 2^{28}$	$1.6 \cdot 2^{29}$	$1.9 \cdot 2^{29}$

If perfect hashing is used in one of the steps, the number of probes remains comparable.

It is easy to see that by setting $\alpha = 1/2$ and $\epsilon = 1$, thereby eliminating the two extreme cases, at most $2B + o(B)$ bits are required for the structure. In the next section we reduce this bound to $B + o(B)$ bits while retaining the constant query time. However, in practice the $o(B)$ term can be as much of a concern as the factor of 2. Indeed the reader of the next section is justified in questioning the notion of $(\lg^{(2)} M_1 - 5)/6$ becoming large in practice. We therefore first illustrate the tuning of the method to specific values of M and N with two examples.

The first is the set of primes that fit in a single 32-bit word, so $M = 2^{32}$ and N is of size approximately $M/\ln M$. We pretend that the set of primes is random and that we are to store them in a structure to support the query of whether a given number is prime. Clearly, we could use some kind of compression (e.g., implicitly omit the even numbers or sieve more carefully), but for the purpose of this example we will not do so. In the second example we consider Canadian Social Insurance Numbers (SINs) allocated to each individual. Canada has approximately 28 million people and each person has a nine-digit SIN. One may want to determine whether or not a given number is allocated. This query is in fact a membership query in the universe of size $M = 10^9$ with a subset of size $N = 28 \cdot 10^6$. One of the digits is a check digit, but we will ignore this issue.

Both examples deal with moderately sparse sets and we can use the method of section 3.2 directly, using buckets and a perfect hashing function described in [11]. On the other hand, no special features of the data are used, which makes our space calculations slightly pessimistic. Using an argument similar to that of Lemma 3.2, we observe that the worst-case distribution occurs when all buckets are equally sparse, and therefore we can assume that in each bucket there are N/p elements. Table 3.2 contains the sizes of data structures for both examples comparing a hash function, a bitmap, and a tuned version of our structure (computed from (3.4)) with the information-theoretic bound.

4. Static solution using $B + o(B)$ space. We now return to the tuning of our technique for asymptotically large sets, achieving a $B + o(B)$ bit space bound.

First, we observe that for very dense sets ($r \leq 1/\alpha$) we cannot afford to use a bitmap because it always takes $B + \Theta(B)$ bits. Similarly we cannot afford to use hash tables for very sparse sets (i.e., $r \geq M^{1-\epsilon}$). Therefore, we categorize sets *only* as sparse or dense (and not moderately dense). The key point in decreasing the space bound, however, is redefining the separation point between sparse and dense set to

$$r_{sep} = (\lg M)^{\lg^{(2)} M}, \tag{4.1}$$

and so

$$N_{sep} = M/(\lg M)^{\lg^{(2)} M}. \tag{4.2}$$

While we intend B to indicate the exact value from (2.1), for sparse sets we can

still use the approximation $N \lg r$ from (2.5) since the error in (2.4) is bounded by $\Theta(N) = o(B)$.

4.1. Sparse subsets. Again, sparse subsets are those whose relative sparseness is greater than r_{sep}. For such subsets we *always* apply the two-level indexing of section 3.2. All equations from section 3.2, and in particular (3.4), still hold. However, the second term of (3.4) can be tightened to $o(B)$, because now the relative sparseness, r, is at least r_{sep} defined in (4.1). This proves the following lemma.

LEMMA 4.1. *If $\infty > r \geq r_{sep}$ as in* (4.1) *(i.e., $N \leq N_{sep}$ as in* (4.2)*), then there is an algorithm to answer membership queries in constant time using a $B + o(B)$ bit data structure.*

4.2. Dense subsets. Dense subsets are treated in the same way as moderately dense subsets were treated in section 3.3. Thus most of the analysis can be taken from that section with the appropriate changes of r_{sep} (cf. (3.1)) and r'_{sep} (cf. (3.7)). To compute the size of the main structure, we first bound the size of pointers into the TSR. Recall that each pointer consists of two fields: the number of elements in the bucket, ν, and the rank (in lexicographic order) of the bucket in question among all buckets with ν elements, β. Although the number of bits needed to describe ν can be as large as the information-theoretic minimum for some buckets, this is not true on the average. By Lemma 3.2, all pointers together occupy no more than $B + o(B)$ bits, where B is the exact one from (2.1). Furthermore, the indices are small enough so that all of them together occupy $o(B)$ bits (cf. (3.11)). As a result we conclude that the main structure occupies $B + o(B)$ bits of space. It remains to bound the size of the TSR at the redefined separation points.

With the redefinition of r_{sep} in (4.1), (3.6) now gives

$$p = M/(r_{sep} \lg M) = M/(\lg M)^{1+\lg^{(2)} M} \tag{4.3}$$

buckets on the first level. Each of these has a range of

$$M_1 = M/p = r_{sep} \lg M = (\lg M)^{1+\lg^{(2)} M}. \tag{4.4}$$

To simplify further analysis we set the redefined separation point between first-level sparse and dense buckets (cf. (3.7)) to

$$r'_{sep} = (\lg M_1)^{(\lg^{(2)} M_1 - 5)/6}, \tag{4.5}$$

which is adequate to keep the space requirement of the sparse buckets to $o(B)$ (cf. (3.4)). The position of this separation point r'_{sep} is further bounded by

$$\begin{aligned}
r'_{sep} &= (\lg(r_{sep} \lg M))^{(\lg^{(2)}(r_{sep} \lg M)-5)/6} && \text{using (4.4)} \\
&< (2 \lg r_{sep})^{(\lg(2 \lg r_{sep})-5)/6} && \text{since } r_{sep} > \lg M \text{ by (4.1)} \\
&< (2(\lg^{(2)} M)^2)^{(\lg((\lg^{(2)} M)^2)-4)/6} && \text{again using (4.1)} \\
&< ((\lg^{(2)} M)^3)^{(} \lg^{(3)} M - 2)/3 && \text{since } 2 < \lg^{(2)} M \\
&< (\lg^{(2)} M)^{\lg^{(3)} M - 1}/3 && \text{since } (\lg^{(2)} M)^{-1} < 1/3. \qquad (4.6)
\end{aligned}$$

Next, the first-level dense buckets are further split into p_1 (cf. (3.8)) subbuckets, each of range $M_2 = M_1/p_1 = r'_{sep} \lg M_1$. Finally, since M_2 is also the range of buckets in the TSR, the size of the table is

$$
\begin{aligned}
M_2 2^{M_2} &= r'_{sep}(\lg M_1) M_1^{r'_{sep}} \\
&= r'_{sep} \lg(r_{sep} \lg M)(r_{sep} \lg M)^{r'_{sep}} && \text{by (4.4)} \\
&< 2(\lg r_{sep}) r'_{sep} {r_{sep}}^{2r'_{sep}} && \text{since } r_{sep} = (\lg M)^{\omega(1)} \\
&< (\lg r_{sep}) {r_{sep}}^{3r'_{sep}-1} \\
&< \lg r_{sep}((\lg M)^{\lg^{(2)} M})^{(\lg^{(2)} M)^{\lg^{(3)} M - 1}}/r_{sep} && \text{by (4.6) and (4.1)} \\
&< \lg r_{sep}(\lg M)^{(\lg^{(2)} M)^{\lg^{(3)} M}}/r_{sep} \\
&= o(M \lg r_{sep}/r_{sep}) = o(N_{sep} \lg r_{sep}) \\
&= o(B)
\end{aligned}
$$

for $r \le r_{sep}$. This brings us to the main asymptotic result.

THEOREM 4.2. *There is an algorithm which solves the static membership problem in* $O(1)$ *time using data structure of size* $B + o(B)$ *bits.*

Proof. The discussion above dealt only with the space bound. However, since the structure is more or less the same as that of section 3, the time bound can be drawn from Theorem 3.3. □

With Theorem 4.2 we proved only that the second term in space complexity is $o(B)$. In fact, using a very rough estimate from the second term of sparse first-level buckets we get the bound $O(B/\lg^{(3)} M)$. To improve the bound one would have to refine values r_{sep} and r'_{sep}.

5. Dynamic version. There are several options for converting our ideas for a static structure into one that supports insertions and deletions. In the interest of simplicity we sketch and demonstrate only one.

THEOREM 5.1. *There exists a data structure requiring* $O(B)$ *bits which supports searches in constant time and insertions and deletions in constant expected amortized time.*

The basic approach is simple, and we make no attempt here to minimize the space bound other than to retain the $O(B)$ requirement. We use the method of section 3 but substitute dynamic perfect hashing [5] for the static perfect hashing scheme used there.

A key observation is that, given a set of N elements, our structure can be built in expected time $O(N)$ and $O(B)$ space if we use dynamic perfect hashing for the hashing aspect. Indeed, any of our substructures can be built in linear time and in space within a constant factor of that suggested in the preceding section. This also applies to the TSR in that it can be created in time linear in its size.

Like dynamic perfect hashing itself, our dynamic scheme operates in phases. At the beginning of a phase, a structure of N_0 elements is built, but each dynamic perfect hashing structure is given $1 + \kappa$ times as much space as it requires. Here κ is an arbitrary positive constant. In addition the entire space allocated for the structure is increased globally by another factor of $1 + \kappa$.

As insertions and deletions are made, two critical conditions can arise. The number of elements in the table may drop, say to $N_0/(1 + \kappa)$, in which case we obtain a new block of space appropriate for a table of the new, reduced, size. A new table is constructed in the new space and the old table is released. The other condition is that we run out of space either in one of the hash tables or in the structure as a whole. It is unlikely that we will run out of space in a single dynamic perfect hashing structure until a number of updates proportional to its original size is made. However,

with our multilevel structure we could have a large number of insertions fall into the same bucket, which could cause a dynamic perfect hashing structure to overflow after a rather small number of updates relative to the size of the entire structure. If this happens, we simply rebuild this subtable using the extra space allocated globally.

6. Discussion and conclusions. In this paper we have presented a solution to a static membership problem. Our initial solution answers queries in constant time and uses space within a small constant factor of the minimum required by the information-theoretic lower bound. Subsequently, we improved the solution reducing the required amount of space to the information-theoretic lower bound plus a lower-order term. We also addressed the dynamic problem and proposed a solution based on a standard doubling technique.

Data structures used in solutions consist of three major substructures which are used in different ranges depending on the relative sparseness of the set at hand (that is, depending on the ratio between the size of the set and the universe). When the set is relatively sparse we use a perfect hashing; when the set is relatively dense we use a bitmap; and in the range between we use recursive splitting (indexing). The depth of the recursion is bounded by the use of *word-size truncation* and in our case it is 2.

The feasibility of the data structure was addressed through a couple of examples. However, to make the structure more practical one would need to tune the parameters c, ϵ, λ, and α mentioned in Algorithm 3.2. Moreover, for practical purposes it is helpful to increase the depth of recursive splitting to cancel out the effect of a constant hidden in the order notation and, in particular, to decrease the size of the TSR below the information-theoretic minimum defined by N and M at hand. For example, in the case of currently common 64- and 32-bit architectures, the depths can be increased to 4 and 5, respectively.

There are several open problems. One may be able to reduce the space requirement of the dynamic structure to $B+o(B)$ by first reexamining the details of dynamic perfect hashing and reducing its storage requirements to $N + o(N)$ words, assuming the universe is large relative to N. Another intriguing problem is to decrease the second-order term in the space complexity as there is still a substantial gap between our result, $B + O(B/\lg^{(3)} M)$, and the information-theoretic minimum, B. But do we need a more powerful machine model to close this gap?

Acknowledgments. We thank Martin Dietzfelbinger and the referees for many very helpful comments that improved this paper.

REFERENCES

[1] A. Brodnik, *Searching in Constant Time and Minimum Space* (*Minimæ Res Magni Momenti Sunt*), Ph.D. thesis, available as Technical report CS-95-41, University of Waterloo, Waterloo, ON, Canada, 1995.

[2] A. Brodnik and J. I. Munro, *Membership in constant time and minimum space*, in Proceedings, Second European Symposium on Algorithms, Lecture Notes in Comput. Sci. 855, Springer-Verlag, 1994, pp. 72–81.

[3] Y. Choueka, A. Fraenkel, S. Klein, and E. Segal, *Improved hierarchical bit-vector compression in document retrieval systems*, in 9th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 1986, pp. 88–96.

[4] M. Dietzfelbinger, J. Gil, Y. Matias, and N. Pippenger, *Polynomial hash functions are reliable*, in Proceedings, 19th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 623, Springer-Verlag, 1992, pp. 235–246.

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

[6] M. Dietzfelbinger and F. Meyer auf der Heide, *A new universal class of hash functions and dynamic hashing in real time*, in Proceedings, 17th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 443, Springer-Verlag, New York, 1990, pp. 6–19.

[7] P. Elias, *Efficient storage retrieval by content and address of static files*, J. ACM, 21 (1974), pp. 246–260.

[8] P. Elias and R. Flower, *The complexity of some simple retrieval problems*, J. ACM, 22 (1975), pp. 367–379.

[9] P. van Emde Boas, *Machine models and simulations*, in Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 1–66.

[10] A. Fiat and M. Naor, *Implicit $O(1)$ probe search*, SIAM J. Comput., 22 (1993), pp. 1–10.

[11] A. Fiat, M. Naor, J. Schmidt, and A. Siegel, *Nonoblivious hashing*, J. ACM, 39 (1992), pp. 764–782.

[12] F. Fich and P. Miltersen, *Tables should be sorted (on random access machines)*, in Proceedings, Fourth Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 955, Springer-Verlag, New York, 1995, pp. 482–493.

[13] M. Fredman, J. Komlós, and E. Szemerédi, *Storing a sparse table with $O(1)$ worst case access time*, J. ACM, 31 (1984), pp. 538–544.

[14] M. Fredman and D. Willard., *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. ACM, 31 (1984), pp. 538–544.

[15] H. Gabow and R. Tarjan, *A linear-time algorithm for a special case of disjoint set union*, J. Comput. System Sci. 30 (1985), pp. 209–221.

[16] T. Hagerup, K. Mehlhorn, and J. I. Munro, *Optimal algorithms for generating discrete random variables with changing distributions*, in Proceedings, 20th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 700, Springer-Verlag, New York, 1993, pp. 253–264.

[17] H. Mairson, *The program complexity of searching a table*, in 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 40–47.

[18] K. Mehlhorn and A. Tsakalidis, *Data structures*, in Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity, J. van Leeuwen, ed., Elsevier, Amsterdam, The Netherlands, 1990, pp. 301–334.

[19] P. Miltersen, *The bit probe complexity measure revisited*, in Proceedings, 10th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 665, Springer-Verlag, New York, 1993, pp. 662–671.

[20] D. Mitrinović, *Analytic Inequalities*, Grundlehren Math. Wiss. 165, Springer-Verlag, Berlin, 1970.

[21] J. I. Munro and H. Suwanda, *Implicit data structures for fast retrieval and update*, J. Comput. System Sci., 21 (1980), pp. 236–250.

[22] J. Schmidt and A. Siegel, *The spatial complexity of oblivious k-probe hash functions*, SIAM J. Comput., 19 (1990), pp. 775–786.

[23] R. Tarjan and A. Yao, *Storing a sparse table*, Comm. ACM, 22 (1979), pp. 606–611.

[24] A.-C. Yao, *Should tables be sorted?*, J. ACM, 28 (1981), pp. 614–628.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1641–1663

DERANDOMIZING APPROXIMATION ALGORITHMS BASED ON SEMIDEFINITE PROGRAMMING*

SANJEEV MAHAJAN† AND H. RAMESH‡

Abstract. Remarkable breakthroughs have been made recently in obtaining approximate solutions to some fundamental NP-hard problems, namely Max-Cut, Max k-Cut, Max-Sat, Max-Dicut, Max-bisection, k-vertex coloring, maximum independent set, etc. All these breakthroughs involve polynomial time *randomized* algorithms based upon *semidefinite programming*, a technique pioneered by Goemans and Williamson.

In this paper, we give techniques to derandomize the above class of randomized algorithms, thus obtaining polynomial time deterministic algorithms with the same approximation ratios for the above problems. At the heart of our technique is the use of spherical symmetry to convert a nested sequence of n integrations, which cannot be approximated sufficiently well in polynomial time, to a nested sequence of just a constant number of integrations, which can be approximated sufficiently well in polynomial time.

Key words. NP-hard, approximation algorithm, derandomization, semidefinite programming

AMS subject classification. 68Q25

PII. S0097539796309326

1. Introduction.

The application of *semidefinite programming* to obtaining approximation algorithms for NP-hard problems was pioneered by Goemans and Williamson [9]. This technique involves relaxing an integer program (solving which is an NP-hard problem) to a semidefinite program (which can be solved with a sufficiently small error in polynomial time).

Recall the Max-Cut problem which requires partitioning the vertex set of a given graph into two so that the number of edges going from one side of the partition to the other is maximized. In a remarkable breakthrough, Goemans and Williamson showed how semidefinite programming could be used to give a *randomized* approximation algorithm for the Max-Cut problem with an approximation ratio of .878. This must be contrasted with the previously best known approximation ratio of .5 obtained by the simple random cut algorithm. Subsequently, techniques based upon semidefinite programming have led to randomized algorithms with substantially better approximation ratios for a number of fundamental problems.

Goemans and Williamson [9] obtained a .878 approximation algorithm for Max-2Sat and a .758 approximation algorithm for Max-Sat, improving upon the previously best known bound of 3/4 for both [18]. Max-2Sat requires finding an assignment to the variables of a given 2Sat formula which satisfies the maximum number of clauses. Max-Sat is the general version of this problem, where the clauses are no longer constrained to have two variables each. Goemans and Williamson [9] also obtained a .796 approximation algorithm for Max-Dicut, improving the previously best known ratio of .25 given by the random cut algorithm. This problem requires partitioning the vertex set of a given directed graph into two so that the number of edges going from the left side of the partition to the right is maximized. Feige and

*Received by the editors September 11, 1996; accepted for publication (in revised form) April 15, 1998; published electronically May 6, 1999. Part of this work was done while both authors were at Max Planck Institut für Informatik, Saarbrücken, Germany, 66123.

http://www.siam.org/journals/sicomp/28-5/30932.html

†LSI Logic, Milpitas, CA (msanjeev@lsil.com).

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

Goemans [6] obtained further improved approximation algorithms for Max-2Sat and Max-Dicut.

Karger, Motwani, and Sudan obtained an algorithm for coloring any k-colorable graph with $O(n^{1-3/(k+1)} \log n)$ colors [12]; in particular, for 3-colorable graphs, this algorithm requires $O(n^{.25} \log n)$ colors. This improves upon the deterministic algorithm of Blum [3], which requires $O(n^{1-\frac{1}{k-4/3}} \log^{\frac{8}{5}} n)$ colors for k-colorable graphs.

Frieze and Jerrum [7] obtained a .65 approximation algorithm for Max-bisection improving the previous best known bound of .5 given by the random bisection algorithm. This problem requires partitioning the vertex set of a given graph into two parts of roughly equal size such that the number of edges going from one side of the partition to the other is maximized. They also obtained a $1-\frac{1}{k}+2\frac{\ln k}{k^2}$ approximation algorithm for the Max k-Cut problem, improving upon the previously best known ratio of $1-\frac{1}{k}$ given by a random k-Cut.

Alon and Kahale [1] obtained an approximation algorithm for the maximum independent set problem on graphs, which requires finding the largest subset of vertices, no two of which are connected by an edge. For any constant $k \geq 3$, if the given graph has an independent set of size $n/k+m$, where n is the number of vertices, they obtain an $\Omega(m^{\frac{3}{k+1}} \log m)$-sized independent set, improving the previously known bound of $\Omega(m^{\frac{1}{k-1}})$ due to Boppana and Halldorsson [4].

All the new developments mentioned above are *randomized* algorithms. All of them share the following common paradigm. First, a semidefinite program is solved to obtain a collection of n vectors in n-dimensional space satisfying some properties dependent upon the particular problem in question. This step is deterministic. (In the Feige and Goemans paper [6], there is another intermediate step of generating a new set of vectors from the vectors obtained above.) Second, a set of independent random vectors is generated, each vector being *spherically symmetric*, i.e., equally likely to pass through any point on the n-dimensional unit sphere centered at the origin. Finally, the solution is obtained using some computation on the n given vectors and the random vectors.

It is not obvious how to derandomize the above randomized algorithms, i.e., to obtain a "good" set of random vectors deterministically. A natural way to derandomize is to use the method of conditional probabilities [14, 16]. The problem that occurs then is to compute the conditional probabilities in polynomial time.

Our contribution. The main contribution of this paper is a technique which enables derandomization of all approximation algorithms based upon semidefinite programming listed above. This leads to deterministic approximation algorithms for Max-Cut, Max k-Cut, Max-bisection, Max-2Sat, Max-Sat, Max-Dicut, k-vertex coloring, and maximum independent set with the same approximation ratios as their randomized counterparts mentioned above. However, we must mention that running times of our deterministic algorithms, though polynomial, are quite slow, for example, $O(n^{30})$ or so for 3-vertex coloring. In this paper, we do not make an effort to pinpoint the exact polynomial or to reduce the running time (within the realm of polynomials, that is).

Our derandomization uses the conditional probability technique. We compute conditional probabilities as follows. First, we show how to express each conditional probability computation as a sequence of $O(n)$ nested integrals. Performing this sequence of integrations with a small enough error seems hard to do in polynomial time. The key observation which facilitates conditional probability computation in

polynomial time is that, using spherical symmetry properties, the above sequence of $O(n)$ nested integrals can be reduced to evaluating an expression with just a constant number of nested integrals for each of the approximation algorithms mentioned above. This new sequence of integrations can be performed with a small enough error in polynomial time. A host of precision issues also crops up in the derandomization. Conditional probabilities must be computed only at a polynomial number of points. Further, each conditional probability computation must be performed within a small error. We show how to handle these precision issues in polynomial time.

As mentioned above, our derandomization techniques apply to all the semidefinite programming based approximation algorithms mentioned above. Loosely speaking, we believe our techniques are even more general, i.e., applicable to any scheme which follows the above paradigm and in which the critical performance analysis boils down to an "elementary event" involving just a constant number of the n vectors at a time. For example, in the graph coloring algorithm, only two vectors, corresponding to the endpoints of some edge, need to be considered at a time. An example of an elementary event involving three vectors is the Max-Dicut algorithm of Goemans and Williamson. Another example of the same is the algorithm of Alon et al. [2] for coloring 2-colorable 3-uniform hypergraphs approximately.

The paper is organized as follows. In section 2, we outline the Goemans and Williamson Max-Cut algorithm and the Karger–Motwani–Sudan coloring algorithm. We then describe our derandomization scheme. Since the Karger–Motwani–Sudan coloring algorithm appears to be the hardest to derandomize amongst the algorithms mentioned above, our exposition concentrates on this algorithm. The derandomization of the other algorithms is similar. Section 3 describes the derandomization procedure. The following sections describe the derandomization procedure in detail.

2. The semidefinite programming paradigm. It is known that any concave polynomial time computable function can be maximized (within some tolerance) over a convex set with *a weak separation oracle* in polynomial time [8]. A weak separation oracle (see [8, p. 51],) is one which, given a point y, either asserts that y is in or close to the convex set in question or produces a hyperplane which "almost" separates all points well within the convex set from y.

One such convex set is the set of *semidefinite matrices*, i.e., those matrices whose eigenvalues are all nonnegative. A set formed by the intersections of half-spaces and the set of semidefinite matrices is also a convex set. Further, this convex set admits a weak separation oracle. A semidefinite program involves maximizing a polynomial time computable concave function over one such convex set. Semidefinite programs are therefore solvable (up to an additive error exponentially small in the input length) in polynomial time. Goemans and Williamson first used this fact to obtain an approximation algorithm for Max-Cut.

The Goemans–Williamson Max-Cut algorithm. Goemans and Williamson took a natural integer program for Max-Cut and showed how to relax it to a semidefinite program. The solution to this program is a set of n unit vectors, one corresponding to each vertex of the graph in question. These vectors emanate from the origin. We call these vectors *vertex vectors*. These are embedded in n-dimensional space. This leads to the question as to how a large cut is obtained from these vectors.

Goemans and Williamson chose a random hyperplane through the origin whose normal is spherically symmetrically distributed; this hyperplane divides the vertex vectors into two groups, which define a cut in the obvious manner. The expected

number $\mathbf{E}(W)$ of edges[1] across the cut is $\sum_{(v,w)\in E} \Pr(sign(v \cdot R) \neq sign(w \cdot R)) = \sum_{(v,w)\in E} \arccos(v \cdot w)/\pi$, where E is the set of edges in the graph and v, w denote both vertices in the graph and the associated vertex vectors. Goemans and Williamson show that $\mathbf{E}(W)$ is at least .878 times the maximum cut.

Note that the n random variables involved above are the n coordinates which define the normal R to the random hyperplane. Let $R_1, R_2, \ldots, R_n$ be these random variables. For R to be spherically symmetrically distributed, it suffices that the R_i's are independent and identically distributed with a mean 0 and variance 1 normal distribution; i.e., the density function is $\frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ [5]. Derandomizing the above algorithm thus requires obtaining values for $R_1, \ldots, R_n$ deterministically so that the value of the cut given by the corresponding hyperplane is at least $\mathbf{E}(W)$.

The Goemans–Williamson derandomization procedure. Goemans and Williamson actually gave the following derandomization procedure for their algorithm, which turned out to have a subtle bug described below.

From the initial set of vertex vectors in n dimensions, they obtain a new set of vertex vectors in $n-1$ dimensions satisfying the property that the expected size of the cut obtained by partitioning the new vertex vectors with a random hyperplane in $n-1$ dimensions is at least $\mathbf{E}(W)$. This procedure is repeated until the number of dimensions is down to 1, at which point partitioning the vectors becomes trivial. It remains to show how the new vertex vectors in $n-1$ dimensions are obtained from the older vertex vectors.

Consider an edge v, w, with vertex vector $v = (v_1, \ldots, v_{n-1}, v_n)$ and vertex vector $w = (w_1, \ldots, w_{n-1}, w_n)$. Recall that $R = (R_1, \ldots, R_n)$ is the normal to the random hyperplane. Goemans and Williamson obtain v', w', R' from v, w, R as follows: $R' = (R_1, \ldots, R_{n-2}, sign(R_{n-1})\sqrt{R_{n-1}^2 + R_n^2})$, $v' = (v_1, \ldots, v_{n-2}, x\cos(\alpha - \gamma))$, and $w' = (w_1, \ldots, w_{n-2}, y\cos(\beta - \gamma))$, where $\gamma = \arctan(R_n/R_{n-1})$, $\alpha = \arctan(v_n/v_{n-1})$, $\beta = \arctan(w_n/w_{n-1})$, $x = sign(v_{n-1})\sqrt{v_{n-1}^2 + v_n^2}$, and $y = sign(w_{n-1})\sqrt{w_{n-1}^2 + w_n^2}$.

These new definitions have the property that $v' \cdot R' = v \cdot R$ and $w' \cdot R' = w \cdot R$. To obtain the new vectors, one needs to decide on the value of γ. By the above property, there exists a value of γ such that $\sum_{(v,w)\in E} \Pr(sign(v' \cdot R') \neq sign(w' \cdot R') | \gamma) \geq \mathbf{E}(W)$. This value of γ is found by computing $\sum_{(v,w)\in E} \Pr(sign(v' \cdot R') \neq sign(w' \cdot R') | \gamma)$ for a polynomial number of points in the suitably discretized interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$. At this point, Goemans and William claim that the vector R' is spherically symmetric for any fixed value of γ, and therefore $\Pr(sign(v' \cdot R') \neq sign(w' \cdot R') | \gamma) = (\frac{v' \cdot w'}{|v'||w'|})/\pi$ and is thus easy to compute.

The flaw lies in the fact that $r = sign(R_{n-1})\sqrt{R_{n-1}^2 + R_n^2}$ is not distributed normally with mean 0 and variance 1, even for a fixed value of γ. In fact, given γ, it can be shown to be distributed according to the density function $\frac{|r|}{2} e^{-r^2/2}$. It is not clear how $\Pr(sign(v' \cdot R') \neq sign(w' \cdot R') | \gamma)$ can be computed for this distribution of R'.

The Karger–Motwani–Sudan coloring algorithm. Our description of this algorithm is based on the conference proceedings version of their paper [12].

The Karger–Motwani–Sudan algorithm shows how to color a 3-colorable graph of n vertices with $O(n^{1/4} \log n)$ colors. The authors use a semidefinite program to obtain a set of vertex vectors such that $v \cdot w \leq -\frac{1}{2}$ for all edges (v, w). Note that

[1] For simplicity, we consider the unweighted Max-Cut problem.

if these vectors are somehow constrained to be in two dimensions, then there are at most three distinct vectors, which would specify a 3-coloring. However, the output to the semidefinite program are vectors in an n-dimensional space. It remains to be described how a coloring is obtained from these vectors. This is done as follows.

Karger, Motwani, and Sudan choose r vectors, $t_1, \ldots, t_r$, independently and at random; each is spherically symmetric. These vectors are called *centers*. The number of centers r will be spelled out in a few paragraphs. Let the jth coordinate of t_i be denoted by $t_i[j]$, $1 \leq j \leq n$. Spherical symmetry is obtained by the following procedure: each $t_i[j]$ is chosen independently at random from a normal distribution with mean 0 and variance 1. The color that vertex v gets is simply c, where $t_c \cdot v = \max_{1 \leq i \leq r} t_i \cdot v$. In other words, the color assigned to a vertex v corresponds to that amongst the r centers which has the largest projection on the vector v. Ties in choosing the vector with the largest projection occur with probability 0 and can therefore be ignored.

To determine how good the above procedure is, it is necessary to determine the probability that an edge is *bad*; i.e., both its endpoints get the same color. Consider two vertex vectors v, w, such that (v, w) is an edge e in $G = (V, E)$. The probability that v and w get the same color in the algorithm is given by $\Pr(E^e) = \sum_{k=1}^{r} \Pr(E_k^e)$, where E_k^e is the event that both get color t_k. E_k^e can be written as

$$\mathrm{E}_k^e : t_k \cdot v = \max\{t_1 \cdot v, \ldots, t_r \cdot v\} \wedge t_k \cdot w = \max\{t_1 \cdot w, \ldots, t_r \cdot w\}.$$

Karger, Motwani, and Sudan [12] show the following theorem (see Theorem 7.7, Corollary 7.8, and Lemma 7.9 of [12]).

THEOREM 2.1. *For $r = O(d^{1/3} \log^{4/3} d)$, if edge $e = (v, w)$ satisfies $v \cdot w \leq -1/2 + O(1/\log r)$, then $\Pr(E^e) = O(\frac{1}{d})$, where d is the maximum degree of the graph. Thus $\sum_{e \in E} \Pr(E^e)$, i.e., the expected number of bad edges, can be made less than $n/4$ by an appropriate choice of constants.*

Thus, at the end of the above procedure, the expected number of bad edges is less than $n/4$. All vertices except those upon which these bad edges are incident are discarded (the colors assigned to them are final). The expected number of remaining vertices is at most $n/2$. Markov's inequality helps to bound their number with a reasonable probability. These vertices are recolored by repeating the above procedure $O(\log n)$ times, using a fresh set of colors each time. This gives an $O(d^{1/3} \log^{4/3} d \log n)$ coloring of a 3-colorable graph. This, combined with a technique due to Wigderson [17], gives an $O(n^{1/4} \log n)$ coloring of a 3-colorable graph.

Derandomizing the above algorithm entails deterministically obtaining values for $t_i[j]$'s so that the number of bad edges is at most the expected number of bad edges above, i.e., $n/4$. Actually it suffices to obtain values for $t_i[j]$'s such that the number of bad edges is at most $n/4 + O(1)$. This is what we will do.

Note that the Goemans–Williamson algorithm uses a random hyperplane while the Karger–Motwani–Sudan algorithm uses a set of random centers. Although these two methods seem different, the hyperplane method can be interpreted as just the center method with two centers.

3. The derandomization scheme. We give an overview of our derandomization scheme in this section. For simplicity, we restrict our exposition here to the derandomization of the Karger–Motwani–Sudan algorithm for coloring 3-colorable graphs. Our procedure easily generalizes to all other known semidefinite programming based approximation algorithms listed in section 1.

Notation. For a vector u, we denote by $u[l \dots m]$ the vector formed by the lth to mth coordinates of u.

Our derandomization procedure has two steps. The first step described in section 3.1 is a discretization step. It is necessary for reasons which will become clear later. This step obtains a new set of vertex vectors which are "close" to the initial vertex vectors. The new vertex vectors satisfy the property that the Karger–Motwani–Sudan randomized algorithm continues to give the claimed theoretical performance on these vectors as well. This justifies the use of the new vectors in the actual derandomization process, which is the second step and is described in section 3.2.

3.1. Preprocessing vertex vectors: Discretization. Before we describe our derandomization scheme, we discretize the vertex vectors obtained from the semidefinite program so as to satisfy the following properties. Let ϵ be a parameter which is $\Theta(\frac{1}{n^2})$.

1. Each component of each vector is at least inverse polynomial (more precisely, $\Omega(\epsilon)$) in absolute value.
2. The dot product of any pair of vectors changes only by an inverse polynomial (more precisely, $O(n\epsilon)$) in absolute value.
3. For each pair of vectors v, w and every h, $1 \leq h < n$, when the coordinate system is rotated so that $v[h \dots n] = (b_1, 0, \dots, 0)$ and $w[h \dots n] = (b_1', b_2', 0, \dots, 0)$, b_1 and b_2' are at least some inverse polynomial (more precisely, $\Omega(\epsilon)$ and $\Omega(\epsilon^2)$, respectively) in absolute value.

The method for performing the discretization is given in Appendix 1. The purpose of the above discretization will become clear in the next subsection. Of course, we have to show that the above discretization does not cause much error. This is true because $v \cdot w$, which was at most $-\frac{1}{2}$ before the discretization, is at most $-\frac{1}{2} + O(\frac{1}{n})$ now for each edge $e = (v, w)$. Then, by Theorem 2.1, the theoretical bounds of the Karger–Motwani–Sudan randomized algorithm continue to hold for discretized vectors as well.

From now on, all our references to vectors will be to the discretized vectors.

3.2. Outline of the derandomization procedure. The scheme is essentially to use the method of conditional expectations to deterministically find values for the vectors $t_1, \dots, t_r$ so that the number of bad edges is just $n/4 + O(1)$.

We order the conditional variables as follows: $t_1[1] \dots t_1[n], t_2[1] \dots t_2[n], \dots, t_r[1] \dots t_r[n]$. The values of these are fixed one by one, in order. So suppose that the values $t_1[1 \cdots n], t_2[1 \cdots n], \dots, t_i[1 \cdots j-1]$ have been determined. We will show how a value for $t_i[j]$ is determined.

Notation. Let $\mathcal{E}$ be an event. Then $\Pr(\mathcal{E}|i, j, \delta)$ denotes the probability that the event $\mathcal{E}$ occurs when the values for all conditional variables before $t_i[j]$ have been fixed as above and $t_i[j]$ itself is assigned value δ. So, for example, $\Pr(E_k^e|i, j, \delta)$ denotes the probability that event E_k^e occurs (i.e., that both endpoints of edge e get the color associated with center t_k) when the values for all conditional variables before $t_i[j]$ have been fixed as above and $t_i[j]$ itself is assigned value δ. For notational brevity, sometimes we use $f_{e,k}(\delta)$ to denote $\Pr(E_e^k|i, j, \delta)$.

Let $p(\delta)$ be the expected number of bad edges when the values for all conditional variables before $t_i[j]$ are fixed as above and $t_i[j]$ is assigned value δ; $p(\delta) = \sum_{e \in E} \sum_{k=1}^{r} f_{e,k}(\delta)$.

Note that both $f_{e,k}(\delta)$ and $p(\delta)$ implicitly refer to some fixed values of i and j. This will be the case throughout this paper.

Problem to be solved now. The problem now is to find a value of δ for which $p(\delta) \leq \sum_{e \in E} \sum_{k=1}^{r} \Pr(E_e^k | t_1[1 \ldots n], t_2[1 \ldots n], \ldots, t_i[1 \ldots j-1])$. In other words, we want a value of δ with the following property: the expected number of bad edges with $t_i[j]$ assigned δ and $t_1[1 \ldots n], t_2[1 \ldots n], \ldots, t_i[1 \ldots j-1]$ fixed as above, is at most the expected number of bad edges with just $t_1[1 \ldots n], t_2[1 \ldots n], \ldots, t_i[1 \ldots j-1]$ fixed as above.

Fixing $t_i[j]$. Let $\tau = \sum_{e \in E} \sum_{k=1}^{r} \Pr(E_e^k | t_1[1 \ldots n], t_2[1 \ldots n], \ldots, t_i[1 \ldots j-1])$. We want to compute a value δ such that $p(\delta) \leq \tau$. We will not be able to compute such a δ. However, we will show the following.

In Theorem 3.1, we claim that working with the discretized vertex vectors, we can compute a value κ, such that $p(\kappa)$ is within $O(1/n^2)$ of τ. Corollary 3.2 then shows that this suffices to obtain the required bound on the number of colors.

THEOREM 3.1. *A value κ for $t_i[j]$ satisfying the following property can be computed in polynomial time: $p(\kappa) \leq \tau + O(1/n^2)$.*

From the above theorem, we get the following corollary.

COROLLARY 3.2. *After all $t_i[j]$'s have been fixed and colors assigned to vertices as in the randomized algorithm, the number of bad edges is at most $n/4 + O(1)$.*

Proof. Note that the number of conditional variables $t_i[j]$ is $nr \leq n^2$ (actually for 3-colorable graphs r is much smaller, namely $d^{1/3} \log^{4/3} d$, where d is the maximum degree).

Recall that the expected number of bad edges before any of the random variables was fixed is at most $n/4$. By Theorem 3.1, the expected number of bad edges after the first conditional variable is fixed is at most $n/4 + O(1/n^2)$. An easy inductive argument shows that the expected number of bad edges after the lth conditional variable is fixed is at most $n/4 + O(1/n^2)$. After all the $nr \leq n^2$ conditional variables have been fixed, the expected number of bad edges (which is just the number of bad edges since all conditional variables are now fixed) is at most $n/4 + O(1)$. ∎

Note that while the Karger–Motwani–Sudan algorithm ensures that the number of bad edges is less than $n/4$, our deterministic algorithm shows a slightly weaker bound, i.e., at most $n/4 + O(1)$. However, it can be seen easily that this weaker bound on the number of bad edges also suffices to obtain the bound of $O(n^{1/4} \log n)$ colors for coloring a 3-colorable graph deterministically.

The rest of this paper will be aimed at proving Theorem 3.1. This is accomplished by performing the following steps, each of which is elaborated in detail in the following sections.

Remark. The purpose of discretizing the vertex vectors earlier can be explained now. The intention is to ensure that derivatives of the functions $f_{e,k}(\delta)$ and $p(\delta)$ (with respect to δ) are bounded by a polynomial in n. This, in turn, ensures that the values of the above functions between any two nearby points will not be too different from their values at these two points, thus facilitating discrete evaluation, which we will need to do.

3.3. Steps required to prove Theorem 3.1. The following steps are performed in our algorithm to obtain the value κ described above. Recall again that we are working with fixed values of i, j and assuming that $t_1[1 \ldots n], t_2[1 \ldots n], \ldots, t_i[1 \ldots j-1]$ have already been fixed.

Step 1. In order to compute κ, we would like to evaluate $p(\delta)$ at a number of points. However, we can afford to evaluate $p(\delta)$ only for a polynomial number of points. In section 4, we show how to obtain a set S of polynomial size such that

$\min_{\delta \in S} p(\delta) - \tau = O(\frac{1}{n^2})$. Therefore, in order to compute κ, it suffices to evaluate $p(\delta)$ at points in S, as long as the evaluation at each point in S is correct to within an additive $O(\frac{1}{n^2})$ error.

Step 2. We now need to show how $p(\delta)$ can be evaluated within an additive $O(\frac{1}{n^2})$ error for any particular point δ in polynomial time. Of course, we need to do this computation for points in S only, but the description of this step is for any general point δ.

To compute $p(\delta)$, we need to compute $f_{e,k}(\delta)$ to within an $O(\frac{1}{n^5})$ additive error, for each edge e and each center k (using the rather liberal upper bounds of $O(n^2)$ for the number of edges and $O(n)$ for the number of centers). We will describe this computation for a particular edge e and a particular center k. This will involve two substeps. The first substep will develop an expression involving a nested sequence of integrations for $f_{e,k}(\delta)$, and the second substep will actually evaluate this expression within the required error.

Substep 2a. We show how to write $f_{e,k}(\delta)$ as an expression involving a nested sequence of integrations of constant depth. This is done is two stages. In the first stage, in section 5, $f_{e,k}(\delta)$ is expressed as a nested sequence of integrations of constant depth, with the integrand comprising only basic functions and a function I, defined below. Subsequently, in section 6, we express the function I itself as an expression involving a constant number of nested integrations. This requires the use of spherical symmetry properties. The fact that the depth of integration is a constant will be significant in Substep 2b. If this were not the case, then it is not clear how $f_{e,k}(\delta)$ could be computed within the required error bounds at any point δ in polynomial time.

DEFINITION 3.3. *Let b, b' be vectors of the same dimension, which is at least* 2. *Let a be another vector of the same dimension whose entries are independent and normally distributed with mean* 0 *and variance* 1. *Let $x \leq y$ and $x' \leq y'$ be in the range $-\infty \ldots \infty$. Then $I(b, b', x, y, x', y')$ denotes* $\Pr((x \leq a \cdot b \leq y) \wedge (x' \leq a \cdot b' \leq y'))$.

Substep 2b. The expression for $f_{e,k}(\delta)$ obtained in Substep 2a is evaluated to within an $O(\frac{1}{n^5})$ additive error in this step in polynomial time. This is described in section 7.

Remarks on $\frac{\mathrm{d} f_{e,k}(\delta)}{\mathrm{d}\delta}$. We will also be concerned with the differentiability of $p(\delta)$ and therefore $f_{e,k}(\delta)$ for each edge e and each center k. From the expressions derived for $f_{e,k}(\delta)$ in section 5, the following properties about the differentiability of $f_{e,k}(\delta)$ will become clear. These properties will be used in section 4, i.e., to obtain the set S as described in Step 1. We state these properties in a lemma for future reference. The proof of this lemma in section 5 will not rely on any usage of the lemma in section 4.

LEMMA 3.4. *When $j < n-1$, $f_{e,k}(\delta)$ is differentiable (with respect to δ) for all δ. When $j = n-1$ or $j = n$, $f_{e,k}(\delta)$ is differentiable for all but at most* 2 *values of δ.*

4. Step 1: Determining set *S*. We show how to obtain a set S of polynomial size such that $\min_{\delta \in S} p(\delta) - \tau = O(\frac{1}{n^2})$. Recall from section 3.2 that $p(\delta) = \sum_{e \in E} \sum_{k=1}^{r} f_{e,k}(\delta) = \sum_{e \in E} \sum_{k=1}^{r} \Pr(E_e^k | i, j, \delta)$.

We will use the following theorem from [5, Chapter 7, Lemma 2].

THEOREM 4.1. *For every $a > 0$, $\int_a^\infty \mathrm{e}^{\frac{-\delta^2}{2}} \mathrm{d}\delta \leq \frac{1}{a} \mathrm{e}^{-\frac{a^2}{2}}$.*

First, we show that we can restrict δ to the range $-3\sqrt{\ln n} \ldots 3\sqrt{\ln n}$.

LEMMA 4.2. $\min_{-3\sqrt{\ln n} < \delta < 3\sqrt{\ln n}} p(\delta) - \tau = O(\frac{1}{n^2})$.

Proof. Note that $\tau = \sum_{e \in E} \sum_{k=1}^{r} \Pr(E_k^e | t_1[1] \ldots t_i[j-1]) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} p(\delta) \mathrm{e}^{-\frac{\delta^2}{2}} \mathrm{d}\delta$.

Let δ_{min} be the point in the above range at which $p(\delta)$ is minimized in this range. Then

$$p(\delta_{min}) \leq \frac{\int_{-3\sqrt{\ln n}}^{3\sqrt{\ln n}} p(\delta) \mathrm{e}^{-\frac{\delta^2}{2}} \mathrm{d}\delta}{\int_{-3\sqrt{\ln n}}^{3\sqrt{\ln n}} \mathrm{e}^{-\frac{\delta^2}{2}} \mathrm{d}\delta} \leq \frac{\tau}{(1 - 2\int_{3\sqrt{\ln n}}^{\infty} \mathrm{e}^{-\frac{\delta^2}{2}} \mathrm{d}\delta)} \leq \frac{\tau}{1 - O(\frac{1}{n^{4.5}})}$$

by Theorem 4.1. Therefore, $p(\delta_{min}) \leq \tau(1 + O(\frac{1}{n^{4.5}})) \leq \tau + O(\frac{1}{n^{2.5}})$ as required. □

The set S we choose will comprise points which are multiples of $\Theta(1/n^9)$ in the range $-3\sqrt{\ln n} \ldots 3\sqrt{\ln n}$. In addition, it will contain all those points in the above range at which $p(\delta)$ is not differentiable. By Lemma 3.4, there will be at most $2r|E| = O(n^3)$ such points, at most 2 for each edge-center pair. These points will be obtained in the process of writing down the expressions for $f_{e,k}(\delta)$ in section 5. The size of S is thus $O(n^9\sqrt{\ln n})$.

We need to show that evaluating $p(\delta)$ at just points in S suffices to approximate $\min_{-3\sqrt{\ln n} < \delta < 3\sqrt{\ln n}} p(\delta)$ to within $O(\frac{1}{n^2})$. For this, we will need to bound the derivative of $p(\delta)$ with respect to δ, wherever it exists, in the above range. This is done in Lemma 4.3.

LEMMA 4.3. *At any point δ, $|\frac{\mathrm{d}p(\delta)}{\mathrm{d}\delta}| = O(n^7)$ whenever it exists. Consider any $\Delta = O(\frac{1}{n^9})$. Then it follows that $|p(\delta + \Delta) - p(\delta)| \leq O(\frac{1}{n^2})$, for any δ, such that $p(\delta)$ is differentiable everywhere in the range $]\delta \ldots \delta + \Delta[$.*

Proof. The second part follows from the fact that $\frac{|p(\delta+\Delta)-p(\delta)|}{\Delta} = O(n^7)$. This is because the slope of $p()$ must equal $\frac{p(\delta+\Delta)-p(\delta)}{\Delta}$ at some point in the range $]\delta \ldots \delta+\Delta[$.

To show the first part, we show in Appendix 2 that the derivative of each $f_{e,k}(\delta)$ is bounded by $O(n^4)$ wherever it exists. Thus the derivative of $p(\delta)$ will be bounded by $O(r|E|n^4) = O(n^7)$ as claimed. □

COROLLARY 4.4. *If δ_{min} is the point in the range $-3\sqrt{\ln n} < \delta < 3\sqrt{\ln n}$ at which $p(\delta)$ is minimized in this range, then $p(\delta_{min})$ can be approximated to within an $O(\frac{1}{n^2})$ additive error by evaluating $p(\delta)$ at the nearest point in S which is bigger than δ_{min}.*

From Lemma 4.2 and Corollary 4.4, we conclude that $\min_{\delta \in S} p(\delta) - \tau = O(\frac{1}{n^2})$ as required.

5. Substep 2a: Deriving expressions for $f_{e,k}(\delta)$. Recall that $f_{e,k}(\delta) = \Pr(E_k^e | i, j, \delta)$ is the probability that both endpoints v, w of e are assigned the color corresponding to center t_k when the values for all conditional variables before $t_i[j]$ have been determined and $t_i[j]$ is assigned δ. For a fixed edge $e = (v, w)$ and some fixed k, we show how to express $f_{e,k}(\delta)$ in terms of some basic functions and the function $I()$ defined earlier. The exact expression will depend upon which of a number of cases occurs. However, the thing to note is that the expression in each case will have only a constant number of nested integrals. For each case, we will also determine points δ, if any, at which $f_{e,k}(\delta)$ is not differentiable. Recall that these points are necessary in defining S in section 4. Then, in section 6, we will show how $I()$ itself can be expressed in terms of basic functions and nested integrals of depth just 2.

Notation. For convenience, we use the notation $\Pr((x \leq t_k \cdot v \leq x + \mathrm{d}x) \wedge (y \leq t_k \cdot w \leq y + \mathrm{d}y))$ to denote the density function of the joint probability distribution of $t_k \cdot v$ and $t_k \cdot w$, multiplied by $\mathrm{d}x\mathrm{d}y$. Informally speaking, the above term denotes the probability that $t_k \cdot v$ is in the infinitesimal range $x \ldots x + \mathrm{d}x$ and $t_k \cdot w$ is in the infinitesimal range $y \ldots y + \mathrm{d}y$. Similarly, we use the notation $I(v, w, x, x + \mathrm{d}x, y, y + \mathrm{d}y)$

to denote $\Pr((x \le a \cdot b \le x+\mathrm{d}x) \wedge (y \le a \cdot b' \le y+\mathrm{d}y))$, where a is as in the definition of I (see section 3.3). So what $I(v, w, x, x+\mathrm{d}x, y, y+\mathrm{d}y)$ effectively denotes is the density function of the joint probability distribution of $a \cdot b, a \cdot b'$ multiplied by $\mathrm{d}x\mathrm{d}y$. The expression we derive for $f_{e,k}(\delta)$ will have terms of the form $I(v, w, x, x+\mathrm{d}x, y, y+\mathrm{d}y)$. In section 6, we will expand this term out in terms of the actual density function.

Before giving the expressions for $f_{e,k}(\delta)$, we need to reiterate a basic fact.

Fact 1. Note that $t_{i+1}, t_{i+2}, \ldots, t_r$ are all completely undetermined, mutually independent, independent of $t_1, \ldots, t_i$, and identically distributed in a spherically symmetric manner in n dimensions. $t_i[j+1 \ldots n]$ is also undetermined and is spherically symmetrically distributed in $n-j$ dimensions and is independent of $t_{i+1}, \ldots, t_r$ and of all the previously fixed components of t_i.

The cases to be considered. There are three cases, depending upon whether $k < i$, $k = i$, or $k > i$. Each case has three subcases, depending upon whether $j < n-1$, $j = n-1$, or $j = n$. We have to consider these three subcases separately for the following reason: When $j < n-1$, we will express the above probability in terms of the function $I()$. For $j = n-1$ and $j = n$, we cannot express the above probability in terms of $I()$. (Recall that $I()$ was only defined when its argument vectors are at least two-dimensional.) Therefore, in these two subcases, we have to express the probability directly. These two subcases themselves need to be separated because the derivative of $f_{e,k}(\delta)$ behaves differently in these two subcases, and the behavior is crucial to the analysis (setting up S in section 4). Recall Lemma 3.4 in this context.

Note from property 1 of section 3.1 that $v[n], w[n]$ are nonzero. We will need to divide by these quantities at points.

Notation. For vectors a, b, let $a \cdot b[l, m]$ denote $a[l \ldots m] \cdot b[l \ldots m] = \sum_{h=l}^{m} a[h]b[h]$. Let $\alpha' = t_i \cdot v[1 \ldots j-1]$, and let $\beta' = t_i \cdot w[1 \ldots j-1]$.

Case 1 ($k < 1$). In this case, the center t_k already has been determined. Let $t_k \cdot v = \alpha$ and $t_k \cdot w = \beta$. Centers $t_1, \ldots, t_{i-1}$ have also been determined. If one of $t_1 \cdot v, \ldots, t_{i-1} \cdot v$ is greater than α or if one of $t_1 \cdot w, \ldots, t_{i-1} \cdot w$ is greater than β, then $\Pr(E_k^e | i, j, \delta)$ is 0. Otherwise, it is

$$f_{e,k}(\delta) = \Pr(\wedge_{l=i}^{r}(t_l \cdot v \le \alpha \wedge t_l \cdot w \le \beta) | i, j, \delta).$$

Note that the events $t_l \cdot v \le \alpha \wedge t_l \cdot w \le \beta$, $i \le l \le r$, are all independent.

Case 1.1 ($j < n-1$). By Fact 1,

$$\begin{aligned}
f_{e,k}(\delta) &= \Pr(t_i \cdot v \le \alpha \wedge t_i \cdot w \le \beta | i, j, \delta) \times \Pr(t_r \cdot v \le \alpha \wedge t_r \cdot w \le \beta)^{r-i} \\
&= \Pr(\alpha' + \delta v[j] + t_i \cdot v[j+1 \ldots n] \le \alpha \wedge \beta' + \delta w[j] + t_i \cdot w[j+1 \ldots n] \le \beta) \\
&\quad \times \Pr(t_r \cdot v \le \alpha \wedge t_r \cdot w \le \beta)^{r-i} \\
&= \Pr(t_i \cdot v[j+1 \ldots n] \le \alpha - \alpha' - \delta v[j] \wedge t_i \cdot w[j+1 \ldots n] \le \beta - \beta' - \delta w[j]) \\
&\quad \times \Pr(t_r \cdot v \le \alpha \wedge t_r \cdot w \le \beta)^{r-i} \\
&= I(v[j+1 \ldots n], w[j+1 \ldots n], -\infty, \alpha - \alpha' - \delta v[j], -\infty, \beta - \beta' - \delta w[j]) \\
&\quad \times I^{r-i}(v, w, -\infty, \alpha, -\infty, \beta).
\end{aligned}$$

Based on the following lemma, we claim that the derivative of $f_{e,k}(\delta)$ with respect to δ is always defined for the above case. The same will be true for Cases 2.1 and 3.1.

LEMMA 5.1. *$I(b, b', \mathcal{A}(\delta), \mathcal{B}(\delta), \mathcal{C}(\delta), \mathcal{D}(\delta))$ is differentiable with respect to δ for all δ, where $\mathcal{A}(), \mathcal{B}(), \mathcal{C}(), \mathcal{D}()$ are linear functions of δ.*

Proof. The proof of Lemma 5.1 will be described in section 6, after the expression for $I()$ is derived. □

Case 1.2 ($j = n-1$). We derive the expression for $f_{e,k}(\delta)$ for the case when $v[n]$ and $w[n]$ are both positive. The other cases are similar. By Fact 1 and the fact that $t_i[n]$ is normally distributed,

$$\begin{aligned}
f_{e,k}(\delta) &= \Pr(t_i \cdot v \le \alpha \wedge t_i \cdot w \le \beta | i, n-1, \delta) \\
&\quad \times \Pr(t_r \cdot v \le \alpha \wedge t_r \cdot w \le \beta)^{r-i} \\
&= \Pr(t_i[n]v[n] \le \alpha - \alpha' - \delta v[n-1] \wedge t_i[n]w[n] \le \beta - \beta' - \delta w[n-1]) \\
&\quad \times \Pr(t_r \cdot v \le \alpha \wedge t_r \cdot w \le \beta)^{r-i} \\
&= \Pr\left(t_i[n] \le \min\{\frac{\alpha - \alpha' - \delta v[n-1]}{v[n]}, \frac{\beta - \beta' - \delta w[n-1]}{w[n]}\}\right) \\
&\quad \times \Pr(t_r \cdot v \le \alpha \wedge t_r \cdot w \le \beta)^{r-i} \\
&= \left(\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\min\{\frac{\alpha-\alpha'-v[n-1]\delta}{v[n]}, \frac{\beta-\beta'-w[n-1]\delta}{w[n]}\}} \mathrm{e}^{-z^2/2} \mathrm{d}z\right) \times I^{r-i}(v, w, -\infty, \alpha, -\infty, \beta).
\end{aligned}$$

Note that the derivative of $f_{e,k}(\delta)$ with respect to δ is undefined at only one point, namely, the value of δ for which $\frac{\alpha-\alpha'-v[n-1]\delta}{v[n]} = \frac{\beta-\beta'-w[n-1]\delta}{w[n]}$.

Case 1.3 ($j = n$). If $t_i \cdot v = \alpha' + v[n]\delta > \alpha$ or $t_i \cdot w = \beta' + w[n]\delta > \beta$, then t_i has a bigger dot product than t_k with at least one of v or w, and therefore, $f_{e,k}(\delta) = 0$. Otherwise

$$\begin{aligned}
f_{e,k}(\delta) &= \Pr(t_i \cdot v \le \alpha \wedge t_i \cdot w \le \beta | i, j, \delta) \Pr(t_r \cdot v \le \alpha \wedge t_r \cdot w \le \beta)^{r-i} \\
&= \Pr(t_r \cdot v \le \alpha \wedge t_r \cdot w \le \beta)^{r-i} \\
&= I^{r-i}(v, w, -\infty, \alpha, -\infty, \beta).
\end{aligned}$$

Note that the derivative of $f_{e,k}(\delta)$ with respect to δ is undefined only for two values, namely, when $\alpha = \alpha' + v[n]\delta$ and $\beta = \beta' + w[n]\delta$.

Case 2 ($k > i$). Let $\max\{t_1 \cdot v, \ldots, t_{i-1} \cdot v\} = \alpha$ and $\max\{t_1 \cdot w, \ldots, t_{i-1} \cdot w\} = \beta$. $t_k \cdot v$ must be greater than α and $t_k \cdot w$ must be greater than β for t_k to be the color assigned to both v and w. Then let A be the event $t_k \cdot v \ge \alpha \wedge t_k \cdot w \ge \beta$ and B_l be the event $t_l \cdot v \le t_k \cdot v \wedge t_l \cdot w \le t_k \cdot w$, $l \ge i, l \ne k$.

Note that the events B_l in this case are not independent. However, they are independent for fixed values of $t_k \cdot v$ and $t_k \cdot w$. In what follows, we will, at appropriate points, fix $t_k \cdot v$ and $t_k \cdot w$ to be in some infinitesimal intervals and then integrate over these intervals. Within such an integral, the values of $t_k \cdot v$ and $t_k \cdot w$ may be treated as fixed, and therefore, the events corresponding to the B_l's with the above values fixed become independent. Note that we do not do any discretization or approximation here. Rather, what we derive here is an exact integral using the slightly nonstandard but intuitively illustrative notation $\Pr((x \le t_k \cdot v \le x + \mathrm{d}x) \wedge (y \le t_k \cdot w \le y + \mathrm{d}y))$ defined earlier in this section.

Case 2.1 ($j < n-1$).

$$\begin{aligned}
f_{e,k}(\delta) &= \Pr(A \wedge B_i \wedge \cdots B_{k-1} \wedge B_{k+1} \wedge \cdots B_r | i, j, \delta) \\
&= \int_{x=\alpha}^{\infty} \int_{y=\beta}^{\infty} (\Pr((x \le t_k \cdot v \le x + \mathrm{d}x) \wedge (y \le t_k \cdot w \le y + \mathrm{d}y)) \\
&\qquad \times \quad \Pr(t_i \cdot v \le x \wedge t_i \cdot w \le y | i, j, \delta)
\end{aligned}$$

$$\times \prod_{l=i+1,\ldots,k-1,k+1,\ldots,r} \Pr(t_l \cdot v \le x \wedge t_l \cdot w \le y | i, j, \delta)$$

$$= \int_{x=\alpha}^{\infty} \int_{y=\beta}^{\infty} (I(v, w, x, x + \mathrm{d}x, y, y + \mathrm{d}y)$$
$$\times \Pr(\alpha' + \delta v[j] + t_i \cdot v[j+1, n] \le x \wedge \beta' + \delta w[j] + t_i \cdot w[j+1, n] \le y)$$
$$\times I^{r-i-1}(v, w, -\infty, x, -\infty, y))$$
$$= \int_{x=\alpha}^{\infty} \int_{y=\beta}^{\infty} (I(v, w, x, x + \mathrm{d}x, y, y + \mathrm{d}y)$$
$$\times I(v[j+1 \ldots n], w[j+1 \ldots n], -\infty, x - \alpha' - v[j]\delta, -\infty, y - \beta' - w[j]\delta)$$
$$\times I^{r-i-1}(v, w, -\infty, x, -\infty, y)).$$

By Lemma 5.1, $f(\delta)$ is always differentiable with respect to δ in this case.

Case 2.2 ($j = n - 1$). Assume that $v[n]$ and $w[n]$ are positive. The remaining cases are similar. Then $f_{e,k}(\delta) = \Pr(A \wedge B_i \wedge \cdots \wedge B_{k-1} \wedge B_{k+1} \wedge \cdots \wedge B_r | i, n-1, \delta)$

$$= \int_{x=\alpha}^{\infty} \int_{y=\beta}^{\infty} (\Pr((x \le t_k \cdot v \le x + \mathrm{d}x) \wedge (y \le t_k \cdot w \le y + \mathrm{d}y))$$
$$\times \prod_{l=i,\ldots,k-1,k+1,\ldots,r} \Pr(t_l \cdot v \le x \wedge t_l \cdot w \le y | i, n-1, \delta))$$
$$= \int_{x=\alpha}^{\infty} \int_{y=\beta}^{\infty} (I(v, w, x, x + \mathrm{d}x, y, y + \mathrm{d}y)$$
$$\times \Pr(\alpha' + \delta v[n-1] + t_i[n] v[n] \le x \wedge \beta' + \delta w[n-1] + t_i[n] w[n] \le y)$$
$$\times \prod_{l=i+1,\ldots,k-1,k+1,\ldots,r} \Pr(t_l \cdot v \le x \wedge t_l \cdot w \le y))$$
$$= \int_{x=\alpha}^{\infty} \int_{y=\beta}^{\infty} (I(v, w, x, x + \mathrm{d}x, y, y + \mathrm{d}y) \left(\frac{1}{\sqrt{2\pi}} \int_{z=-\infty}^{\min\{\frac{x-\alpha'-v[n-1]\delta}{v[n]}, \frac{y-\beta'-w[n-1]\delta}{w[n]}\}} \mathrm{e}^{-z^2/2} \mathrm{d}z \right)$$
$$\times I^{r-i-1}(v, w, -\infty, x, -\infty, y))$$
$$= \frac{1}{\sqrt{2\pi}} \int_{z=-\infty}^{\infty} \int_{x=\max\{\alpha, \alpha'+v[n]z+v[n-1]\delta\}}^{\infty} \int_{y=\max\{\beta, \beta'+w[n]z+w[n-1]\delta\}}$$
$$(I(v, w, x, x + \mathrm{d}x, y, y + \mathrm{d}y) \times I^{r-i-1}(v, w, -\infty, x, -\infty, y) \mathrm{e}^{-z^2/2}) \mathrm{d}z.$$

Note that the derivative of $f_{e,k}(\delta)$ with respect to δ is undefined only when $\frac{\alpha-\alpha'-v[n-1]\delta}{v[n]} = \frac{\beta-\beta'-w[n-1]\delta}{w[n]}$. We see this by the following argument. Consider the values of δ for which $\frac{\alpha-\alpha'-v[n-1]\delta}{v[n]} < \frac{\beta-\beta'-w[n-1]\delta}{w[n]}$. The above expression for $f_{e,k}(\delta)$ can then be split up into a sum of three terms described below. From the resulting expression, it is clear that it is differentiable for all values of δ such that $\frac{\alpha-\alpha'-v[n-1]\delta}{v[n]} < \frac{\beta-\beta'-w[n-1]\delta}{w[n]}$. A similar argument shows that $f_{e,k}(\delta)$ is differentiable for all values of δ such that $\frac{\alpha-\alpha'-v[n-1]\delta}{v[n]} > \frac{\beta-\beta'-w[n-1]\delta}{w[n]}$.

$$f_{e,k}(\delta) = \tfrac{1}{\sqrt{2\pi}} \int_{z=-\infty}^{\frac{\alpha-\alpha'-v[n-1]\delta}{v[n]}} \int_{x=\alpha}^{\infty} \int_{y=\beta}^{\infty} (I(v, w, x, x + \mathrm{d}x, y, y + \mathrm{d}y)$$
$$\times I^{r-i-1}(v, w, -\infty, x, -\infty, y) \mathrm{e}^{-\frac{z^2}{2}} \mathrm{d}z)$$

$$+\frac{1}{\sqrt{2\pi}}\int_{z=\frac{\alpha-\alpha'-v[n-1]\delta}{v[n]}}^{\frac{\beta-\beta'-w[n-1]\delta}{w[n]}}\int_{x=\alpha'+v[n]z+v[n-1]\delta}^{\infty}\int_{y=\beta}^{\infty}(I(v,w,x,x+\mathrm{d}x,y,y+\mathrm{d}y)$$
$$\times\quad I^{r-i-1}(v,w,-\infty,x,-\infty,y)\mathrm{e}^{-\frac{z^2}{2}}\,\mathrm{d}z)$$
$$+\frac{1}{\sqrt{2\pi}}\int_{z=\frac{\beta-\beta'-w[n-1]\delta}{w[n]}}^{\infty}\int_{x=\alpha'+v[n]z+v[n-1]\delta}^{\infty}$$
$$\int_{y=\beta'+w[n]z+w[n-1]\delta}^{\infty}(I(v,w,x,x+\mathrm{d}x,y,y+\mathrm{d}y)$$
$$\times\quad I^{r-i-1}(v,w,-\infty,x,-\infty,y)\mathrm{e}^{-\frac{z^2}{2}}\,\mathrm{d}z).$$

Case 2.3 ($j=n$). Since $t_i[n]$ is assigned to δ and all other components of t_i are fixed, $t_k\cdot v>\max\{\alpha,\alpha'+v[n]\delta\}$ and $t_k\cdot w>\max\{\beta,\beta'+w[n]\delta\}$ for t_k to be the color assigned to both v and w. Then $f_{e,k}(\delta)=\Pr(A\wedge B_i\wedge\cdots,B_{k-1}\wedge B_{k+1}\wedge\cdots,B_r|i,n,\delta)$

$$=\int_{y=\max\{\beta,\beta'+w[n]\delta\}}^{\infty}\int_{x=\max\{\alpha,\alpha'+v[n]\delta\}}^{\infty}(\Pr((x\le t_k\cdot v\le x+\mathrm{d}x)\wedge(y\le t_k\cdot w\le y+\mathrm{d}y))$$
$$\times\prod_{l=i+1,\ldots,k-1,k+1,\ldots,r}\Pr(t_l\cdot v\le x\wedge t_l\cdot w\le y|i,n,\delta))$$
$$=\int_{y=\max\{\beta,\beta'+w[n]\delta\}}^{\infty}\int_{x=\max\{\alpha,\alpha'+v[n]\delta\}}^{\infty}(\Pr((x\le t_k\cdot v\le x+\mathrm{d}x)\wedge(y\le t_k\cdot w\le y+\mathrm{d}y))$$
$$\times\prod_{l=i+1,\ldots,k-1,k+1,\ldots,r}\Pr(t_l\cdot v\le x\wedge t_l\cdot w\le y))$$
$$=\int_{\max\{\beta,\beta'+w[n]\delta\}}^{\infty}\int_{\max\{\alpha,\alpha'+v[n]\delta\}}^{\infty}(I(v,w,x,x+\mathrm{d}x,y,y+\mathrm{d}y)$$
$$\times\ I^{r-i-1}(v,w,-\infty,x,-\infty,y)).$$

Note that the derivative of the above expression with respect to δ is undefined only for two values, namely, when $\alpha=\alpha'+v[n]\delta$ and $\beta=\beta'+w[n]\delta$.

Case 3 ($k=i$). Let $\max\{t_1\cdot v,\ldots,t_{i-1}\cdot v\}=\alpha$ and $\max\{t_1\cdot w,\ldots,t_{i-1}\cdot w\}=\beta$. $t_i\cdot v>\alpha$ and $t_i\cdot w>\beta$ for t_i to be the color assigned to both v and w. Then, let A be the event $t_i\cdot v\ge\alpha\wedge t_i\cdot w\ge\beta$ and B_l be the event $t_l\cdot v\le t_i\cdot v\wedge t_l\cdot w\le t_i\cdot w$, $l>i$.

Again, note that the events B_l in this case are not independent. However, they are independent for fixed values of $t_i\cdot v$ and $t_i\cdot w$.

Case 3.1 ($j<n-1$).

$$f_{e,k}(\delta)=\Pr(A\wedge B_{i+1}\wedge\cdots\wedge B_r|i,j,\delta)$$
$$=\int_{x=\alpha}^{\infty}\int_{y=\beta}^{\infty}(I(v[j+1\ldots n],w[j+1\ldots n],x-\alpha'-v[j]\delta,x+\mathrm{d}x-\alpha'-v[j]\delta,$$
$$y-\beta'-w[j]\delta,y+\mathrm{d}y-\beta'-w[j]\delta)$$
$$\times\ I^{r-i}(v,w,-\infty,x,-\infty,y)).$$

By Lemma 5.1, $f(\delta)$ is always differentiable with respect to δ in this case.

Case 3.2 ($j=n-1$). Assume that $v[n]$ and $w[n]$ are positive. The other cases are similar.

$$f_{e,k}(\delta) = \Pr(A \wedge B_{i+1} \wedge \cdots \wedge B_r | i, n-1, \delta)$$
$$= \frac{1}{\sqrt{2\pi}} \int_{z=\max\{\frac{\alpha-\alpha'-v[n-1]\delta}{v[n]}, \frac{\beta-\beta'-w[n-1]\delta}{w[n]}\}}^{\infty} \times (I^{r-i}(v, w, -\infty, \alpha'+v[n-1]\delta+v[n]z, -\infty, \beta'+w[n-1]\delta+w[n]z)\mathrm{e}^{-z^2/2}\mathrm{d}z).$$

Note that the derivative of the above expression with respect to δ is undefined only when $\frac{\alpha-\alpha'-v[n-1]\delta}{v[n]} = \frac{\beta-\beta'-w[n-1]\delta}{w[n]}$.

Case 3.3 ($j = n$). If $v[n]\delta + \alpha' < \alpha$ or $w[n]\delta + \beta' < \beta$ then this probability is 0. Otherwise,

$$f_{e,k}(\delta) = \Pr(A \wedge B_{i+1} \wedge \cdots \wedge B_r | i, j, \delta) = I^{r-i}(v, w, -\infty, \alpha' + v[n]\delta, -\infty, \beta' + w[n]\delta).$$

Note that the derivative of the above expression with respect to δ is possibly undefined only for at most two values, namely, when $\alpha = \alpha' + v[n]\delta$ and $\beta = \beta' + w[n]\delta$.

6. Substep 2a: Expressing $I(b, b', x, y, x', y')$. Recall that $I(b, b', x, y, x', y')$ denotes $\Pr((x \le a \cdot b \le y) \wedge (x' \le a \cdot b' \le y'))$, where a is a vector whose entries are independent and normally distributed with mean 0 and variance 1. We show how to derive an expression for this probability in terms of nested integrals of depth 2, with the integrand comprising only basic functions. Let b and b' be h-dimensional. Note that $h \ge 2$. Consider the h-dimensional coordinate system with respect to which b, b' are specified.

The naive way. Note that a naive way to compute I is to perform a sequence of h nested integrals, one over each of $a[1] \ldots a[h]$, the coordinates of the vector a. The following is the naive expression for the case when $b[h]$ and $b'[h]$ are both positive. A similar expression holds for other cases.

$$\left(\frac{1}{\sqrt{2\pi}}\right)^h \int_{a[1]=-\infty}^{\infty} \int_{a[2]=-\infty}^{\infty} \cdots \int_{a[h-1]=-\infty}^{\infty} \int_{a[h]=\alpha}^{\beta} \mathrm{e}^{-\frac{\sum_{l=1}^{h} a[l]^2}{2}} \mathrm{d}a[h]\mathrm{d}a[h-1]\cdots\mathrm{d}a[1],$$

where

$$\alpha = \max\left\{ \left(x - \sum_{1}^{h-1} a[i]b[i]\right)/b[h], \left(x' - \sum_{1}^{h-1} a[i]b'[i]\right)/b'[h] \right\},$$

$$\beta = \min\left\{ \left(y - \sum_{1}^{h-1} a[i]b[i]\right)/b[h], \left(y' - \sum_{1}^{h-1} a[i]b'[i]\right)/b'[h] \right\}.$$

Computing this integral to within the required error bounds in polynomial time seems hard. We use the following method instead.

Our method. Note that since each coordinate of a is normally distributed with mean 0 and variance 1, a has a spherically symmetric distribution. We rotate the coordinate system so that $b = (b_1, 0, \ldots, 0)$ and $b' = (b'_1, b'_2, 0, \ldots, 0)$, where $b_1, b'_2 \ge 0$. As we will show shortly, both b_1, b'_2 will be strictly positive for all our calls to I. Let $a'[1]a'[2]\ldots a'[h]$ be the coordinates of a under the rotated coordinate system. The following lemma is key.

LEMMA 6.1. *The probability distribution of a' is identical to that of a. That is, all the coordinates of a' are independently distributed according to the normal distribution with mean* 0 *and variance* 1.

Proof. Let $x_1, \ldots, x_h$ denote the initial coordinate system, and let $x'_1, x'_2, \ldots, x'_h$ denote the coordinate system after rotation. Then $(x_1, x_2, \ldots, x_h)A = (x'_1, x'_2, \ldots, x'_h)$, where A is the orthonormal rotation matrix; i.e., $AA^T = I$.

Next, recall that the probability density function of $a[l]$ is $\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{x_l^2}{2}}$ for $l = 1 \ldots h$. We show that the probability density function of $a'[l]$ is $\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{(x'_l)^2}{2}}$ for $l = 1 \ldots h$. We also show that the joint probability density function of the $a'[l]$s is just the product of their individual density functions. The lemma then follows.

The density function of the joint distribution of $a[1], \ldots, a[h]$ is $\frac{1}{(\sqrt{2\pi})^h}\mathrm{e}^{-\frac{\Sigma_1^h x_l^2}{2}}$. We now derive the joint distribution of $a'[1], \ldots, a'[h]$. Since $(x_1, x_2, \ldots, x_h)A = (x'_1, x'_2, \ldots, x'_h)$ and A is orthonormal, $\sum_1^h x_l^2 = \sum_1^h (x'_l)^2$. Using the standard method for performing coordinate transformation, the density function of the joint distribution of $a'[1], \ldots, a'[h]$ is $\frac{1}{(\sqrt{2\pi})^h}\mathrm{e}^{-\frac{\Sigma_1^h (x'_l)^2}{2}} \det(B)$, where B is the matrix whose p, qth entry is $\frac{\partial x_p}{\partial x'_q}$. Since $(x_1, x_2, \ldots, x_h) = (x'_1, x'_2, \ldots, x'_h)A^T$, the matrix B is easily seen to be identical to A. Since A is orthonormal, $\det(A) = 1$, and therefore, the density function of the joint distribution of $a'[1], \ldots, a'[h]$ is just $\frac{1}{(\sqrt{2\pi})^h}\mathrm{e}^{-\frac{\Sigma_1^h (x'_l)^2}{2}}$.

Finally, the density function of $a'[l]$ can be seen to be $\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{(x'_l)^2}{2}}$ by integrating away the other terms, i.e.,

$$\int_{x_1=-\infty}^{\infty}\int_{x_2=-\infty}^{\infty}\cdots\int_{x_{l-1}=-\infty}^{\infty}\int_{x_{l+1}=-\infty}^{\infty}\cdots\int_{x_h=-\infty}^{\infty}\left(\frac{1}{(\sqrt{2\pi})^h}\mathrm{e}^{-\frac{\sum_1^h (x'_l)^2}{2}}\right) \times \mathrm{d}x'_h\mathrm{d}x'_{h-1}\cdots\mathrm{d}x'_{l+1}\mathrm{d}x'_{l-1}\cdots\mathrm{d}x'_1 = \frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{(x'_l)^2}{2}}. \qquad \square$$

Having rotated the coordinate axes, note that $a' \cdot b = a'[1]b_1$ and $a' \cdot b' = a'[1]b'_1 + a'[2]b'_2$. Now $I(b, b', x, y, x', y')$ denotes $\Pr((x \le a'[1]b_1 \le y) \wedge (x' \le a'[1]b'_1 + a'[2]b'_2 \le y'))$. We give the expression for $I(b, b', x, y, x', y')$ in the following lemma for future reference.

LEMMA 6.2.

$$I(b, b', x, y, x', y') = \Pr((x \le a'[1]b_1 \le y) \wedge (x' \le a'[1]b'_1 + a'[2]b'_2 \le y'))$$

$$= \Pr\left(\left(\frac{x}{b_1} \le a'[1] \le \frac{y}{b_1}\right) \wedge \left(\frac{x' - a'[1]b'_1}{b'_2} \le a'[2] \le \frac{y' - a'[1]b'_1}{b'_2}\right)\right)$$

$$= \frac{1}{2\pi}\int_{\frac{x}{b_1}}^{\frac{y}{b_1}} e^{-\frac{z^2}{2}}\left(\int_{\frac{(x' - zb'_1)}{b'_2}}^{\frac{(y' - zb'_1)}{b'_2}} e^{-\frac{z'^2}{2}}\ \mathrm{d}z'\right)\ \mathrm{d}z.$$

LEMMA 6.3. *$|b_1| = \Omega(\epsilon) = \Omega(\frac{1}{n^2})$ and $|b'_2| = \Omega(\epsilon^2) = \Omega(\frac{1}{n^4})$.*

Proof. Note that b, b' are of the form $v[h \ldots n], w[h \ldots n]$ for vertex vectors v and w and $h \le n-1$ in all the calls we make to $I()$. The lemma now follows from property 3 of the discretization of vertex vectors in section 3.1. $\square$

Two remarks. We remark that Lemma 5.1 can be easily seen to hold by inspecting the expression derived in Lemma 6.2, with x, x', y, y' replaced by linear functions of δ. Second, recall from section 5 that some of the expressions involve occurrences of $I()$ with $y = x + \mathrm{d}x$ and $y' = x' + \mathrm{d}x'$. It can be easily seen that

$$\Pr((x \le a{\cdot}b \le x{+}\mathrm{d}x){\wedge}(x' \le a{\cdot}b' \le x'{+}\mathrm{d}x')) = \frac{1}{2\pi}\frac{1}{b_1 b_2'}\mathrm{e}^{-\frac{(x/b_1)^2}{2}}\mathrm{e}^{-\frac{(x'-xb_1'/b_1)^2}{2(b_2')^2}}\ \mathrm{d}x'\ \mathrm{d}x.$$

7. Substep 2b: Evaluating $f_{e,k}(\delta)$. Sections 5 and 6 derived expressions for $f_{e,k}(\delta)$, which were nested integrals with constant depth (at most five). We now show how to evaluate these expressions at any given value of δ in polynomial time with just $O(\frac{1}{n^5})$ error.

First, in Lemma 7.1, we show that all the expressions we derived in sections 5 and 6 involve integrations where the integrand has a particular form. This enables us to focus on only integrations of this form. We show how to perform each such integration within an inverse polynomial error, with the polynomial being chosen so that the final error in computing $f_{e,k}(\delta)$ is $O(\frac{1}{n^5})$ as required.

To perform each such integration, we have to do two things: first, restrict the range of the limits of integration and, second, convert the integration to a summation and compute the summation. Each of these steps will incur an error, which we will show can be made inverse polynomial, with the actual polynomial chosen to keep the overall error within the stated bounds. The error in restricting the range of the limits of integration is bounded in Lemma 7.2. To bound the error in converting the integrations to summations, we give Lemma 7.3, which states that it suffices to bound the absolute value of the derivative of the integrands. In Lemma 7.4, we show that the derivative of each integrand is bounded by some polynomial in n. Together Lemmas 7.1, 7.2, 7.3, and 7.4 imply that each integration can be computed to within an inverse polynomial error. Finally, since expressions for $f_{e,k}(\delta)$ involve up to five nested integrations, the inverse polynomial error terms in each integration have to be chosen so that the final combined error of all five integrations is $O(\frac{1}{n^5})$. This is described under the heading Algorithm for Performing Integrations. Lemma 7.1 obtains the general form of each integration.

LEMMA 7.1. *Each integration we perform can be expressed in the following form:*

$$\int_l^m \frac{1}{\sqrt{2\pi}} e^{-\frac{h^2}{2}} H(\mathcal{G}(h))\mathrm{d}h$$

for some function $\mathcal{G}(h)$, where $H()$ is such that $0 \le H(e) \le 1$ for all e.

Proof. This is easily verified by an inspection of the expressions to be integrated in section 5 and the integral for $I()$ in section 6. The functions $H()$ are always probabilities. The only fact to be noted is that $I(v, w, x, x + \mathrm{d}x, y, y + \mathrm{d}y)$, which appears in the integrals in Case 2 of section 5, equals

$$\frac{1}{\sqrt{2\pi}}\frac{1}{b_1}\mathrm{e}^{-\frac{x^2}{2b_1^2}}\frac{1}{\sqrt{2\pi}}\frac{1}{b_2'}\mathrm{e}^{-\frac{\left(y-x\frac{b_1'}{b_1}\right)^2}{2(b_2')^2}}\mathrm{d}y\mathrm{d}x = \frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{h^2}{2}}\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{(h')^2}{2}}\mathrm{d}h'\mathrm{d}h,$$

where $v = (b_1, 0, \ldots, 0)$ and $w = (b_1', b_2', 0, \ldots, 0)$ in the rotated coordinate system, as in section 6, and the last equality is obtained by a change of variables $h = \frac{x}{b_1}$ and $h' = \frac{y - xb_1'/b_1}{b_2'}$. This change of variables affects the limits of the integration, but we are not claiming any special properties for the limits.

Similarly,

$$I(v[j+1\ldots n], w[j+1\ldots n], x-\alpha'-v[j]\delta, x+\mathrm{d}x-\alpha'-v[j]\delta, y-\beta'-w[j]\delta, y+\mathrm{d}y-\beta'-w[j]\delta),$$

which appears in the integrals in Case 3 of section 5, equals

$$\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{h^2}{2}}\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{(h')^2}{2}}\mathrm{d}h'\mathrm{d}h,$$

where $v[j+1\ldots n] = (b_1, 0, \ldots, 0)$ and $w[j+1\ldots n] = (b'_1, b'_2, 0, \ldots, 0)$ in the rotated coordinate system, as in section 6, and the last equality is obtained by a change of variables $h = \frac{x-\alpha'-v[j]\delta}{b_1}$ and $h' = \frac{y-\beta'-w[j]\delta-\frac{(x-\alpha'-v[j]\delta)b'_1}{b_1}}{b'_2}$. □

The next lemma shows that limits of each integration we perform can be clipped to some small range.

LEMMA 7.2.

$$\int_{\max\{l,-a\sqrt{\ln n}\}}^{\min\{m,a\sqrt{\ln n}\}} \frac{1}{\sqrt{2\pi}} e^{-\frac{h^2}{2}} H(\mathcal{G}(h))\mathrm{d}h \le \int_l^m \frac{1}{\sqrt{2\pi}} e^{-\frac{h^2}{2}} H(\mathcal{G}(h))\mathrm{d}h$$

$$\le \int_{\max\{l,-a\sqrt{\ln n}\}}^{\min\{m,a\sqrt{\ln n}\}} \frac{1}{\sqrt{2\pi}} e^{-\frac{h^2}{2}} H(\mathcal{G}(h))\mathrm{d}h + O\left(\frac{1}{n^{a^2/2}}\right)$$

for all $a > 0$.

Proof. The first inequality is obvious. The second is derived using Theorem 4.1 as follows:

$$\begin{aligned}
&\int_l^m \frac{1}{\sqrt{2\pi}}\left(\mathrm{e}^{-\frac{h^2}{2}} H(\mathcal{G}(h))\right)\mathrm{d}h \\
&\le \int_{\max\{l,-a\sqrt{\ln n}\}}^{\min\{m,a\sqrt{\ln n}\}} \frac{1}{\sqrt{2\pi}}\left(\mathrm{e}^{-\frac{h^2}{2}} H(\mathcal{G}(h))\right)\mathrm{d}h + \int_{-\infty}^{-a\sqrt{\ln n}} \frac{1}{\sqrt{2\pi}}\left(\mathrm{e}^{-\frac{h^2}{2}} H(\mathcal{G}(h))\right)\mathrm{d}h \\
&\quad + \int_{a\sqrt{\ln n}}^{\infty} \frac{1}{\sqrt{2\pi}}\left(\mathrm{e}^{-\frac{h^2}{2}} H(\mathcal{G}(h))\right)\mathrm{d}h \\
&\le \int_{\max\{l,-a\sqrt{\ln n}\}}^{\min\{m,a\sqrt{\ln n}\}} \frac{1}{\sqrt{2\pi}}\left(\mathrm{e}^{-\frac{h^2}{2}} H(\mathcal{G}(h))\right)\mathrm{d}h + \int_{-\infty}^{-a\sqrt{\ln n}} \frac{1}{\sqrt{2\pi}}\left(\mathrm{e}^{-\frac{h^2}{2}}\right)\mathrm{d}h \\
&\quad + \int_{a\sqrt{\ln n}}^{\infty} \frac{1}{\sqrt{2\pi}}\left(\mathrm{e}^{-\frac{h^2}{2}}\right)\mathrm{d}h \\
&\le \int_{\max\{l,-a\sqrt{\ln n}\}}^{\min\{m,a\sqrt{\ln n}\}} \frac{1}{\sqrt{2\pi}}\left(\mathrm{e}^{-\frac{h^2}{2}} H(\mathcal{G}(h))\right)\mathrm{d}h + 2\int_{a\sqrt{\ln n}}^{\infty} \frac{1}{\sqrt{2\pi}}\left(\mathrm{e}^{-\frac{h^2}{2}}\right)\mathrm{d}h \\
&\le \int_{\max\{l,-a\sqrt{\ln n}\}}^{\min\{m,a\sqrt{\ln n}\}} \frac{1}{\sqrt{2\pi}}\left(\mathrm{e}^{-\frac{h^2}{2}} H(\mathcal{G}(h))\right)\mathrm{d}h + O\left(\frac{1}{n^{a^2/2}}\right).
\end{aligned}$$

Theorem 4.1 is used in the last step above. The fact that $0 \le H() \le 1$ is used in the second step. □

The next lemma is classical and will be used to show that each integration can be converted to a summation by discretizing the range between the limits of integration.

LEMMA 7.3. $|\int_l^{l+\rho} \frac{1}{\sqrt{2\pi}}(e^{-\frac{h^2}{2}} H(\mathcal{G}(h)))\mathrm{d}h - \frac{1}{\sqrt{2\pi}} e^{-\frac{l^2}{2}} H(\mathcal{G}(l))\rho| \leq \frac{M\rho^2}{2}$, *where* M *upper bounds the derivative of* $\frac{1}{\sqrt{2\pi}}(e^{-\frac{h^2}{2}} H(\mathcal{G}(h)))$ *with respect to* h.

LEMMA 7.4. *The derivative of* $\frac{1}{\sqrt{2\pi}}(e^{-\frac{h^2}{2}} H(\mathcal{G}(h)))$ *with respect to* h *is at most a polynomial in* n *in absolute value in all our integrations.*

Proof. The proof of Lemma 3 is found in Appendix 3. □

ALGORITHM FOR PERFORMING INTEGRATIONS. The four lemmas above lead to the following algorithm for performing integrations. Consider a particular integral $\int_l^m \frac{1}{\sqrt{2\pi}}(\mathrm{e}^{-\frac{h^2}{2}} H(\mathcal{G}(h)))\mathrm{d}h$. We first replace the above integral with

$$\int_{\max\{l,-a\sqrt{\ln n}\}}^{\min\{m,a\sqrt{\ln n}\}} \frac{1}{\sqrt{2\pi}} \left(\mathrm{e}^{-\frac{h^2}{2}} H(\mathcal{G}(h))\right) \mathrm{d}h.$$

Here a will be a constant to be fixed later. Next, we convert this integral to a sum by dividing the range between the limits of integration into steps of size $\Theta(\frac{1}{n^b})$ for some b to be fixed later.

Suppose the derivative of $\frac{1}{\sqrt{2\pi}}(\mathrm{e}^{-\frac{h^2}{2}} H(\mathcal{G}(h)))$ is bounded by $O(n^c)$. We compute the total error incurred above.

By Lemma 7.2, clipping the limits of integration incurs an error of $O(\frac{1}{n^{a^2/2}})$. By Lemma 7.3, the error incurred in each step of the summation is $O(\frac{n^c}{n^{2b}})$, assuming there is no error in computing $\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{l^2}{2}} H(\mathcal{G}(h))$. However, $H()$ itself may have been obtained as a result of performing a nested integration or as a product of $O(n)$ distinct integrations nested one level deeper (as in Case 2.2 of section 5, for example). This implies that the value of $H()$ computed itself will have some error. So suppose we have computed each of these nested integrations within an error of $O(\frac{1}{n^f})$. Then the error in $H()$ is $O(\frac{1}{n^{f-1}})$. Therefore, the error incurred in each step of the summation is $O(\frac{n^c}{n^{2b}} + \frac{1}{n^{f-1}n^b})$; this sums to $O(\sqrt{\ln n}\frac{n^c}{n^b} + \frac{\sqrt{\ln n}}{n^{f-1}})$ over all $O(\sqrt{\ln n}nn^b)$ steps. The total error is thus $O(\frac{1}{n^{a^2/2}} + \sqrt{\ln n}\frac{n^c}{n^b} + \frac{\sqrt{\ln n}}{n^{f-1}})$ and the time taken for this integration (ignoring the time taken for the nested integrals in $H()$) is $O(\sqrt{\ln n}nn^b)$.

Finally, note that the depth of nesting in our integrals is at most five (in Case 2.2 of section 5, it is five). It can be easily seen that starting with the innermost integral and working outward, values a, b can be chosen for these successive integrals based upon the respective c, f values so that the final error is $O(\frac{1}{n^5})$.

8. Comments on derandomizing the Max-Cut algorithm. We need to show that discretizing the vectors in this case so as to satisfy properties 1–3 (with ϵ chosen appropriately) of section 3.1 is justified. In the case of the Karger–Motwani–Sudan algorithm, it was justified using Theorem 2.1.

To compensate for this theorem we need only to observe that the value of the Goemans and Williamson objective function (that is, $\sum_{i,j} w_{ij}\frac{1-v_i.v_j}{2}$) for the discretized vector configuration is at least $(1 - \frac{1}{poly(n)})$ times that for the initial vector set (this is because the sum of the edge weights is at most twice the value of the objective function for the initial vector set). The rest is just a matter of choosing the appropriate inverse polynomial terms.

9. Conclusions. We believe that the techniques used here can be used to derandomize a general class of randomized algorithms based on semidefinite programming. Loosely speaking, this class would comprise those whose expected value calculations

involve just a constant number of vectors in each "elementary" event. This class contains all randomized semidefinite programming based algorithms known so far. It would be nice to obtain a general theorem to this effect.

Also, it would be nice to obtain a more efficient derandomization scheme, since the running time of our algorithm is a large polynomial, around $O(n^{30})$ for the 3-coloring problem.

Appendix 1. Discretizing vertex vectors. Let ϵ be a parameter which is $\Theta(\frac{1}{n^2})$. In this section, we show how to discretize the vertex vectors so as to satisfy the three properties specified in section 3.1.

The vertex vectors are considered one by one in the order $v_1, v_2, \ldots, v_n$. We describe the processing of vector v_i.

First, each entry in v_i is rounded upward (in absolute value) to the nearest nonzero multiple of ϵ. Next, up to $2n\epsilon$ is added to $v_i[n-1]$ so that $|v_i[n-1]v_j[n] - v_i[n]v_j[n-1]| > \epsilon^2$ for every $j < i$. This is done to satisfy property 3. Property 1 is clearly satisfied for v_i now. Property 2 is also satisfied because each component of v_i other than $v_i[n-1]$ changes by $O(\epsilon)$ and $v_i[n-1]$ changes by $O(n\epsilon)$. However, note that in this process the vertex vectors no longer remain unit vectors. In fact, $1 \leq |v_i|^2 \leq 2$ now, for small enough ϵ, i.e., for large enough n. So we divide each vector v_i by its new norm and make it a unit vector. Since we divide by a number between one and two, property 1 and property 2 continue to hold.

It remains to show that property 3 holds. We need the following lemma.

LEMMA 9.1. *For each pair of vertex vectors v, w and every h, $1 \leq h < n$, when the coordinate system is rotated so that $v[h \ldots n] = (b_1, 0, \ldots, 0)$ and $w[h \ldots n] = (b_1', b_2', 0, \ldots, 0)$, b_1 and b_2' are at least some inverse polynomial (more precisely, $\Omega(\epsilon)$ and $\Omega(\epsilon^2)$, respectively) in absolute value.*

Proof. Let $v' = v[h \ldots n]$ and $w' = w[h \ldots n]$. Note that b_1 is just the norm of v' which is $\Omega(\epsilon)$ by property 1. Also note that $|b_2'| = \sqrt{||w'|^2 - \frac{(v' \cdot w')^2}{|v'|^2}|}$, since b_2' is just the projection of w' on the line orthogonal to v' in the plane containing v' and w'. So we need to show that $||w'|^2 - \frac{(v' \cdot w')^2}{|v'|^2}| = \Omega(\epsilon^4)$ for every h $1 \leq h < n$.

First consider $h = n-1$. $(v' \cdot w')^2 = (v[n-1]w[n-1] + v[n]w[n])^2 = (v[n-1]^2 + v[n]^2)(w[n-1]^2 + w[n]^2) - (v[n-1]w[n] - w[n-1]v[n])^2 \leq |v'|^2|w'|^2 - \Omega(\epsilon^4)$. Therefore, $|w'|^2 - \frac{(v' \cdot w')^2}{|v'|^2} = \Omega(\epsilon^4)$.

Next consider $h < n-1$. Let $l = v[h \ldots n-2]$ and $m = w[h \ldots n-2]$. Let $l' = v[n-1, n]$ and $m' = w[n-1, n]$; $(v' \cdot w')^2 = (l \cdot m + l' \cdot m')^2 = (l \cdot m)^2 + (l' \cdot m')^2 + 2(l' \cdot m')(l \cdot m) \leq |l|^2|m|^2 + (l' \cdot m')^2 + 2|l'||m'||l||m|$. By the previous paragraph, $(l' \cdot m')^2 \leq |l'|^2|m'|^2 - \Omega(\epsilon^4)$. Therefore, $(v' \cdot w')^2 \leq |l|^2|m|^2 + |l'|^2|m'|^2 + |l'|^2|m|^2 + |l|^2|m'|^2 - \Omega(\epsilon^4) \leq (|l|^2 + |l'|^2)(|m|^2 + |m'|^2) - \Omega(\epsilon^4) = |v'|^2|w'|^2 - \Omega(\epsilon^4)$. Therefore, $||w'|^2 - \frac{(v' \cdot w')^2}{|v'|^2}| = \Omega(\epsilon^4)$. ∎

Appendix 2. Proof of Lemma 4.3. We show that for each edge e and each center k, the derivative of $f_{e,k}(\delta)$ (with respect to δ) is $O(n^4)$.

Recall from section 5 that the expression for $f_{e,k}(\delta)$ depends upon which one of Cases 1, 2, and 3 and which one of the conditions $j < n-1, j = n-1, j = n$ hold.

We show the above claim only for one representative case, i.e., Case 2.1, where $j < n-1$. The other cases can be shown similarly. For Case 2.1

$$f_{e,k}(\delta) = \int_{\alpha}^{\infty} \int_{\beta}^{\infty} g(x, y) h(x, y, \delta) \mathrm{d}y \mathrm{d}x,$$

where

$$g(x,y)\mathrm{d}y\mathrm{d}x = I(v,w,x,x+\mathrm{d}x,y,y+\mathrm{d}y)I^{r-i-1}(v,w,-\infty,x,-\infty,y)$$

and

$$\begin{aligned}h(x,y,\delta) =& I(v[j+1\ldots n],w[j+1\ldots n],-\infty,x-t_i\cdot v[1\ldots j-1]-v[j]\delta,-\infty,\\ & y-t_i\cdot w[1\ldots j-1]-w[j]\delta).\end{aligned}$$

Now

$$|f'_{e,k}(\delta)| \le \int_\alpha^\infty \int_\beta^\infty |g(x,y)|\left|\frac{\partial h(x,y,\delta)}{\partial\delta}\right|\mathrm{d}y\mathrm{d}x \le \max_{x,y}\left|\frac{\partial h(x,y,\delta)}{\partial\delta}\right|$$

since $\int_\alpha^\infty \int_\beta^\infty g(x,y)\mathrm{d}y\mathrm{d}x$ is a probability and therefore ≤ 1. We show that $|f'_{e,k}(\delta)| = O(n^4)$ by estimating $\max_{x,y}|\frac{\partial h(x,y,\delta)}{\partial\delta}|$. Let $c(x,\delta) = x - t_i\cdot v[1\ldots j-1] - v[j]\delta = c' - v[j]\delta$ and $d(y,\delta) = y - t_i\cdot w[1\ldots j-1] - w[j]\delta = d' - w[j]\delta$.

By Lemma 6.2, $h(x,y,\delta) = \frac{1}{2\pi}\int_{-\infty}^{c(x,\delta)/b_1} \mathrm{e}^{-\frac{z^2}{2}}(\int_{-\infty}^{(d(y,\delta)-zb'_1)/b'_2}\mathrm{e}^{-\frac{z'^2}{2}}\ \mathrm{d}z')\mathrm{d}z$, where b_1, b'_1, b'_2 are obtained by rotating the coordinates, as in section 6. Let $G(y,\delta,l) = \frac{1}{\sqrt{2\pi}}\int \mathrm{e}^{-\frac{l^2}{2}}H(y,\delta,l)\mathrm{d}l$, where $H(y,\delta,l) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{(d(y,\delta)-lb'_1)/b'_2}\mathrm{e}^{-\frac{z'^2}{2}}\ \mathrm{d}z'$. Then $\frac{\partial h}{\partial\delta}$ can be expressed as $A+B$, where

$$A = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{c(x,\delta)/b_1}\mathrm{e}^{-\frac{l^2}{2}}\frac{\partial H(y,\delta,l)}{\partial\delta}\mathrm{d}l$$

and

$$B = \frac{\partial G}{\partial l}_{l=c(x,\delta)/b_1}\frac{\partial l}{\partial\delta}_{l=c(x,\delta)/b_1}.$$

Therefore, $|\frac{\partial h}{\partial\delta}| = |A+B| \le |A|+|B|$. Next, we bound $|A|$ and $|B|$ separately.

$|B|$ is bounded as follows: Note that $|\frac{\partial G}{\partial l}| = |\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{l^2}{2}}H(y,\delta,l)| \le 1$ for all l, since $0 \le H(y,\delta,l) \le 1$. Further, $|\frac{\partial l}{\partial\delta}_{l=c(x,\delta)/b_1}| = |v[j]/b_1| = O(n^2)$, by Lemma 6.3. Therefore, $|B| = O(n^2)$.

$|A|$ is bounded as follows: $|A| = |\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{c(x,\delta)/b_1}\mathrm{e}^{-\frac{l^2}{2}}\frac{\partial H(y,\delta,l)}{\partial\delta}\mathrm{d}l| \le \max_{y,\delta,l}|\frac{\partial H(y,\delta,l)}{\partial\delta}|$. It remains to bound $\max_{y,\delta,l}|\frac{\partial H(y,\delta,l)}{\partial\delta}|$. This is done below using the same technique as above. Recall that

$$H(y,\delta,l) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{(d(y,\delta)-lb'_1)/b'_2}\mathrm{e}^{-\frac{z'^2}{2}}\ \mathrm{d}z'.$$

Let $J(m) = \frac{1}{\sqrt{2\pi}}\int \mathrm{e}^{-\frac{m^2}{2}}\ \mathrm{d}m$. Then $|\frac{\partial H}{\partial\delta}| = |\frac{\mathrm{d}J}{\mathrm{d}m}_{m=(d(y,\delta)-lb'_1)/b'_2}||\frac{\partial m}{\partial\delta}_{m=(d(y,\delta)-lb'_1)/b'_2}| \le |w[j]/b'_2| = O(n^4)$, by Lemma 6.3. Therefore, $|f'_{e,k}(\delta)| \le |A|+|B| = O(n^4)$.

Appendix 3. Proof of Lemma 7.4.

Bounding derivatives of integrands in $I()$. Recall that

$$I(b,b',x,y,x',y') = \frac{1}{\sqrt{2\pi}} \int_{\frac{x}{b_1}}^{\frac{y}{b_1}} \mathrm{e}^{\frac{-z^2}{2}} \left(\int_{\frac{(x'-zb_1')}{b_2'}}^{\frac{(y'-zb_1')}{b_2'}} \frac{1}{\sqrt{2\pi}} \mathrm{e}^{\frac{-z'^2}{2}} \mathrm{d}z' \right) \mathrm{d}z.$$

Here b, b' have been rotated so that $b = (b_1, 0, \ldots, 0)$ and $b' = (b_1', b_2', 0, \ldots, 0)$.

The derivative of $\frac{1}{\sqrt{2\pi}} \mathrm{e}^{\frac{-z'^2}{2}}$ with respect to z' is $-\frac{1}{\sqrt{2\pi}} z' \mathrm{e}^{\frac{-z'^2}{2}}$, which is bounded in absolute value by $\frac{1}{\sqrt{2\pi}} \mathrm{e}^{\frac{-1}{2}}$, a constant.

Next, we compute the derivative of the outer integrand. We first denote the inner integral by $h(z)$. Then we compute the derivative of the function to be integrated; that is, $\frac{1}{\sqrt{2\pi}} h(z) \mathrm{e}^{\frac{-z^2}{2}}$ is

$$\frac{1}{\sqrt{2\pi}} \left(-z \mathrm{e}^{\frac{-z^2}{2}} h(z) + \frac{1}{2\pi} \mathrm{e}^{\frac{-z^2}{2}} (-b_1'/b_2') \left(\mathrm{e}^{\frac{-1}{2}\left(\frac{y'-zb_1'}{b_2'}\right)^2} - \mathrm{e}^{-\frac{1}{2}\left(\frac{x'-zb_1'}{b_2'}\right)^2} \right) \right).$$

The first term in this sum is bounded in absolute value by a constant as $h(z) \leq 1$, and the second term is bounded by $O(n^4)$ by Lemma 6.3. Hence the derivative is bounded by $O(n^4)$.

Bounding derivatives of other integrands. We bound the derivatives for the integrands in Case 2.2 of section 5. This is the most complicated case. For other cases, a similar procedure works.

Recall that in this case, the conditional probability $f_{e,k}(\delta)$ can be split into three terms. We show how the derivatives of the integrands involved in the first term can be bounded by polynomial functions of n. The remaining two terms are similar.

The first term is

$$g(\delta) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{\alpha-\alpha'-v[n-1]\delta}{v[n]}} \int_{x=\alpha}^{\infty} \int_{y=\beta}^{\infty} I(v,w,x,x+\mathrm{d}x,y,y+\mathrm{d}y) \times I^{r-i-1}(v,w,-\infty,x,-\infty,y) \mathrm{e}^{\frac{-z^2}{2}} \mathrm{d}z.$$

To simplify notation, we denote by c the value $\frac{\alpha-\alpha'-v[n-1]\delta}{v[n]}$. As in section 6, let the coordinate system be so rotated that the new coordinates of v are $(b_1, 0, \ldots, 0)$ and the new coordinates of w are $(b_1', b_2', 0, \ldots, 0)$, where $b_1, b_2' \geq 0$. Recall from section 6 that

$$I(v,w,x,x+\mathrm{d}x,y,y+\mathrm{d}y) = \frac{1}{\sqrt{2\pi}} \frac{1}{b_1} \mathrm{e}^{\frac{-x^2}{2b_1{}^2}} \frac{1}{\sqrt{2\pi}} \frac{1}{b_2'} \mathrm{e}^{-\frac{1}{2}\left(\frac{y-\frac{x}{b_1}b_1'}{b_2'}\right)^2} \mathrm{d}y\mathrm{d}x.$$

Therefore,

$$g(\delta) = \int_{z=-\infty}^{c} \int_{x=\alpha}^{\infty} \int_{y=\beta}^{\infty} I^{r-i-1}(v,w,-\infty,x,-\infty,y) \frac{1}{(2\pi)b_1 b_2'} \mathrm{e}^{\frac{-x^2}{2b_1{}^2}} \mathrm{e}^{-\frac{1}{2}\left(\frac{y-\frac{x}{b_1}b_1'}{b_2'}\right)^2} \times \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-z^2/2} \mathrm{d}y\mathrm{d}x\mathrm{d}z.$$

We first consider innermost integral, that is, with respect to y. The term to be integrated is

$$I^{r-i-1}(v,w,-\infty,x,-\infty,y) \frac{1}{\sqrt{2\pi}b_2'} \mathrm{e}^{-\frac{1}{2}\left(\frac{y-\frac{x}{b_1}b_1'}{b_2'}\right)^2}.$$

The other terms are independent of y. Its derivative with respect to y is

$$\frac{1}{\sqrt{2\pi}b_2'}(r-i-1)I^{r-i-2}(v,w,-\infty,x,-\infty,y)\frac{\partial I(v,w,-\infty,x,-\infty,y)}{\partial y}\mathrm{e}^{-\frac{1}{2}\left(\frac{y-\frac{x}{b_1}b_1'}{b_2'}\right)^2}$$
$$-\frac{1}{\sqrt{2\pi}b_2'}I^{r-i-1}(v,w,-\infty,x,-\infty,y)\left(\frac{y-\frac{x}{b_1}b_1'}{{b_2'}^2}\right)\mathrm{e}^{-\frac{1}{2}\left(\frac{y-\frac{x}{b_1}b_1'}{b_2'}\right)^2}.$$

Now

$$\frac{\partial I(v,w,-\infty,x,-\infty,y)}{\partial y}=\frac{1}{2\pi b_2'}\int_{-\infty}^{\frac{x}{b_1}}\mathrm{e}^{-\frac{1}{2}z^2}\mathrm{e}^{-\frac{1}{2}\left(\frac{y-b_1'z}{b_2'}\right)^2}\mathrm{d}z=O\left(\frac{1}{b_2'}\right).$$

Observe that as the functions I, $x\mathrm{e}^{-x^2/2}$ are all bounded by constants, the value of the above derivative is bounded in absolute value by $O\left(\frac{(r-i-1)}{(b_2')^2}+\frac{1}{(b_2')^2}\right)$. Since $r-i-1\le n$, $b_1=\Omega(\frac{1}{n^2})$, b_2' is $\Omega(\frac{1}{n^4})$ by Lemma 6.3, the above derivative is bounded by $O(n^9)$.

The second innermost integral, i.e., the one with respect to x, is considered next. The function inside the integral is $h(x)\frac{1}{\sqrt{(2\pi)}b_1}\mathrm{e}^{\frac{-x^2}{2b_1^2}}$, where

$$h(x)=\int_{y=\beta}^{\infty}I^{r-i-1}(v,w,-\infty,x,-\infty,y)\frac{1}{\sqrt{2\pi}b_2'}\mathrm{e}^{-\frac{1}{2}\left(\frac{y-\frac{x}{b_1}b_1'}{b_2'}\right)^2}\mathrm{d}y.$$

Since $0\le I()\le 1$, $h(x)=O(1)$. The derivative with respect to x is

$$-\frac{x}{\sqrt{2\pi}b_1^3}\mathrm{e}^{-\frac{x^2}{2b_1^2}}h(x)$$
$$+\frac{1}{\sqrt{2\pi}b_1}\mathrm{e}^{-\frac{x^2}{2b_1^2}}\int_{\beta}^{\infty}(r-i-1)I^{r-i-2}(v,w,-\infty,x,-\infty,y)\frac{\partial I(v,w,-\infty,x,-\infty,y)}{\partial x}$$
$$\times\frac{1}{\sqrt{2\pi}b_2'}\mathrm{e}^{-\frac{1}{2}\left(\frac{y-\frac{x}{b_1}b_1'}{b_2'}\right)^2}\mathrm{d}y$$
$$+\frac{1}{\sqrt{2\pi}b_1}\mathrm{e}^{-\frac{x^2}{2b_1^2}}\int_{\beta}^{\infty}I^{r-i-1}(v,w,-\infty,x,-\infty,y)\frac{1}{\sqrt{2\pi}b_2'}\frac{b_1'(y-\frac{xb_1'}{b_1})}{(b_2')^2b_1}\mathrm{e}^{-\frac{1}{2}\left(\frac{y-\frac{x}{b_1}b_1'}{b_2'}\right)^2}\mathrm{d}y.$$

Here,

$$\frac{\partial I(v,w,-\infty,x,-\infty,y)}{\partial x}=\frac{1}{2\pi b_1}\mathrm{e}^{-\frac{x^2}{2b_1^2}}\int_{-\infty}^{\frac{y-xb_1'/b_1}{b_2'}}\mathrm{e}^{-\frac{1}{2}(z')^2}\mathrm{d}z'=O\left(\frac{1}{b_1}\right).$$

Since $x\mathrm{e}^{-\frac{x^2}{2}}, h(x), I()$ are all $O(1)$, $r-i-1\le n$, and

$$\int_{\beta}^{\infty}\frac{1}{b_2'}\mathrm{e}^{-\frac{1}{2}\left(\frac{y-\frac{x}{b_1}b_1'}{b_2'}\right)^2}\mathrm{d}y=O(1),\int_{\beta}^{\infty}\frac{b_1'(y-\frac{x}{b_1}b_1')}{b_1(b_2')^3}\mathrm{e}^{-\frac{1}{2}\left(\frac{y-\frac{x}{b_1}b_1'}{b_2'}\right)^2}\mathrm{d}y=O\left(\frac{b_1'}{b_1b_2'}\right)$$
$$=O\left(\frac{1}{b_1{b_2'}^2}\right),$$

the above derivative is bounded by $O(\frac{n}{{b_1}^2} + \frac{1}{{b'_2}^2 b_1^2}) = O(n^{12})$, by Lemma 6.3.

This leaves only the outermost integration, where the integrand is

$$\frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2}z^2} \int_{x=\alpha}^{\infty} h(x) \frac{1}{\sqrt{2\pi} b_1} \mathrm{e}^{-\frac{x^2}{2b_1^2}} \, \mathrm{d}x,$$

whose derivative with respect to z is $O(1)$.

Acknowledgments. We thank Naveen Garg, Kurt Mehlhorn, David Williamson, Michel Goemans, and Madhu Sudan for comments. We thank Aravind Srinivasan for reading a draft of the manuscript and for his detailed comments. We thank the anonymous referees for many comments which have improved the readability of this paper and also for pointing us to the tighter tail bounds for the normal distribution, which improves the running time of some parts of the algorithm.

REFERENCES

[1] N. Alon and N. Kahale, *Approximating the independence number via the θ-function*, Math. Programming, 80 (1998), pp. 253–264.

[2] N. Alon, P. Kelsen, S. Mahajan, and H. Ramesh, *Approximately coloring hypergraphs*, Nordic J. Comput. 3 (1996), pp. 425–439.

[3] A. Blum, *New approximation algorithms for graph coloring*, J. Assoc. Comput. Mach., 41 (1994), pp. 470–516.

[4] R. Boppana and M. Halldorsson, *Approximating maximum independent sets by excluding subgraphs*, BIT, 32 (1992), pp. 180–196.

[5] W. Feller, *Introduction to Probability Theory and its Applications*, Vol. 1, John Wiley, New York, 1966.

[6] U. Feige and M. Goemans, *Approximating the value of two prover proof systems, with applications to Max-2Sat and Max- Dicut*, Proc. 3rd Israeli Symposium on Theory of Computing and Systems, Tel Aviv, 1995, pp. 182–189.

[7] A. Frieze and M. Jerrum, *Improved approximation algorithms for Max k-Cut and Max Bisection*, Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci., Springer-Verlag, Berlin, 1995, pp. 1115–1165.

[8] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, New York, 1987.

[9] M. Goemans and D. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. Assoc. Comput. Mach., 42 (1996), pp. 1115–1145.

[10] M. Halldorsson, *A still better performance guarantee for approximate graph colouring*, Inform. Process. Lett. 45 (1993), pp. 19–23.

[11] D.S. Johnson, *Worst case behaviour of graph coloring algorithms*, in Proc. 5th South-Eastern Conference on Combinatorics, Graph Theory and Computing, Congressus Numeratium X, (1974), pp. 513–527.

[12] D. Karger, R. Motwani, and M. Sudan, *Approximate graph coloring by semidefinite programming*, Proc. 35th IEEE Symposium on Foundations of Computer Science, Santa Fe, NM, 1994, pp. 1–10.

[13] S. Khanna, N. Linial, and S. Safra, *On the hardness of approximating the chromatic number*, Proc. 2nd Israeli Symposium on Theory and Computing Systems, 1993, pp. 250–260.

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

[15] P. Raghavan, *Randomized Approximation algorithms in combinatorial optimization*, Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci., Springer-Verlag, New York, 1994, pp. 300–317.

[16] J. Spencer, *Ten Lectures on the Probabilistic Method*, SIAM, Philadelphia, PA, 1987.

[17] A. Wigderson, *Improving the performance guarantee of approximate graph coloring*, J. Assoc. Comput. Mach., 30 (1983), pp. 729–735.

[18] M. Yannakakis, *On the Approximation of Maximum Satisfiability*, in Proc. 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, 1992, SIAM, Philadelphia, PA, pp. 1–9.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1664–1688

THE DYNAMIC PARALLEL COMPLEXITY OF COMPUTATIONAL CIRCUITS*

GARY L. MILLER† AND SHANG-HUA TENG‡

Abstract. We establish connections between parallel circuit evaluation and uniform algebraic closure properties of unary function classes. We use this connection in the development of time-efficient and processor-efficient parallel algorithms for the evaluation of algebraic circuits. Our algorithm provides a nontrivial upper bound on the parallel complexity of the circuit value problem over $\{\mathbb{R}, \min, \max, +\}$ and $\{\mathbb{R}^+, \min, \max, \times\}$. We partially answer an open question of Miller, Ramachandran, and Kaltofen by showing that circuits over a polynomial-bounded noncommutative semiring and circuits over infinite noncommutative semirings with a polynomial-bounded dimension over a commutative semiring can be evaluated in polylogarithmic time in their size and degree using a polynomial number of processors. We also present an improved parallel algorithm for Boolean circuits.

Key words. algebraic computations, Boolean circuits, complexity, NC problems, parallel algorithms, the circuit value problem

AMS subject classifications. 05C50, 68R10

PII. S0097539795281724

1. Introduction.

The *circuit value problem* has been recognized as an important problem in parallel computation [3, 5, 7, 18]. In 1975, Ladner [5] showed that the problem of the evaluation of Boolean circuits is P-complete. Goldschlager [3] extended Ladner's result to monotone Boolean circuits and planar Boolean circuits.

In spite of the P-completeness result, NC algorithms have been developed for several restricted classes of circuits that arise in parsing, Huffman code generation, compiler optimization, and numeric and algebraic computation [7, 8, 9, 10, 16, 18]. In 1983, Valiant et al. [18] showed that if a circuit over $\{+, -, \times\}$ can be evaluated in C time sequentially, then with preprocessing it can be evaluated in parallel $O(\log d \log Cdn)$ time using $C^{O(1)}$ processors. Miller, Ramachandran, and Kaltofen [7] improved and simplified this result by showing that any circuit with size n and degree d (see section 2.2 for the definition) over a commutative semiring can be evaluated without preprocessing in $O(\log n \log nd)$ time using $M(n)$ processors, where $M(n)$ is the number of processors needed for multiplying $n \times n$ matrices over the semiring in $O(\log n)$ time. Ramachandran and Yang [12] showed that the general planar monotonic circuit value problem can be evaluated in parallel polylogarithmic time with a linear number of processors. Miller, Reif, and Teng [6, 9, 10] showed that trees over

*Received by the editors February 17, 1995; accepted for publication (in revised form) June 5, 1997; published electronically May 7, 1999.

http://www.siam.org/journals/sicomp/28-5/28172.html

†School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 (glmiller@theory.cs.cmu.edu). This research was supported in part by National Science Foundation grant CCR-9016641. Part of the work was performed while the author was at the Department of Computer Science, University of Southern California, Los Angeles, CA 90089-0782.

‡Deparment of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield, Urbana, IL 61801-2987, and Department of Computer Science, University of Minnesota, Minneapolis, MN 55455 (steng@cs.uiuc.edu). This research was supported by an NSF Career award (CCR-9502540) and an Alfred P. Sloan Research Fellowship. Part of the work was done while the author was at the Department of Computer Science, University of Southern California, Los Angeles, CA 90089-0782, and Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139.

$\{+, -, \times, \div\}$ and $\{\min, \max, +, -, \times, \div\}$ can be evaluated in parallel $O(\log n)$ time using n processors.

However, there is no similar result for noncommutative semirings [7], and we know of no previous result on the parallel evaluation of circuits with three or more operators—for example, circuits over $\{\mathbb{R}, \min, \max, +\}$ and $\{\mathbb{R}^+, \min, \max, \times\}$. In fact, Kosaraju [4] and Nisan [11] showed that in general it is impossible to perform circuit evaluation in polylogarithmic time for noncommutative circuits.

In this paper, we propose a systematic method that facilitates the design of processor-efficient parallel algorithms for the circuit value problem. Our method utilizes the *uniform closure properties* of certain unary function classes.

Using this method, we give the first nontrivial upper bound on the parallel complexity for evaluating circuits over $\{\mathbb{R}, \min, \max, +\}$ and $\{\mathbb{R}^+, \min, \max, \times\}$. In particular, we show that a circuit over $\{\mathbb{R}, \min, \max, +\}$ or $\{\mathbb{R}^+, \min, \max, \times\}$ of n nodes and degree d can be evaluated in $O(\log n \log dn)$ time, using $nM(n)$ processors, where the degree of a min–max-plus circuit $\mathcal{C}$ is defined as the *formal arithmetic degree* (see section 2.2) of the arithmetic circuit obtained from $\mathcal{C}$ by mapping min to $+$ and max and $+$ to $\times$. We also extend the previously known results of circuits over commutative semirings to several noncommutative semirings. For instance, if a noncommutative semiring is *polynomially bounded* (see section 7) or has a *polynomially bounded dimension* over a commutative semiring (also see section 7), then circuits over it can be evaluated in polylogarithmic time in its size and degree by using a polynomial number of processors. Our results partially answer an open question of Miller, Ramachandran, and Kaltofen [7].

We also consider *structured parallel computation.* We use the circuit value problem over *matrix rings* as an example. Let $\mathcal{CR} = \{\mathcal{D}_1, +, \times\}$ be a commutative semiring, and let $\mathcal{MR} = \{D, +, \times\}$ be a *matrix ring* of *dimension* m over $\mathcal{CR}$, where

$$\mathcal{D} = \{A^{m\times m} \mid A^{m\times m} \in \mathcal{D}_1, 1 \le i, j \le m\}.$$

$\mathcal{MR}$ is an infinite noncommutative semiring whenever $\mathcal{CR}$ is infinite. The best known result to this problem, achieved by Miller, Ramachandran, and Kaltofen [7], is that circuit $\mathcal{C}$ of size n and degree d can be evaluated in $O(\log \max(n, m) \log dn)$ time using $M(m^3 n)$ processors. They obtained this result by expanding the matrix operation to the underlying commutative ring operations and then applying the parallel circuit evaluation algorithm in [7]. We call a method of this kind *nonstructured* because it replaces the structured matrix operations with the lower-level operations. We present a *structured algorithm* based on our parallel method; our method does not replace matrix operations by lower-level underlying commutative ring operations. It reduces the processor count to $M(m^2 n)$ without increasing the time complexity, $O(\log \max(n, m) \log dn)$. Our structured algorithm can also be used to reduce the processor count for context-free language circuits when the size of the grammar is not a constant.

Using the *duality* property of Boolean algebra, we give an improved parallel algorithm for Boolean circuits. It evaluates a Boolean circuit in $O(\log n \log(\min(d_{\wedge\vee}, d_{\vee\wedge})))$ time, using $M(n)$ processors. We also show that there are Boolean circuits with exponential $d_{\wedge\vee}$ and $d_{\vee\wedge}$ that can be evaluated by our algorithm in polylogarithmic time, using $M(n)$ processors, where $d_{\wedge\vee}$ ($d_{\vee\wedge}$) is the degree of the Boolean circuit when $\vee$ ($\wedge$) is viewed as addition in the semiring formed by the Boolean algebra.

The parallel algorithms are designed on *parallel random access machines* (*PRAMs*), which are shared memory computers where many processors work together synchronously. The communication among processors is done through the information

exchange in a common random-access memory. Models in this family can be further partitioned according to the conventions for solving read and write conflict. The algorithms are developed on the CRCW model in which concurrent reads and concurrent writes to a cell in the common memory are allowed.

An extended abstract of this work appeared in [8]. In our preparation of this journal version, we have obtained several new results and also improved some of the original results.

2. Preliminaries. A *computation circuit* is a triple $(\mathcal{A}, \mathcal{F}, \mathcal{C}\}$ where the following hold:

1. $\mathcal{A} = \{\mathcal{D}, \odot_1, \ldots, \odot_k\}$ is an algebraic system whose *domain* is $\mathcal{D}$ and whose operator set is $\mathcal{OP} = \{\odot_1, \ldots, \odot_k\}$.
2. $\mathcal{F}$ is a set of unary functions over $\mathcal{D}$. We assume that $\mathcal{F}$ includes the identity function.
3. $\mathcal{C}$ is a labeled directed acyclic simple graph whose leaf nodes, the nodes with zero indegree, are labeled with values from $\mathcal{D}$, whose internal nodes are labeled with operators from $\mathcal{OP}$, and whose edges are labeled with unary functions from $\mathcal{F}$.

If there is an edge from u to v in $\mathcal{C}$, we call u a *child* of v and v a *parent* of u.

The *value* of a node v in $\mathcal{C}$, denoted by $value(v)$, is defined inductively.

- If v is a leaf, then $value(v)$ is equal to its label.
- If v is an internal node, labeled by $\odot$ and with children $w_1, \ldots, w_l$, then

$$value(v) = \odot(f_1(value(w_1)), \ldots, f_l(value(w_l))),$$

where $f_1, \ldots, f_k$ are the unary functions of edges $(w_1, v), \ldots, (w_k, v)$, respectively.

DEFINITION 2.1 (circuit value problem). *Given a computation circuit* $\{\mathcal{A}, \mathcal{F}, \mathcal{C}\}$, *compute the values of all its internal nodes.*

We consider $\mathcal{A}$ and $\mathcal{F}$ fixed and analyze the parallel complexity of the circuit value problem for a given circuit $\mathcal{C}$. To apply our parallel circuit evaluation algorithm, we require that the set of unary functions $\mathcal{F}$ satisfies certain closure properties. We will discuss these requirement in section 3.

2.1. Semirings. An algebraic system $\{\mathcal{D}, \oplus, \otimes\}$ is a *semiring* if $\oplus$ is associative and commutative, and $\otimes$ is associative and distributive over $\oplus$. If $\otimes$ is also commutative, then semiring $\{\mathcal{D}, \oplus, \otimes\}$ is a *commutative semiring*; otherwise, it is a *noncommutative semiring*. A semiring $\{\mathcal{D}, \oplus, \otimes\}$ is *finite* if $\mathcal{D}$ is finite.

For example, $\{R, +, \times\}$ is an infinite and commutative semiring.

Let $\mathcal{CR} = \{\mathcal{D}, \oplus, \otimes\}$ be a commutative semiring. A *matrix semiring* over $\mathcal{CR}$ of *dimension* k is the semiring $\mathcal{MR} = \{\mathcal{MD}, \oplus, \otimes\}$, where

$$\mathcal{MD} = \{A^{k \times k} \mid A_{ij} \in \mathcal{D}, 1 \leq i, j \leq k\}.$$

Clearly, $\mathcal{MR}$ is a noncommutative semiring with infinite domain whenever $\mathcal{CR}$ is infinite.

For example, the set of all k by k real matrices together with $+$ and $\times$ form an infinite matrix semiring of dimension k.

2.2. CE-algebras and CE-circuits. The algebraic systems considered in this paper is restricted to a family of algebraic systems called *CE-algebra*s. An algebraic system $\mathcal{A} = \{\mathcal{D}, \mathcal{OP}\}$ is a CE-algebra if there is one operator $\oplus \in \mathcal{OP}$ that is *associa-*

tive and commutative, and all other operators in $\mathcal{OP}-\{\oplus\}$ are *distributed* over $\oplus$. The operator $\oplus$ is called the *cheap operator* and rest of the operators are called *expensive operator*s. In the remainder of this paper, the operator set $\mathcal{OP}$ of a CE-algebra with $k+1$ operators is written as $\mathcal{OP} = \{\oplus, \otimes_1, \ldots, \otimes_k\}$.

For example, $\{R, +, \times\}$ is a CE-algebra, where $+$ is cheap and $\times$ is expensive. In general, every semiring $\{\mathcal{D}, \oplus, \otimes\}$ is a CE-algebra in which $\oplus$ is cheap and $\otimes$ is expensive. $\{\mathbb{R}, \min, \max, +\}$, $\{\mathbb{R}^+, \min, \max, +, \times\}$, and $\{\mathbb{R}^+, \min, \max, \times\}$ are other examples of CE-algebras, where we regard either min or max but not both as the cheap operator. A circuit over a CE-algebra is called a *CE-circuit*. We call a node of a CE-circuit an *expensive node* if it is labeled with an expensive operator. Similarly, we call a node with a cheap operator a *cheap node*. In most of our discussions, we will make the following two technical restrictions in order to simplify our presentation.

ASSUMPTION 2.2.

1. *The indegree of each expensive node is two.*
2. *There is no edge from an expensive node to another expensive node.*

In section 5, we will not impose those restrictions when we consider Boolean circuits.

The *degree* of a CE-circuit $\mathcal{C}$ is defined as the *formal arithmetic degree* of the arithmetic circuit obtained from $\mathcal{C}$ by relabeling each cheap node by $+$ and each expensive node by $\times$. In other words, we can define the degrees of nodes in a CE-circuit inductively.

- If v is a leaf, then $degree(v) = 1$.
- If v is an expensive internal node with children $w_1, \ldots, w_l$, then

$$degree(v) = \sum_{j=1}^{l} degree(w_j).$$

- If v is a cheap internal node with children $w_1, \ldots, w_l$, then

$$degree(v) = \max_{j=1}^{l} degree(w_j).$$

3. The general paradigm. In this section, we first review the parallel arithmetic circuit-evaluation algorithm of Miller, Ramachandran, and Kaltofen [7]. We then show how to extend their algorithm to CE-circuits. Our extension is based on certain closure properties of the unary functions.

3.1. Arithmetic circuits. We will refer to circuits over $\{\mathbb{R}, +, \times\}$ as arithmetic circuits. Following [7], we will consider only the following set of unary functions $\mathcal{F} = \{f(x) = ax | a \in \mathbb{R}\}$. We can express an arithmetic circuit $\mathcal{C}$ of n nodes with unary function set $\mathcal{F}$ by a matrix U when entries are given as

$$Z_{ij} = \begin{cases} z_{ij} & \text{if } (i,j) \text{ is an edge in } \mathcal{C} \text{ with unary function } f_{ij}(x) = z_{ij}x, \\ 0 & \text{if there no edge from } i \text{ to } j. \end{cases}$$

The Miller–Ramachandran–Kaltofen algorithm consists of repeated applications of three elementary procedures on the circuit and its matrix. At a high level, these procedures transform an arithmetic circuit to a circuit of smaller degree or depth on the same node set so that the value of each node is preserved in the new circuit.

The first procedure is `rake`. It simply evaluates each node in the circuit whose children are leaves and removes all incoming edges to make it a leaf.

FIG. 1. *Rake.*

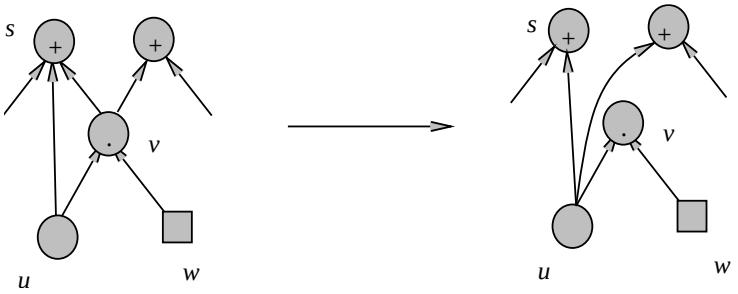


FIG. 2. *Shunt.*

PROCEDURE `rake`$(\mathcal{C}, Z)$ (Figure 1). For each node v whose children $w_1, \ldots, w_k$ are leaves,

1. if v is an addition node, then

$$value(v) = \sum_{j=1}^{k} Z_{w_j,v} value(w_j);$$

if v is a multiplication node (and hence $k = 2$), then

$$value(v) = (Z_{w_1,v} value(w_1))(Z_{w_2,v} value(w_2));$$

2. for each j in the range $1 \le j \le k$, remove the edge between v and w_i and let $Z_{w_j,v} = 0$.

The next procedure is `shunt`, which handles the case when a multiplication node v has one leaf child w and one nonleaf child u. It merges the contribution of w (namely, $Z_{vw} value(w)$) to a parent s of v into the edge from u to s. Thus u will bypass v in its connection to s. Note that v may have many parents.

PROCEDURE `shunt`$(\mathcal{C}, Z)$ (Figure 2). For each multiplication node v with children u and w in which one of u or w is a leaf (say, w),

1. let $h_{uv} = Z_{uv}(Z_{wv} value(w))$;
2. for each s such that v is a child of s (s has to be a cheap node by Assumption 2.2),
   - let $Z_{us} = Z_{us} + Z_{vs} h_{uv}$,
   - remove the edge (v, s) and let $Z_{vs} = 0$.

Finally, `compress` is used to transform long paths of addition nodes into paths of about half their lengths.

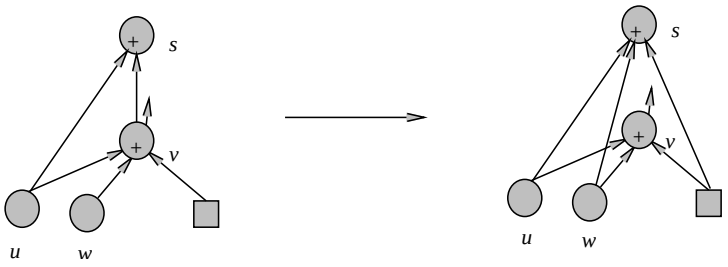


FIG. 3. *Compress.*

PROCEDURE `compress`$(\mathcal{C}, Z)$ (Figure 3). For each edge (v, s) in $\mathcal{C}$ such that both v and s are addition nodes,

1. for each child u of v, $Z_{us} = Z_{us} + Z_{vs}Z_{uv}$;
2. remove (v, s) and let $Z_{vs} = 0$.

Miller, Ramachandran, and Kaltofen [7] observed that one can use the following matrix expression for the `compress`:

$$Z = Z_{X,C}Z_{C,C} + U_{X,X},$$

where

$$Z(C,C)_{uv} = \begin{cases} Z_{uv} & \text{if both } u \text{ and } v \text{ are addition nodes,} \\ 0 & \text{otherwise;} \end{cases}$$

$$Z(X,C)_{uv} = \begin{cases} Z_{uv} & \text{if } v \text{ is an addition node,} \\ 0 & \text{otherwise;} \end{cases}$$

$$Z(X,X)_{uv} = \begin{cases} Z_{uv} & \text{if either } u \text{ or } u \text{ is a multiplication node,} \\ 0 & \text{otherwise.} \end{cases}$$

We now summarize their main result [7].

THEOREM 3.1 (Miller–Ramachandran–Kaltofen [7]). *An arithmetic circuit of n nodes and degree d can be evaluated in $O(\log n \log dn)$ time, using $M(n)$ processors.*

3.2. CE-circuits. In this subsection, we will extend the Miller–Ramachandran–Kaltofen algorithm from arithmetic circuits to CE-circuits. The high-level idea is the same as for arithmetic circuits. We will simplify a CE-circuit by repeated applications of three elementary procedures: `rake`, `shunt`, and `compress`. These procedures transform an arithmetic circuit to a circuit of smaller degree or depth on the same node set so that the value of each node is preserved in the new circuit.

We now define these three elementary procedures for CE-circuits. We consider a CE-algebra $\{\mathcal{D}, \oplus, \otimes_1, \ldots, \otimes_k\}$. Like an arithmetic circuit, a CE-circuit can be represented by a matrix Z whose entries are the unary functions used in the circuit. The entry Z_{uv} is defined as

$$Z_{uv} = \begin{cases} f_{uv} & \text{if } (u, v) \text{ is an edge with label } f_{uv}, \\ 0 & \text{otherwise,} \end{cases}$$

where 0 is the *zero function*.

The `rake` is the same as in arithmetic circuit. It evaluates each node in the circuit whose children are leaves and removes all incoming edges to make it a leaf.

PROCEDURE `rake`$(\mathcal{C}, Z)$. For each node v whose children $w_1, \ldots, w_k$ are leaves, let $\odot$ be the operator of v. Then

1. $$value(v) = \odot(Z_{w_1,v}(value(w_1)), \ldots, Z_{w_k,v}(value(w_k)));$$
2. for each j in the range $1 \leq j \leq k$, remove the edge between v and w_i and let $Z_{w_j,v} = 0$.

The procedure `shunt` handles the case when an expensive node v has one leaf child w and one nonleaf child u.

PROCEDURE `shunt`$(\mathcal{C}, Z)$. For each expensive node v with children u and w in which one of u or w is a leaf, let $t \in \{u, w\}$ denote the nonleaf child and $\otimes$ be the label of v. Then

1. if u is a leaf, let $h_{tv}(x) = Z_{uv}(value(u)) \otimes Z_{wv}(x)$, and if w is a leaf, let $h_{tv}(x) = Z_{uv}(x) \otimes Z_{uv}(value(w))$;
2. for each s such that v is a child of s (s must be an addition node by Assumption 2.2),
   - let $Z_{ts}(x) = Z_{ts}(x) \oplus Z_{vs}(h_{tv}(x))$,
   - remove the edge (v, s) and let $Z_{vs} = 0$.

Finally, `compress` is used to transform long paths of cheap nodes into paths of about half their lengths.

PROCEDURE `compress`$(\mathcal{C}, Z)$. For each edge (v, s) in $\mathcal{C}$ such that both v and s are cheap nodes,

1. for each child u of v, $Z_{us}(x) = Z_{us}(x) \oplus Z_{vs}(Z_{uv}(x))$;
2. remove (v, s) and let $Z_{vs} = 0$.

We can also describe `compress` with the following matrix expression:

$$Z = Z_{X,C} \circ Z_{C,C} \oplus U_{X,X},$$

where

$$Z(C,C)_{uv} = \begin{cases} Z_{uv} & \text{if both } u \text{ and } v \text{ are cheap nodes,} \\ 0 & \text{otherwise;} \end{cases}$$

$$Z(X,C)_{uv} = \begin{cases} Z_{uv} & \text{if } v \text{ is a cheap node,} \\ 0 & \text{otherwise;} \end{cases}$$

$$Z(X,X)_{uv} = \begin{cases} Z_{uv} & \text{if either } u \text{ or } u \text{ is an expensive node,} \\ 0 & \text{otherwise;} \end{cases}$$

and $\circ$ is the composition operator, i.e., for each pair of f and g in $\mathcal{F}$, $f \circ g(x) = f(g(x))$.

Lemma 3.2 follows directly from the definition of these three procedures.

LEMMA 3.2. *The application of* `rake`, `shunt`, *and* `compress` *preserves the values of all nodes in the circuit.*

Note that if the input circuit is an arithmetic circuit, then the above parallel circuit evaluation algorithm is equivalent to the parallel arithmetic circuit evaluation algorithm of Miller, Ramachandran, and Kaltofen [7]. Therefore, $O(\log n \log dn)$ applications `rake`, `shunt`, and `compress` will evaluate an arithmetic circuit of n nodes and degree d [7].

Our circuit evaluation algorithm can now be specified as follows.

ALGORITHM CIRCUIT-REDUCTION$(\mathcal{C}, Z)$

1. `rake`$(\mathcal{C}, Z)$.
2. `shunt`$(\mathcal{C}, Z)$.
3. `compress`$(\mathcal{C}, Z)$.

3.3. Unary functions and their closure properties. The basic idea of the Miller–Ramachandran–Kaltofen algorithm is to use `rake`, `shunt`, and `compress` to partially evaluate some subcircuits and reduce them to equivalent unary functions. It then represents these functions as weights on the edges of the circuit. The correctness of the algorithm relies on the fact that the unary function class $\{f(x) = ax | a \in \mathbb{R}\}$ is closed under composition; i.e., if $f(x) = ax$ and $g(x) = bx$, then $f \circ g(x) = (ab)x$, which is still a function in $\mathcal{F}$. Moreover, it takes a constant time to compute the composition.

The basic idea of our algorithm is the same: `shunt` computes a unary function $Z_{ts}(x) = Z_{ts}(x) \oplus Z_{vs}(h_{tv}(x))$ and writes it on edge (t, s); `compress` computes and writes a unary function $Z_{us}(x) = Z_{us}(x) \oplus Z_{vs}(Z_{uv}(x))$ on edge (u, s).

The idea behind our algorithm is to reduce the circuit value on CE-circuits to a circuit value problem over the semiring $\{\mathcal{F}, \oplus, \circ\}$. Hence we can apply the analysis from [7].

The correctness of our algorithm is based on the following closure properties:

- *Composition:* A unary function class $\mathcal{F}$ is *closed under composition* if for all $f_1, f_2 \in \mathcal{F}$, $f_2 \circ f_1 \in \mathcal{F}$.
- *Combination:* A unary function class $\mathcal{F}$ is *closed under combination* over a binary, associative, and commutative operator $\oplus$ if for all $f, g \in \mathcal{F}$, $f(x) \oplus g(x) \in \mathcal{F}$, where $\oplus$ of two unary functions $f(x)$ and $g(x)$ is $F(x) = f(x) \oplus g(x)$.
- *Projection:* A unary function class $\mathcal{F}$ is *closed under projection* over a set of binary operators $\mathcal{OP}$ if for all $f \in \mathcal{F}$, $\odot \in \mathcal{OP}$, and $a \in \mathcal{D}$, $x \odot a \in \mathcal{F}$ and $a \odot x \in \mathcal{F}$, where x is a variable with domain $\mathcal{D}$.
- *Linear:* A unary function class $\mathcal{F}$ is *linear over* an operator $\oplus$ if for all $f \in \mathcal{F}$, $x, y \in \mathcal{D}$, $f(x \oplus y) = f(x) \oplus f(y)$.

Clearly, if $\oplus$ is the cheap operator in a CE-algebra and $\mathcal{F}$ is a set of unary functions which is linear over $\oplus$, then the *closure* of $\mathcal{F}$ under composition, combination over $\oplus$, and projection over the expensive operations is still linear over $\oplus$.

DEFINITION 3.3. *A class of unary functions $\mathcal{F}$ is closed over $\mathcal{OP} = \{\oplus, \otimes_1, \ldots, \otimes_k\}$ if $\mathcal{F}$ is closed under composition, combination over $\oplus$, linear over $\oplus$, and projection over $\{\otimes_1, \ldots, \otimes_k\}$.*

LEMMA 3.4. *If $\mathcal{F}$ is linear over $\oplus$ and closed under combination over $\oplus$, then $\{\mathcal{F}, \oplus, \circ\}$ forms a semiring.*

Proof. Let f, g, h be functions in $\mathcal{F}$. Because $\oplus$ is communicative, $f(x) \oplus g(x) = g(x) \oplus f(x)$. By the linearity of $\mathcal{F}$, we have $f(g(x) \oplus h(x)) = f(g(x)) + f(h(x))$ and hence $\circ$ is distributive over $\oplus$. ∎

To efficiently support our algorithm, we need to show that composition and projection over $\{\mathcal{F}, \oplus, \circ\}$ can be computed efficiently. Moreover, we need to represent these functions economically. In arithmetic circuit evaluation, each unary function in $\{f(x) = ax | a \in \mathbb{R}\}$ can be expressed by a single real number and can be evaluated in a constant time. However, for the CE-circuits that we will consider, the complexity of the unary functions may grow after each application of `rake`, `shunt`, and `compress`. Therefore, we need to show that they do not grow too fast.

The class of unary functions that we use can be defined by a restricted class of CE-circuits.

DEFINITION 3.5. *A* restricted CE-circuit *in one free variable x is a CE-circuit C in which*

1. *there exists a unique leaf node whose value is the variable x, and*
2. *every expensive node has at most two children, one of which is a leaf.*

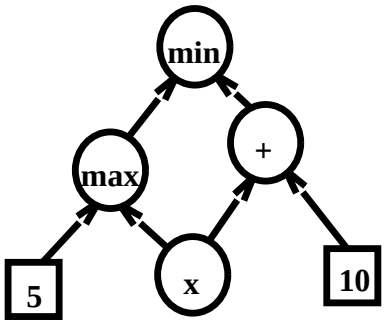


FIG. 4. *A restricted circuit that defines* $f(x) = \min(\max(5, x), (10 + x))$.

A restricted CE-circuit C defines a unary function f at each of its nodes v in variable x. If C has a distinguished output node whose function is $f(x)$, then f is called the function of C, and we denote it by f_C.

A unary function f is a *restricted CE-function* over an operator $\mathcal{OP}$ and a unary function class $\mathcal{F}$, if there is a restricted CE-circuit over $\mathcal{OP}$ and $\mathcal{F}$ that defines f (see Figure 4).

Let $\mathcal{RCE}_{(\mathcal{OP},\mathcal{F})}$ denote all restricted CE-functions over $\mathcal{OP}$ and $\mathcal{F}$. For simplicity, let $\mathcal{RCE}_{\mathcal{OP}}$ denote all restricted CE-functions over $\mathcal{OP}$ and $\{identity\ function\}$. By definition, we have the following proposition.

PROPOSITION 3.6. *$\mathcal{RCE}_{\mathcal{OP}}$ is closed over $\mathcal{OP}$.*

The *size* of a restricted CE-circuit is the number of nodes and edges in the circuit. The *circuit size* of a restricted CE-function f is defined to be the minimum size of the restricted CE-circuit that defines f.

DEFINITION 3.7 (uniform closure properties). *$\mathcal{F}$ is (T, P)-uniform closed over $\mathcal{OP}$ if $\mathcal{F}$ is closed over $\mathcal{OP}$, and for all $f \in \mathcal{F}$ of circuit size n, f can be evaluated in $O(T(n))$ time, using $P(n)$ processors. The composition and combination of two functions f and $g \in \mathcal{F}$ of circuit size no more than n can be computed in $O(T(n))$ time, using $P(n)$ processors, where $T(n)$ and $P(n)$ are two integer functions.*

LEMMA 3.8. *Let $\mathcal{C}$ be a CE-circuit of n nodes and degree d. Then $O(\log n \log dn)$ applications of circuit-reduction will evaluate $\mathcal{C}$.*

Proof. If we replace each cheap operator with $+$ and all expensive operators with $\times$, we obtain an arithmetic circuit $\mathcal{A}(\mathcal{C})$ which also has n nodes and degree d. If $\mathcal{C}_1$ is the circuit obtained by applying circuit-reduction to $\mathcal{C}$ and A_1 is the circuit obtained by applying the circuit-reduction to $\mathcal{A}(\mathcal{C})$, then $\mathcal{A}_1 = \mathcal{A}(\mathcal{C}_1)$. Hence, this lemma follows directly from Theorem 3.1. □

Therefore, we have the following theorem.

THEOREM 3.9 (circuit theorem). *Let $\mathcal{C}$ be a CE-circuit of size n and degree d over operator set $\mathcal{OP}$. If there exists a unary function class $\mathcal{F}$ which is (T, P)-uniformly closed over $\mathcal{OP}$, then $\mathcal{C}$ can be evaluated in $O(T(n) \log n \log nd)$ time, using $P(n)M(n)$ processors.*

Proof. Because $\mathcal{F}$ is (T, P)-uniformly closed over $\mathcal{OP}$, `rake` can be performed in $O(T(n))$ time, using $P(n)n$ processors, and `shunt` can be performed in $O(T(n))$ time, using $P(n)n^2$ processors. In addition, because $\{\mathcal{F}, \oplus, \circ\}$ is a semiring, `compress` can be performed in $O(T(n) \log n)$ time, using $P(n)M(n)$ processors. Therefore, the theorem follows from Lemma 3.8. □

4. A lemma on closure properties. As we have shown in the previous section, the correctness of our algorithm relies on the closure properties of the unary function classes, while the efficiency of our algorithm depends on the *uniformity* of the unary function classes. Thus, it is important to develop a general theory and a systemic method to study the closure properties and to prove the uniformity of the classes. The following is a simple yet important lemma for proving the closure properties of certain unary function classes.

LEMMA 4.1 (closure property lemma). *If $\mathcal{F} = \{L(x)\}$ is a class of monotonically increasing unary functions over a domain $\mathcal{D} \subseteq \mathbb{R}$ and is closed under composition and projection over $\mathcal{OP}_1$, then*

$$\mathcal{F}_{\min} = \{\min(L_1(x), \ldots, L_i(x)) \mid i \in \mathcal{N}, L_j(x) \in \mathcal{F}_1\}$$

is closed over $\{\min, \mathcal{OP}_1\}$, and

$$\mathcal{F}_{\max} = \{\max(L_1(x), \ldots, L_i(x)) \mid i \in \mathcal{N}, L_j(x) \in \mathcal{F}_1\}$$

is closed over $\{\max, \mathcal{OP}_1\}$.

Proof. $\mathcal{F}_{\min}$ is closed under composition because for all $f(x) = \min(L_1(x), \ldots, L_i(x)) \in \mathcal{F}_{\min}$ and $g(x) = \min(L'_1(x), \ldots, L'_j(x)) \in \mathcal{F}$, since all functions in $\mathcal{F}_1$ are monotonically increasing, and $\mathcal{F}_i$ is closed under composition,

$$\begin{aligned} g \circ f(x) &= \min(L'_1(f(x)), \ldots, L'_j(f(x))) \\ &= \min(L'_1(L_1(x)), \ldots, L'_1(L_i(x)), \ldots, L'_j(L_i(x))) \in \mathcal{F}_{\min}. \end{aligned}$$

$\mathcal{F}_{\min}$ is closed under combination over $\{\min\}$ because

$$\min(f, g) \;=\; \min(L_1(x), \ldots, L_i(x), L'_1(x), \ldots, L'_j(x)) \in \mathcal{F}_{\min}.$$

$\mathcal{F}_{\min}$ is linear over $\{\min\}$ because all functions in $\mathcal{F}_1$ are monotonically increasing.

$\mathcal{F}_{\min}$ is closed under projection over $\mathcal{OP}_1$ because $\mathcal{F}_1$ is closed under projection under $\mathcal{OP}_1$.

The second part of the lemma is proved by duality. ∎

The following are some consequences of Lemma 4.1. Those facts will be used later in this paper.

- $\mathcal{F}_{(\min,\max,+)} = \{\min(L_1(x), \ldots, L_i(x)) \mid i \in \mathcal{N}, L_1, \ldots, L_i \in \mathcal{F}_{(\max,+)}\}$ is closed over $\{\min, \max, +\}$, where $\mathcal{F}_{(\max,+)} = \{\max(x + a, b) \mid a, b \in \mathbb{R}\}$.
- $\mathcal{F}_{(\min,\max,\times)} = \{\min(L_1(x), \ldots, L_i(x)) \mid i \in \mathcal{N}, L_1, \ldots, L_i \in \mathcal{F}_{(\max,\times)}\}$ is closed over $\{\min, \max, \times\}$, where $\mathcal{F}_{(\max,\times)} = \{\max(ax, b) \mid a, b \in \mathbb{R}^+\}$.
- $\mathcal{F}_{(\min,\max,+,\times)} = \{\min(L_1(x), \ldots, L_i) \mid i \in \mathcal{N}, L_1, \ldots, L_i \in \mathcal{F}_{(\max,+,\times)}\}$ is closed over $\{\min, \max, +, \times\}$, where $\mathcal{F}_{(\max,+,\times)} = \{\max(a, b \cdot x + c) \mid a, b, c \in \mathbb{R}^+\}$.

5. Min–max-plus circuits. We know of no previous results on the parallel evaluation of circuits with three or more operators. In this section, we consider circuits over $\{\mathbb{R}, \min, \max, +\}$. Our result can be extended directly to circuits over $\{\mathbb{R}^+, \min, \max, \times\}$. Min–max-plus circuits (and even min–max-plus trees) are used in computational artificial intelligence (AI) and game theory. Moreover, several dynamic programming problems can be reduced to the min–max-plus circuit value problem. Note that a circuit over any pair of operators is easy to evaluate in parallel, since $\{\mathbb{R}, \min, \max\}$ forms a Boolean algebra; and $\{\mathbb{R}, \min, +\}$ and $\{\mathbb{R}, \max, +\}$ form commutative semirings.

5.1. Unary functions. It is natural to view $\{\min, \max\}$ as the cheap operators and $\{+\}$ as the expensive operators. However, if $\mathcal{NC} \neq \mathcal{P}$, then there is no class of unary functions which is uniformly closed over $\{\min, \max, +\}$. By this observation, min (or max) is taken as the cheap operator.

The class of unary functions $\mathcal{F}$ for min–max-plus is defined as follows.

DEFINITION 5.1 (unary function). *Let $L_{a,b}$ denote the linear form $\max(a+x, b) = (a+x) \wedge b$, where we use $\wedge$ to denote* max*; for all $i \in \mathcal{N}$, let $L_{a_1,b_1} \vee \cdots \vee L_{a_i,b_i}$ denote $\min(L_{a_1,b_1}(x), \ldots, L_{a_i,b_i}(x))$. The class of unary functions of min–max-plus is defined as*

$$F = \bigcup_{i=1}^{+\infty} \{L_{a_1,b_1} \vee \cdots \vee L_{a_i,b_i} \mid a_1, \ldots, a_i, b_1, \ldots, b_i \in R\}.$$

This is called the linear form representation *of f. The* size of the representation *is the number of linear forms presented in the representation. The* minimum representation *of f is a linear form representation containing the minimum number of forms.*

Note that the unary function class $\mathcal{F}$ defined above is equal to the $\mathcal{F}_{(\min,\max,+)}$ defined in section 4. Therefore, we have the following lemma.

LEMMA 5.2. *$\mathcal{F}$ is closed over $\{\min, \max, +\}$.*

These unary functions allow the partial evaluation of parts of the circuit so that certain subcircuits can be replaced by a single edge that has a unary function on it. These subcircuits are restricted CE-circuits defined in section 3; the unary functions used to replace subcircuits are restricted CE-functions defined in section 3.

The main question considered in this and the next section is the size (to be defined in the next subsection) of a restricted CE-function f as a function of the size of the original circuit. This size will affect the time and number of processors needed to evaluate the circuit.

LEMMA 5.3. *Any unary function f produced during the evaluation of a CE-circuit $\mathcal{C}$ by repeated applications of the circuit-reduction algorithm is computable by a restricted CE-circuit obtained from $\mathcal{C}$ by* `rake`*,* `compress`*, and* `shunt`*. Thus, f is computable by a restricted circuit of size at most $|C|$.*

Proof. We will prove the lemma by induction on the number of applications of the circuit-reduction algorithm performed on the circuit. If no application of the procedure is performed, the lemma is trivially true. Circuit-reduction is performed by sequentially applying `rake`, `shunt`, and `compress`, where `rake` simply deletes parts of the circuit, `shunt` partially evaluates some part of the circuit, and `compress` combines two subcircuits under composition to get each unary function. The lemma then follows from the induction hypothesis. □

5.2. Uniformity of $\mathcal{F}$.

DEFINITION 5.4 (size of functions in $\mathcal{F}$). *The size of a function $f \in \mathcal{F}$ is the number of forms in its minimum linear form representation, denoted size(f).*

During the evaluation of the circuit, we need to reduce unary functions to its minimum linear form representation.

LEMMA 5.5. *A linear form representation of size n can be reduced to a minimum in $O(\log n)$ time, using n processors.*

Proof. Let $L_{a_1,b_1} \vee \cdots \vee L_{a_n,b_n}$ be an arbitrary linear form representation. We first sort the forms in increasing order by their $b_i - a_i$ values. It takes $O(\log n)$ time, using n processors [1, 13]. Without loss of generality, we assume that no two forms have the same $b_i - a_i$ value. Now we assume that $L_{a_1,b_1} \vee \cdots \vee L_{a_n,b_n}$ are sorted as above.

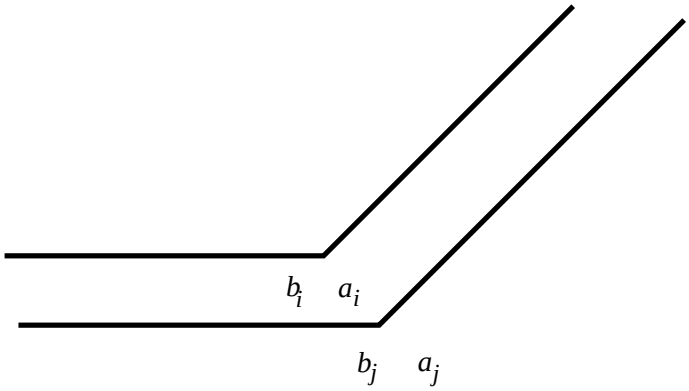


FIG. 5. *The case when* $b_i - a_i < b_j - a_j$.

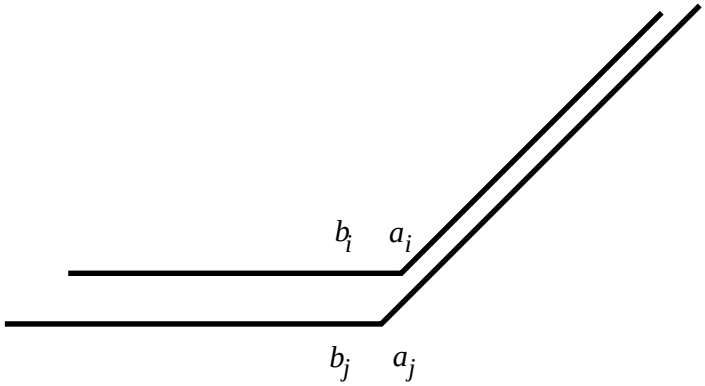


FIG. 6. *The case when* $b_i - a_i > b_j - a_j$.

There are two cases by which L_{a_i,b_i} can be eliminated by L_{a_j,b_j} in the minimum representation:

1. When $b_i - a_i < b_j - a_j$ (Figure 5), then L_{a_i,b_i} is eliminated by L_{a_j,b_j} if and only if $b_j < b_i$.
2. When $b_i - a_i > b_j - a_j$ (Figure 6), then L_{a_i,b_i} is eliminated by L_{a_j,b_j} if and only if $a_i < a_j$.

It follows that a linear form L_{a_i,b_i} is in the minimum representation if a_i is strictly smaller than a_j for $1 \leq j < i$ and b_i is strictly larger than a_j

Thus, by a prefix max of $(a_1, \ldots, a_n)$ we can determine the first case in $O(\log n)$ time using $n/\log n$ processors. Thus, by a prefix min of $(b_n, \ldots, b_1)$ we can determine the second case in $O(\log n)$ time by using $n/\log n$ processors. Hence, the minimum representation can be determined in $O(\log n)$ time by using n processors. □

Given a sorted minimum representation for a function f, it follows that the value of f at x can be computed in constant time, using n processors, or in $O(\log n)$ time, using one processor. We now show that the minimum representation of unary functions used during the evaluation of the circuit is not too large. Let the *breakpoints* of a linear form function f be the set of $(b - a)$'s in the minimum representation of f.

DEFINITION 5.6. *The* breakpoints *of a restricted CE-circuit C for the operations* $\{\min, \max, +\}$ *are the union over the breakpoints of all functions computed by C.*

LEMMA 5.7. *The number of breakpoints of a restricted CE-circuit that occur over* $\{\min, \max, +\}$ *is bounded by the sum of the number of max nodes and the total number of breakpoints of the edge functions of* $\mathcal{C}$.

Proof. We will prove only the case when the functions on the edge are all identity functions. If a function on the edge has p breakpoints, then we can replace the edge by a restricted CE-circuit of size $O(p)$ where all edges have identify functions. Thus, our proof can be directly extended to the general case. Note that if f has p breakpoints and g has q breakpoints, then $g \circ f$ has at most $p + q$ breakpoints. This statement was proved in Lemma 7.3 in [9]. See also Lemma 5.9.

We will prove the lemma by an induction on the size of the restricted CE-circuit. A single node is a restricted CE-circuit, and the only function it computes is a constant function. Thus, it has no breakpoints, as stated in the lemma.

Suppose that the lemma is true for all circuits of size n or less. We now prove the lemma for circuits of size $n + 1$. Let C be such a circuit of size $n + 1$ and v be one of its output nodes. If v is removed from C and all the edges associated with v are removed, then a circuit C' of size n is obtained, which is possibly a disconnected circuit. By induction, the lemma holds for C'. To prove the lemma for C, we consider the following three cases, depending on whether v is a min, max, or plus node. Let $v_1, \ldots, v_k$ be the children of v.

First, suppose that v is a min node. In this case the value computed at v is just the min of the values computed at $v_1, \ldots, v_k$. Thus, the breakpoints of v are at most the union of the breakpoints of $v_1, \ldots, v_k$. Thus, no new breakpoint has been introduced by v.

Second, suppose that v is a max node. In this case v has at most two children, v_1 and v_2, where v_1 is a leaf with value a and v_2 computes the value of $f(x)$. It follows that the max of the constant function a with f will introduce at most one new breakpoint, the point where curve a intersects the monotone increasing curve f. If they intersect in a line, then no breakpoint is introduced. Since C has one more max node than C' this case also follows.

Third, suppose that v is a plus node. Here, the value of v is the sum of a constant a and a function f. But, the breakpoints of f are unchanged by translation of f by an additive constant. Thus, the breakpoints of the function computed by v are just the breakpoints of f. Therefore, the breakpoints of C are the same as the breakpoints of C'. □

LEMMA 5.8. *If* $\mathcal{C}$ *is a* min–max-plus *circuit with* m *max nodes and* e *edges and all unary functions on the edges of* $\mathcal{C}$ *have sizes bounded by a constant* c*, then the size of a unary function used by circuit-reduction during the evaluation of* C *will have size of at most* $ce + m$*. In the case where the edge functions are trivial, the size of the functions are all bounded by* m.

Proof. By Lemma 5.3 every unary function is computed by a subcircuit of C. Furthermore, by Lemma 5.7 the number of breakpoints in any restricted CE-subcircuit is bounded by the number of breakpoints contributed by the edge function, which is at most $c \cdot e$, plus the number of max nodes, m. □

It follows from Lemma 5.8 that $\mathcal{F}$, the unary function class used for evaluating min–max-plus circuits, is $(\log n, n)$-uniformly closed over $\{\min, \max, +\}$. Therefore, by Theorem 3.9, circuits over $\{\min, \max, +\}$ of n nodes, m max nodes, and degree d can be evaluated in $O(\log^2 n \log dn)$ time, using $mM(n)$ processors.

In the following, it will be shown that the time count for evaluating a min–max-plus circuit can be reduced from $O(\log^2 n \log dn)$ to $O(\log n \log dn)$ without increasing

the processor count.

Note that the most time consuming step in *circuit-reduction* is the MM_C operation, where matrix–matrix multiplication of matrices whose entries are unary functions given in minimum linear form representation is performed. The product is over the semiring $\{\min, \mathit{composition}\}$. The matrices are assumed to be of size n, and the total number of breakpoints of all the functions, including those in the product, is bounded by n. Thus, the performance of composition in $O(\log n)$ time and n processors will be shown as well as the computation of the min of n functions in the same amount of time, using the same number of processors.

LEMMA 5.9. *If f and g are two functions of size at most n, then their composition and minimum representation can be computed in $O(\log n)$ time, using n processors. In general, the composition of any two monotone, piecewise linear functions each with n breakpoints can be computed in $O(\log n)$ time, using $n/\log n$ processors.*

Proof. See Lemma 7.3 in [9]. □

LEMMA 5.10. *The min of a set of n functions, each of size at most n, can be computed in $O(\log n)$ time, using n^2 processors.*

Proof. The min of n functions can be computed by first concatenating the representations of the n functions together and then applying the method in Lemma 5.5 to get a minimum representation for the answer. □

THEOREM 5.11. Min–max-plus *circuits with m max nodes, n nodes, and degree d can be evaluated in $O(\log n \log dn)$ time, using $mM(n)$ processors.*

Section 4 showed that $\mathcal{F}_{(\min,\max,\times)}$ is closed over $\{\min, \max, \times\}$. Using the similar uniformity proof in this section, we can prove $\mathcal{F}_{(\min,\max,\times)}$ to be $(\log n, n)$-uniformly closed over $\{\mathbb{R}^+, \min, \max, \times\}$.

THEOREM 5.12. Min–max-times *circuits over $\mathbb{R}^+$ with m max nodes, n nodes, and degree d, can be evaluated in $O(\log n \log dn)$ time, using $mM(n)$ processors.*

REMARK 5.13. *It can be shown that there is a class of unary functions which is $(\log n, n)$-uniformly closed over $\{\min, \max, +\}$ as well as $\{\max, \min, +\}$. Therefore, a parallel evaluation algorithm similar to the parallel Boolean circuit-evaluation algorithm (see next section) and which has dynamic adaptivity for evaluating min–max-plus circuits can be developed.*

6. Boolean circuits. We will present an algorithm for the parallel evaluation of Boolean circuits. The time complexity $T_{\mathcal{C}}$ of this algorithm always satisfies $T_{\mathcal{C}} \leq O(\log n \ \log n(\min(d_{\wedge\vee}, d_{\vee\wedge})))$. Moreover, there are Boolean circuits with exponential $d_{\wedge\vee}$ and $d_{\vee\wedge}$ which can be evaluated in polylogarithmic time using $M(n)$ processors according to our algorithm without the structure of the circuit being known in advance.

Boolean algebra $\mathcal{B} = \{\mathcal{D}, \vee, \wedge\}$ itself forms a commutative semiring; and therefore, as a consequence of results in [7], the simple Boolean circuit can be evaluated in $O(\log n \log nd)$ time with $M(n)$ processors. But this does not tighten the upper bound because the definition of the degree of a Boolean circuit is not clear, since either $\vee$ or $\wedge$ can be viewed as the addition operator in the semiring defined by the Boolean algebra. The corresponding degrees are denoted by $d_{\wedge\vee}$ and $d_{\vee\wedge}$, respectively.

One naive method to evaluate a Boolean circuit is to apply the algorithm in [7] to compute $d_{\wedge\vee}$ and $d_{\vee\wedge}$ first and then choose the semiring with smaller degree. Another method is first to make two copies of a Boolean circuit and then to apply the parallel algorithm from [7] to evaluate one circuit by viewing $\vee$ as the addition operator and to evaluate the other circuit by viewing $\wedge$ as the addition operator.

However, both methods are not uniform. In the first one, the evaluation of an arithmetic circuit has to be introduced, and it is not easier to compute the degree of a Boolean than to evaluate the circuit. In the second method, two evaluation processes have to be coordinated. If $d_{\wedge\vee}$ and $d_{\vee\wedge}$ are both exponential, no $\mathcal{NC}$ parallel algorithm can be deduced. Moreover, in order to use the parallel algorithm in [7], the indegrees of $\vee$-nodes or $\wedge$-nodes must be 2.

6.1. Closure properties.

LEMMA 6.1 (duality lemma). *For all $a, b \in \mathcal{D}$ in a Boolean algebra $\{\mathcal{D}, \vee, \wedge\}$, there is $c \in \mathcal{D}$, such that*

$$(a \wedge x) \vee b = (b \vee x) \wedge c.$$

LEMMA 6.2 (symmetric lemma). *$\mathcal{F} = \{(a \wedge x) \vee b \mid a, b \in \mathcal{D}\}$ is closed over $\{\vee, \wedge\}$ as well as $\{\wedge, \vee\}$.*

Proof. Due to the duality lemma, it suffices to prove that $\mathcal{F}$ is closed over $\{\vee, \wedge\}$.

Let $f_{(a,b)}(x) = (a \wedge x) \vee b$.

$\mathcal{F}$ is closed under composition, since $f_{(a_2,b_2)}(f_{(a_1,b_1)}(x)) = f_{(a_1 \wedge a_2, a_2 \wedge b_1 \vee b_2)} \in \mathcal{F}$.

$\mathcal{F}$ is closed under combination over $\vee$, since $f_{(a_1,b_1)} \vee f_{(a_2,b_2)} = f_{(a_1 \vee a_2, b_1 \vee b_2)} \in \mathcal{F}$.

$\mathcal{F}$ is linear over $\vee$, since $f_{(a,b)}(x \vee y) = f_{(a,b)}(x) \vee f_{(a,b)}(y)$.

$\mathcal{F}$ is closed under projection over $\wedge$, since for all $c \in \mathcal{D}$, $c \wedge x = (c \wedge x) \vee 0 \in \mathcal{F}$. □

6.2. The algorithm. Because of the duality lemma, there is no reason to view one of $\vee$ or $\wedge$ as the cheap operator and another as the expensive one. However, since the Boolean circuit value problem is $\mathcal{P}$-complete [5], one of them should be expensive. However, Lemma 6.2 implies that operations `shunt` and `compress` can be applied to both $\vee$-nodes and $\wedge$-nodes. Those operations are denoted $\texttt{shunt}_\vee$, $\texttt{shunt}_\wedge$, $\texttt{compress}_\vee$, and $\texttt{compress}_\wedge$, respectively.

We will use this symmetry in our algorithm, which applies two parallel evaluation phases. The first phase views $\vee$ as the expensive operation, while the second phase views $\wedge$ as the expensive operation. These two phases are denoted by $Phase(\wedge, \vee)$ and $Phase(\vee, \wedge)$, respectively. A Boolean circuit is evaluated by alternating applications of $Phase(\wedge, \vee)$ and $Phase(\vee, \wedge)$.

In order to make this work, we need to address the following three problems:

1. the problem caused by the unbounded fan-in of expensive nodes,
2. the problem caused by edges from expensive node to expensive node, and
3. the problem caused by alternation.

We will use the following operation to resolve the problem caused by the unbounded fan-in of expensive nodes.

PROCEDURE TRIMMING(U).

for all v whose children $w_1, \ldots, w_k$ are all leaves **do**
$v := \odot(U_{w_1 v}(w_1), \ldots, U_{w_k v}(w_k))$;;
$U_{w_i v} := 0$;; /* where $\odot$ is the operator of v */
for all v with children $w_1, \ldots, w_k$, w.l.o.g, $w_1, \ldots, w_t$ are leaves $(t < k)$ **do**
$a := \odot(U_{w_1 v}(w_1), \ldots, U_{w_t v}(w_t))$;;
$U_{w_{t+1} u}(x) := a \odot U_{w_{t+1} u}(x)$;;
$U_{w_1 v}, \ldots, U_{w_t v} := 0$;; /* where $\odot$ is the operator of v */

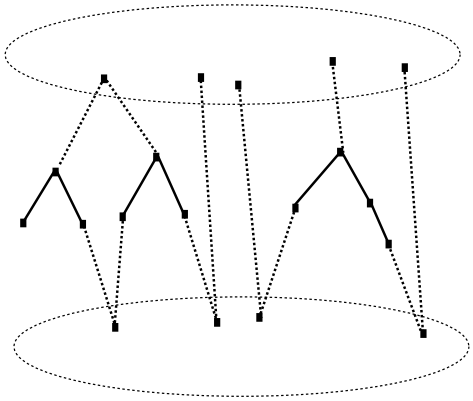


FIG. 7. *Forest formed by nodes of indegree* 1.

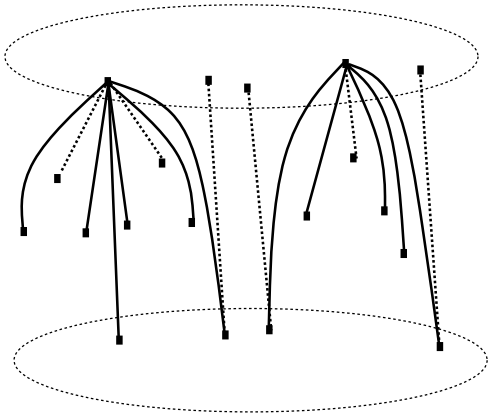


FIG. 8. *Forest removal for the example pictured in Figure* 1.

Graphically, the *Trimming* operation simply disconnects all leaves from their parents. This operation can be performed in $O(\log n)$ time, using n^2 processors. It can be easily proven that the *Trimming* operation preserves the value of each node in the circuit.

In general, *Trimming* will introduce many nodes with fan-in 1, and these nodes form a forest (subcircuit) in the circuit (see Figure 7).

The forest will be removed and replaced by equivalent unary functions. The equivalent unary functions of all nodes in the forest are the composition of the unary functions from the child of its root to itself (see Figure 8).

Using the parallel tree contraction technique [2, 6], the equivalent unary functions for all nodes in the forest can be found in $O(\log n)$ time, using only $n/\log n$ processors,

PROCEDURE FOREST-REMOVAL(U).
for all v in the forest **do**
find the equivalent unary function f_v;;
/* let r denote the root of v and $c(r)$ and $c(v)$ be the */
/* children of r and v in the circuit respectively */
$U_{c(v)v} := 0$;;

$U_{c(r)v} := f_v$;;
for all w and u where w is a child of some roots in the forest
and u is an $\odot$-node with some of its children $v_1, \ldots, v_k$
being nodes in the forest rooted by some parents of w **do**
$g_u := \odot_{i=1}^{k}(U_{v_i u} \circ f_{v_i})$;;
$U_{c(r)u} := U_{c(r)u} \odot g_u$;;
$U_{v_i u} := 0$;;

LEMMA 6.3. *The procedure Forest-Removal eliminates any node which is not an output node but has indegree* 1 *and preserves the value of each node. Moreover, Forest-Removal can be performed in* $O(\log n)$ *time, using* n^2 *processors.*

The following specifies the two phases of the parallel Boolean circuit evaluation algorithm.

Phase($\vee, \wedge$**)**
Trimming;; Trimming;; Trimming;;
Forest-Removal;; $\texttt{compress}_{\vee}$;;

Phase($\wedge, \vee$**)**
Trimming;; Trimming;; Trimming;;
Forest-Removal;; $\texttt{compress}_{\wedge}$;;

Algorithm Parallel Boolean Circuit Eval
Forest-Removal;;
repeat
Phase($\vee, \wedge$);; Phase($\wedge, \vee$);;
until all nodes are output nodes.

LEMMA 6.4. *After the application of Phase(*$\vee, \wedge$*) or Phase(*$\wedge, \vee$*), the indegrees of all the nonoutput nodes are at least two.*

Proof. Clearly, after the application of *Forest-Removal*, all nonoutput nodes have indegree at least 2. For all nonoutput nodes v, $\texttt{compress}_{\vee}$ and $\texttt{compress}_{\wedge}$ either make no change on v or connect all the children of some of its children to it. Hence, after the application of Phase($\vee, \wedge$) or Phase($\wedge, \vee$), v has at least two children. ∎

DEFINITION 6.5. *The* $\wedge\vee$*-height of a node* v*, denoted by* $\underline{h}_{\wedge\vee}(v)$*, in a Boolean circuit is defined inductively.*

- *If* v *is a leaf, then* $\underline{h}_{\wedge\vee}(v) = 1$.
- *If* v *is a* $\vee$*-node, then* $\underline{h}_{\wedge\vee}(v) = \sum_{u \in C(v)} \underline{h}_{\wedge\vee}(u)$.
- *If* v *is a* $\wedge$*-node, then*

$$\underline{h}_{\wedge\vee}(v) = \max\left(\max_{u \in CC(v)} \underline{h}_{\wedge\vee}(u) + 1/2, \max_{u \in EC(v)} (\underline{h}_{\wedge\vee}(u))\right),$$

where $C(v)$*,* $CC(v)$*, and* $EC(v)$ *are the set of children,* $\wedge$*-children, and* $\vee$*-children or leaf children of* v*, respectively.*

The $\wedge\vee$-height of a circuit is the maximum height over its nodes. A child w of a $\wedge$-node v is *dominant* if either w is a $\vee$-node and $\underline{\mathrm{h}}_{\wedge\vee}(w)=\underline{\mathrm{h}}_{\wedge\vee}(v)$ or w is a $\wedge$-node and $\underline{\mathrm{h}}_{\wedge\vee}(w)+1/2=\underline{\mathrm{h}}_{\wedge\vee}(v)$. The $\vee\wedge$-height, denoted by $\underline{\mathrm{h}}_{\vee\wedge}(v)$, is defined similarly.

The following lemma is proven as Theorem 4.2 in [7].

LEMMA 6.6. *The* $\wedge\vee$*-height (or* $\vee\wedge$*-height) of a Boolean circuit of* e *edges is bounded by* $ed_{\wedge\vee} + d_{\wedge\vee}$ *(*$ed_{\vee\wedge} + d_{\vee\wedge}$*).*

From now on, let $\mathcal{C}$ denote a Boolean circuit in which all nonoutput nodes have indegree at least 2.

LEMMA 6.7. *Let $\mathcal{C}'$ be the circuit after the application of Phase($\vee, \wedge$) on $\mathcal{C}$, and v' be the image of v. If v' is a leaf and there exists at least one $\wedge$-node u' such that (v', u') is an edge in $\mathcal{C}'$, then $\vee\wedge$-height of v is at least 2.*

Proof. Suppose that the $\vee\wedge$-height of v is less than 2: If its height is 1, then either v is a leaf or all its children are leaves. Hence, after the first *Trimming* operation, v becomes a leaf. If the height of v is 3/2, then v must be a $\vee$ node whose children are either leaves or $\vee$-nodes with all leaf children. Hence, the second *Trimming* makes v a leaf. Therefore, in each case, the application of the third *Trimming* disconnects v' and u'—a contradiction. □

LEMMA 6.8. *Let $\mathcal{C}'$ be the circuit after the application of Phase($\vee, \wedge$) on $\mathcal{C}$, and let v' be the image of v. Then $\underline{h}_{\vee\wedge}(v') \leq \underline{h}_{\vee\wedge}(v)/2$.*

Proof. The proof is done by induction on the *level* of v, the length of the longest path from a leaf to v. Suppose that all the children of v' are leaves.

- If v' is a $\wedge$-node, then by definition, the height $\underline{\mathrm{h}}_{\vee\wedge}(v')=k$, where k is the indegree of node v'. By Lemma 6.7, $\underline{\mathrm{h}}_{\vee\wedge}(v)\geq 2k$.
- If v is a $\vee$-node, then by definition, $\underline{\mathrm{h}}_{\vee\wedge}(v')=1$. Suppose that $\underline{\mathrm{h}}_{\vee\wedge}(v)\leq 3/2$; then after the second *Trimming* operation, v become a leaf—a contradiction.

Suppose that all the children of v' are either leaf nodes or internal nodes in C'.

- If v is a $\wedge$-node, then the lemma follows from Lemma 6.7 and the assumption.
- If v is a $\vee$-node and u' is a dominant child of v', then if u is a $\wedge$-node, clearly $\underline{h}_{\vee\wedge}(v') \leq \underline{h}_{\vee\wedge}(v)/2$. Now, suppose that u is a $\vee$-node. Since u and v are all $\vee$-nodes, there must be some other nodes on one of the paths from u to v; otherwise u' will not be a child of v' after the application of the `compress`$_{\vee}$ operation. Because all nodes have indegree at least 2, following from the definition of height, $\underline{\mathrm{h}}_{\vee\wedge}(u)+1\leq\underline{\mathrm{h}}_{\vee\wedge}(v)$. Hence, $\underline{\mathrm{h}}_{\vee\wedge}(v)\geq 2\cdot\underline{\mathrm{h}}_{\vee\wedge}(v')$. □

The following lemma can be proven by a simple induction on the level of nodes.

LEMMA 6.9. *For all nodes v in a Boolean circuit $\mathcal{C}$, the application of Phase($\vee, \wedge$) (Phase($\wedge, \vee$)) will not increase $\underline{h}_{\wedge\vee}(v)(\underline{h}_{\vee\wedge}(v))$.*

THEOREM 6.10. *Boolean circuits can be evaluated in time no more than $O(\log n \log n(\min(d_{\wedge\vee}, d_{\vee\wedge})))$, using $M(n)$ processors. Moreover, there are Boolean circuits with exponential $d_{\wedge\vee}$ and $d_{\vee\wedge}$ which can be evaluated in polylogarithmic time, using $M(n)$ processors, without the structure of the circuit being known in advance.*

Proof. The first part of the theorem is a consequence of Lemmas 6.8 and 6.9. The second part follows from the fact that for all nodes v in Boolean circuits $\mathcal{C}$, if at least one of $d_{\wedge\vee}(v)$ and $d_{\vee\wedge}(v)$ is no more than $O(2^{(\log n)^*})$, then v will get its value in polylogarithmic time. It is easy to construct Boolean circuits of exponential $d_{\wedge\vee}$ and $d_{\vee\wedge}$, but all nodes v in it have one of $d_{\wedge\vee}(v)$ and $d_{\vee\wedge}(v)$ is no more than $O(2^{(\log n)^*})$.

Figure 9 shows Boolean circuit with exponential $d_{\wedge\vee}$ and $d_{\vee\wedge}$ that can be evaluated in polylogarithmic time with $M(n)$ processors by our algorithm but not the Miller–Ramachandran–Kaltofen algorithm. □

The beauty of the above parallel algorithm lies in its dynamic adaptability to the structure of the circuit. This power comes from the symmetric structure of the unary function class. However, the tight bound on the running time of this parallel algorithm is still unknown, as are some stronger characterizations of the circuits which have $\mathcal{NC}$ solution. Nevertheless, this algorithm can be used to define the *dynamic degree* of a Boolean circuit.

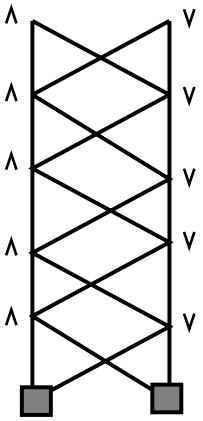


FIG. 9. *A Boolean circuit with exponential $d_{\wedge\vee}$ and $d_{\vee\wedge}$ that can be evaluated in polylogarithmic time with $M(n)$ processors by our algorithm but not the Miller–Ramachandran–Kaltofen algorithm.*

DEFINITION 6.11 (dynamic degree). *If Boolean circuit $\mathcal{C}$ can be evaluated by using the parallel Boolean circuit evaluation algorithm in $T_{\mathcal{C}}$ time, then the dynamic degree of $\mathcal{C}$ (denoted by $d_{\mathcal{C}}$) is defined as $d_C = 2^{T_C/(\log n)-\log n}$.*

7. Circuits over noncommutative semirings. In this section, we will extend the previously known results from circuits over commutative semirings to many noncommutative semirings. We will show that all circuits over *polynomial bounded* noncommutative semirings and circuits over infinite noncommutative semirings which have *polynomial bounded dimensions* over commutative semirings can be evaluated in polylogarithmic time in their size and degree using a polynomial number of processors. This provides a partial answer to the open problem raised in [7].

7.1. General case. Let $\mathcal{NR} = \{\mathcal{D}, \oplus, \odot\}$ be a noncommutative semiring. Let

$$\mathcal{F} = \{\Sigma_{i=1}^{k} a_i \odot x \odot b_i \mid k \in \mathcal{N}, a_i, b_i \in D\},$$

where $a_i \odot x \odot b_i$ is called an *item*. The *size* of a function in $\mathcal{F}$ is defined as the minimum number of items that define it.

LEMMA 7.1. *$\mathcal{F}$ is closed over $\{\oplus, \odot\}$.*

Let $\mathcal{INR} = \{\mathcal{D}, \oplus, \odot\}$, a noncommutative semiring with infinite domain $\mathcal{D}$. $\mathcal{INR}$ is *noncombinable* if for all $a, b, c, d \in \mathcal{D}$, there exists s and $t \in \mathcal{D}$ such that $(a \odot x \odot b) \oplus (c \odot x \odot d) = s \odot x \odot t$ if and only if $a = b$ or $c = d$. It is not difficult to give a constructive proof that for all noncombinable infinite noncommutative semirings, $\mathcal{F}$ is not uniformly closed over $\{\oplus, \odot\}$. That is, there exists a restricted CE-circuit $\mathcal{C}$ over $\mathcal{INR}$, which generates a function in $\mathcal{F}$ whose size is exponential in the size of the circuit. However, it will be shown that there are infinite noncommutative semirings such that all circuits over them can be evaluated in time logarithmic in their degree and size.

7.2. Finite case.

THEOREM 7.2. *For all finite, noncommutative semirings $\mathcal{FNR} = \{\mathcal{D}, \oplus, \odot\}$ and for all computational circuits $\mathcal{C}$ over $\mathcal{FNR}$ with size n and degree d, $\mathcal{C}$ can be evaluated in $O((\log n)(\log dn))$ time, using $M(n)$ processors on CRCW PRAMs.*

Proof. Let the unary function class for $\mathcal{C}$ be

$$\mathcal{F} = \{\Sigma_{i=1}^{k} a_i \odot x \odot b_i \mid k \leq \mid D \mid^2, a_i, b_i \in D\}.$$

It suffices to prove that $\mathcal{F}$ is closed over $(\{\oplus, \odot\})$. Clearly, $\mathcal{F}$ is linear over $\oplus$, and $\mathcal{F}$ is closed under projection over $\odot$. Let $f(x) = \Sigma_{i=1}^{k_1} a_i \odot x \odot b_i$ and $g(x) = \Sigma_{i=1}^{k_2} c_i \odot x \odot d_i$. $f \oplus g = (\Sigma_{i=1}^{k_1} a_i \odot x \odot b_i) \oplus (\Sigma_{i=1}^{k_2} c_i \odot x \odot d_i)$. If $k_1 + k_2 \leq \mid \mathcal{D} \mid^2$, then clearly $f \oplus g \in \mathcal{F}$. Otherwise, since $\mathcal{D}$ is finite and there are at most $\mid \mathcal{D} \mid^2$ different items, there must exist a $t \leq \mid \mathcal{D} \mid^2$, $u_1, \ldots, u_t, v_1, \ldots, v_t \in \mathcal{D}$, such that $f \oplus g = \Sigma_{i=1}^{k_1} u_i \odot x \odot v_i$. Hence, $\mathcal{F}$ is closed under combination. Similarly, we can prove that $\mathcal{F}$ is closed under composition. □

7.3. Examples of finite noncommutative rings. Let $G = \{V, \Sigma, P, S\}$ be a context-free grammar in Chomsky normal form. Let the domain be $\mathcal{D}_G = 2^{(V-\Sigma)}$ and operator set be $\mathcal{OP}_G = \{\cup, \lhd\}$, where $\cup$ is the conventional union operator in set theory and $\lhd$ is a binary operator such that for all $S, T \in \mathcal{D}$, $\lhd(S, T) = \{C \mid \exists A \in S, B \in T, C \to AB\}$. The circuits are defined over domain $\mathcal{D}_G$ and operator set $\mathcal{OP}_G$ and are called *CFL-circuits*.

It is clear that $\{\mathcal{D}_G, \cup\}$ forms a group and $\{\mathcal{D}_G, \lhd\}$ forms a semigroup. Since $\lhd$ is distributive over $\cup$, $\{\mathcal{D}_G, \cup, \lhd\}$ forms a finite, noncommutative semiring.

COROLLARY 7.3. *A CFL-circuit of size n and degree d can be evaluated in $O((\log n)(\log(nd)))$ time, using $M(n)$ processors.*

A *context-free language recognition problem* can be reduced to a *CFL-circuit* evaluation problem in polylogarithmic time since the corresponding *CFL-circuit* has a polynomial size and linear degree in the length of input string. This leads to the following corollary.

COROLLARY 7.4. *The context-free language recognition problem lies in $\mathcal{NC}$.*

Ruzzo [14, 15] was the first to show that there are Boolean circuits that simultaneously have polynomial size and polylogarithmic depth, which recognize context free languages. Valiant et al. [18] showed that the Boolean circuits defined by the Cocke–Kasami–Younger algorithm have a linear degree; therefore the context-free language recognition problem lies in $\mathcal{NC}$. The difference between our solution and their solutions is that they break the operation of context-free language recognition down to Boolean operations, while we implement context-free language recognition by using higher-level, structured operations.

We can also reduce the logical query programs [17] to a circuit whose size is a polynomial in the size of the program. Also, it is easy to prove that if the basic logic program has a polynomial fringe [17], then the degree of the circuit is a polynomial in the size of the program. This is expressed in the next corollary,

COROLLARY 7.5. *A basic logic program with the polynomial fringe property is in $\mathcal{NC}$.*

7.4. Noncommutative semirings with variable size. The *coalescing problem* arises when the composition and combination of functions is computed in $\mathcal{F}$. For all $f \in \mathcal{F}$, the size of f, defined to be the minimum possible number of items representing f, is less than m, the size of the semiring. Can we find the minimum representation of each function in $\mathcal{F}$ efficiently in parallel?

DEFINITION 7.6 (optimal coalescing problem). *Given a formula $f(x) = \Sigma_{i=1}^{n} a_i x b_i$, find a minimum representation for $f(x)$.*

THEOREM 7.7. *If $\mathcal{PNR}$ is noncombinable, then the optimal coalescing problem is $\mathcal{NP}$-hard.*

Proof. The partition problem can be reduced in polynomial time to the optimal coalescing problem. For $a_1, \ldots, a_n \in D$, let $s = \Sigma_{i=1}^{n} a_i$. Suppose there exists d such that $s = d + d$. If no such d exists, trivially, there is no partition. Let $f(x) =$

$\Sigma_{i=1}^{n} a_i x b + dxc$, for some $b \neq c$. Clearly, $a_1, \ldots, a_n$ is partitionable if and only if the optimal representation contains only one item, $dx(b+b+c)$. □

DEFINITION 7.8 (minimal coalescing problem). *A representation $f(x) = \Sigma_{i=1}^{k} a_i x b_i$ is minimal if for all i and $j \leq k (i \neq j), a_i \neq a_j, b_i \neq b_j$. The minimal coalescing problem is defined as follows: given a formula $f(x) = \Sigma_{i=1}^{n} c_i x d_i$, find a minimal representation for $f(x)$.*

Since $\mathcal{D}$ contains only m elements, there is a one-to-one onto function from $\mathcal{D}$ to $\{1, \ldots, m\}$. Let π be such a function. For all input $\Sigma_{i=1}^{n} a_i x b_i$, a_i's are sorted according to $\pi(a_i)$; then all items with the same a part are combined; next b part is sorted and combined. This procedure is one phase. The phase is repeated until all a parts and b parts are distinct from each other. Note that the number of items is reduced at least by 1 with each phase; therefore, n iterations are sufficient to find a minimal representation.

LEMMA 7.9. *The minimal coalescing problem can be solved in $O(n^2 \log n)$ time sequentially.*

CONJECTURE 7.10 (minimal coalescing problem). *The minimal coalescing problem is complete in $\mathcal{P}$.*

Because of the $\mathcal{NP}$-hardness of the optimal coalescing problem and the sequentiality of the minimal coalescing problem, the unary functions are not expected to be minimally represented. A weaker version called one-sided minimal representation is used.

DEFINITION 7.11. *A representation $\Sigma_{i=1}^{k} a_i x b_i$ is* left-hand side minimal *if for all i and $j \leq k (i \neq j), a_i \neq a_j$. The left-hand side minimal coalescing problem is defined as follows: given a formula, $f(x) = \Sigma_{i=1}^{n} c_i x d_i$, find a left-hand side minimal representation for $f(x)$.*

Clearly, if a representation, $\Sigma_{i=1}^{k} a_i x b_i$, is left-hand side minimal, then $k \leq m$.

LEMMA 7.12. *For all $f(x) = \Sigma_{i=1}^{n} c_i x d_i$, the left-hand side minimal representation can be found in $O(\log n)$ time, using n processors.*

Proof. A left-hand side minimal representation can be found after the first step (half phase) of the algorithm for finding minimal representation. Hence, the complexity is equal to sorting n elements. □

Following from Lemma 7.12, for all left-hand side minimally represented functions $f(x) = \Sigma_{i=1}^{k} a_i x b_i$ and $g(x) = \Sigma_{j=1}^{l} c_i x d_i$, a left-hand side minimal representation of $h_1(x)$ and $\mathrm{h}_2(x)$ can be computed in $O(\log(k+l))$ and $O(\log kl)$ time, using $O(k+l)$ and $O(kl)$ processors, respectively, where $h_1(x) = f(x) + g(x)$ and $h_1(x) = (f \circ g)(x)$. This leads to the following theorem.

THEOREM 7.13. *Let $\mathcal{PNR} = \{\mathcal{D}, \oplus, \odot\}$ be a noncommutative semiring of size m. Using $m^2 M(n)$ processors, all circuits over $\mathcal{PNR}$ of size n and degree d can be evaluated in $O((\log \max(m,n))(\log nd))$ time. Therefore, if m is a polynomial of n, then all circuits over $\mathcal{PNR}$ can be evaluated in polylogarithmic time in their size and degree, using only a polynomial number of processors.*

7.5. Semiring over matrices. Let $\mathcal{CR}$ be any commutative semiring and $\mathcal{MR} = \{\mathcal{D}, \oplus, \odot\}$ be a *matrix ring* over $\mathcal{CR}$, i.e.,

$$\mathcal{D} = \{A^{k \times k} \mid A_{ij} \in \mathcal{CR}, 1 \leq i, j \leq k\}.$$

Clearly, $\mathcal{MR}$ is a noncommutative semiring with infinite domain, whenever $\mathcal{CR}$ is infinite.

THEOREM 7.14. *All circuits $\mathcal{C}$ over $\mathcal{MR}$ with n nodes and degree d can be evaluated in $O((\log n)(\log nd))$ time, using $M(n)$ processors.*

Proof. Let $\mathcal{F} = \{C^{m^2\times m^2} \oslash X \mid C^{m^2\times m^2} \in \mathcal{D}_1\}$. The theorem follows from the next lemma. ∎

LEMMA 7.15. *$\mathcal{F}$ is closed over $\{\oplus, \odot\}$.*

Proof. For all $A^{k\times k} \in \mathcal{D}$, $A^{k\times k}X^{k\times k}$ and $X^{k\times k}A^{k\times k}$ are linear transformations of $(x_{1,1}, \ldots, x_{k,k})$ in space ICR^{k^2}. Let $\mathcal{D}_1 = \{C^{k^2\times k^2} \mid C_{i,j} \in \mathcal{CR}, 1 \le i, j \le k^2\}$. Therefore, there exists $B^{k^2\times k^2}$ and $C^{k^2\times k^2} \in \mathcal{D}_1$ such that

$$B^{k^2\times k^2} \oslash \begin{bmatrix} x_{1,1} & x_{1,2} & . & x_{1,k} \\ x_{2,1} & . & . & x_{2,k} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ x_{k,1} & x_{k,2} & . & x_{k,k} \end{bmatrix} = B^{k^2\times k^2} \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ . \\ . \\ . \\ x_{k,k} \end{pmatrix}$$

$$= A^{k\times k} \begin{pmatrix} x_{1,1} & x_{1,2} & . & x_{1,k} \\ x_{2,1} & . & . & x_{2,k} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ x_{k,1} & x_{k,2} & . & x_{k,k} \end{pmatrix},$$

$$C^{k^2\times k^2} \oslash \begin{bmatrix} x_{1,1} & x_{1,2} & . & x_{1,k} \\ x_{2,1} & . & . & x_{2,k} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ x_{k,1} & x_{k,2} & . & x_{k,k} \end{bmatrix} = C^{k^2\times k^2} \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ . \\ . \\ . \\ x_{k,k} \end{pmatrix}$$

$$= \begin{pmatrix} x_{1,1} & x_{1,2} & . & x_{1,k} \\ x_{2,1} & . & . & x_{2,k} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ x_{k,1} & x_{k,2} & . & x_{k,k} \end{pmatrix} A^{k\times k}.$$

Therefore, $\mathcal{F}$ is closed under projection over $\times$. Note that the composition of two functions in $\mathcal{F}$ is the matrix multiplication; hence, $\mathcal{F}$ is closed under composition. Clearly, $\mathcal{F}$ is linear and closed under combination over $+$. ∎

7.6. The power of structured computation. The power of *structured computation* in parallel computation is considered in the example of the circuit value problem over a matrix ring. Let $\mathcal{CR} = \{\mathcal{D}_1, +, \times\}$ be an infinite commutative semiring; let $\mathcal{D} = \{A^{m\times m} \mid A^{m\times m}_{ij} \in \mathcal{D}_1, 1 \le i, j \le m\}$ and $\mathcal{MR} = \{\mathcal{D}, +, \times\}$. Let $\mathcal{C}$ be a circuit of size n and degree d over an $\mathcal{MR}$. In order to evaluate $\mathcal{C}$ in parallel, the best known solution is to expand the matrix operations down to the operations defined on $\mathcal{CR}$, a commutative semiring, i.e., replacing each node in $\mathcal{C}$ by a subcircuit which simulates the operation. The corresponding circuit is denoted by $\mathcal{C}'$. Note that the degree of $\mathcal{C}'$ is also d. Therefore, the algorithm presented in [7] can be applied.

The complexity of this solution is now analyzed. If m is a constant, then the algorithm takes $O(\log n \ \log nd)$ time, using n^3 processors.[1] However, when m is a

[1] The number of processors to compute the multiplication of two matrices of dimension n in $O(\log n)$ time is charged to be n^3. The effect of the advanced matrix multiplication algorithm will be discussed in a later subsection.

polynomial of n, for instance, $m = n$. Since each multiplication node in $\mathcal{C}$ is replaced by a subcircuit of size m^3 and each addition node in $\mathcal{C}$ is replaced by a subcircuit of size m^2, in the worst case $\mathcal{C}'$ has a size of $O(n^4)$. Thus, this method takes $O(\log n \log nd)$ time, using n^{12} processors!

We will show that the same circuit can be evaluated in $O(\log n \log nd)$ time, using n^9 processors. The n^3 saving of processors is achieved by using *structured computation.*

Note that the proof of Lemma 7.15 is independent of m, the dimension of the matrix-ring. Now, we show that projection, combination, and composition can be computed in parallel efficiently.

LEMMA 7.16.

1. *The projection over* $\times$ *can be computed in* $O(\log m)$ *time, using* m^4 *processors.*
2. *The combination of the two functions in* $\mathcal{F}$ *can be computed in constant time, using* m^4 *processors.*
3. *The composition of the two functions in* $\mathcal{F}$ *can be computed in* $O(\log m)$ *time, using* m^6 *processors.*

Proof. 1. The projection operation over a $\times$ node is a linear transformation of a vector in m^2 dimension. As shown in the proof of Lemma 7.15, it is not difficult to see that this linear transformation ($B^{m^2 \times m^2}, C^{m^2 \times m^2}$ in the proof of Lemma 7.15) can be computed by using the time and processors specified in the Lemma.

2. The combination of the two functions in $\mathcal{F}$ is simply a result of performing a matrix addition.

3. The composition of the two functions is derived by performing a matrix multiplication. □

Let $\mathcal{D}_2 = \{U^{n \times n} \mid U_{i,j} \in \mathcal{F}\}$. Since $\mathcal{F}$ is closed under composition and linear over $+$, $\{\mathcal{F}, \circ, +\}$ forms a semiring. Therefore, $\{\mathcal{D}_2, \circ, +\}$ forms a semiring.

LEMMA 7.17. *For all* $U_1, U_2 \in \mathcal{D}_2$, $U_1 \circ U_2$ *can be computed in* $O(\log \max(m, n))$ *time, using* $O(n^3 m^6)$ *processors.* $U_1 + U_2$ *can be computed in constant time, using* $n^2 m^4$ *processors.*

Proof. Each entry of U_i is an $m^2 \times m^2$ matrix; and $\Sigma_{k=1}^{n} (U_1)_{i,k} (U_2)_{k,j}$, $1 \le i, j \le n$, can be computed in $O(\log \max(m, n))$ time, using $O(n^3 m^6)$ processors. The second part of this lemma follows Lemma 7.16. □

Therefore, the `compress` operation can be computed in $O(\log \max(m, n))$ time, using $n^3 m^6$ processors. The following theorem results.

THEOREM 7.18. *For all circuits* $\mathcal{C}$ *over* $\mathcal{MR}$ *with* n *nodes and degree* d, $\mathcal{C}$ *can be evaluated in* $O((\log \max(m, n))(\log nd))$ *time, using* $n^3 m^6$ *processors.*

7.7. The effect of advanced matrix multiplication algorithms. The above discussion is based on the assumption that the simplest $O(\log n)$ time, the n^3 processor, matrix multiplication algorithm is used. *What is the effect of advanced matrix multiplication algorithms?*

Let $M_{\mathcal{CR}}(n)$ be number of processors used to multiply two $n \times n$-matrices over commutative semiring $\mathcal{CR}$ in $O(\log n)$ time. Let $M_{\mathcal{MR}}(n)$ be number of processors used to multiple two $n \times n$-matrices over matrix semiring $\mathcal{MR} = \{\mathcal{D}, +, \times\}$, where $\mathcal{D} = \{A^{m^2 \times m^2} \mid A_{ij}^{m^2 \times m^2} \in \mathcal{CR}, 1 \le i, j \le m\}$, in $O(\log n)$ time, when the time and processors needed to multiply and add two $m^2 \times m^2$ matrices are charged as a constant. Clearly, the multiplication of any two $n \times n$-matrices over $\mathcal{MR}$ can be computed in $O(\log \max(m, n))$ time, using no more than $M_{\mathcal{MR}}(n) M_{\mathcal{CR}}(m^2)$ processors.

Using the above notations, the processor count for nonstructured and structured method to evaluate a matrix circuit is $P_{ns}(n, m) = M_{\mathcal{CR}}(nM_{\mathcal{CR}}(m))$ and $P_s(n, m) =$

$M_{\mathcal{CR}}(n)M_{\mathcal{MR}}(m^2)$, respectively. Since $M_{\mathcal{CR}}(n_1n_2) = M_{\mathcal{CR}}(n_1)M_{\mathcal{CR}}(n_2)$,

$$\frac{P_{ns}(n,m)}{P_s(n,m)} = M_{\mathcal{CR}}(M_{\mathcal{CR}}(m)/m^2)\frac{M_{\mathcal{CR}}(n)}{M_{\mathcal{MR}}(n)}.$$

It is reasonable to assume that $n^3 \geq M_{\mathcal{MR}}(n) \geq M_{\mathcal{CR}}(n) \geq n^2$. If $M_{\mathcal{CR}} > n^2$, then the larger the m, the more saving the structured method has. However, if $M_{\mathcal{CR}}(n) = n^2$ (possible?), then there is no reason to use the structured method.

8. Open problems.

- It is interesting and important to have a general theory for unary function class construction when an operator set is given and to have a theory for proving closure properties of certain classes of unary functions.
- The dynamic complexity of min–max-plus-times circuit over $\mathbb{R}^+$ is unknown even though we construct a class of unary functions which is closed over $(\{\min\}, \{\max, +, \times\})$ by means of reduction lemma in section 3; i.e., the uniformity of $\mathcal{F}_{(\min,\max,+,\times)}$ is still open.
- Is there an operator set whose cheap operation has more than one operator?
- What is the tight bound on the running time of the parallel Boolean circuit evaluation algorithm?
- What is the dynamic complexity of circuit over a ring with division?

Acknowledgments. We would like to thank both referees for carefully reading an earlier version of this paper and for their technical and editorial comments and suggestions that greatly improved this paper.

REFERENCES

[1] M. Ajtai, J. Komlos, and E. Szemeredi, *Sorting in c log n parallel steps*, Combinatorica, 3 (1983), pp. 1–19.

[2] H. Gazit, G. L. Miller, and S.-H. Teng, *Optimal tree contraction in an EREW model*, in Proceedings of the 1987 Princeton Workshop on Algorithm, Architecture and Technology: Issues for Models of Concurrent Computation, Princeton, NJ, 1987, pp. 139–156.

[3] L. Goldschlager, *The monotone and planar circuit value problems are log-space complete for P*, SIGACT News, (1977), pp. 25–29.

[4] S. R. Kosaraju, *On the parallel evaluation of classes of circuits*, in Proceedings of the 10th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 472, Springer-Verlag, New York, 1990, pp. 232–237.

[5] R. E. Ladner, *The circuit value problem is log-space complete for P*, SIGACT News, 7 (1975), pp. 18–20.

[6] G. L. Miller and J. H. Reif, *Parallel tree contraction and its applications*, in 26th Symposium on Foundations of Computer Science, IEEE, Portland, OR, 1985, pp. 478–489.

[7] G. L. Miller, V. Ramachandran, and E. Kaltofen, *Efficient parallel evaluation of straight-line code and arithmetic circuits*, SIAM J. Comput., 17 (1988), pp. 687–695.

[8] G. L. Miller and S.-H. Teng, *Dynamic parallel complexity of computational circuits*, in Proceedings, Symposium on the Theory of Computing, ACM, 1987, pp. 254–263.

[9] G. L. Miller and S.-H. Teng, *Systematic methods for tree based parallel algorithm development*, in Proceedings, Second International Conference on Supercomputing, Santa Clara, CA, May 1987, pp. 392–403.

[10] G. L. Miller and S.-H. Teng, *Tree-based parallel algorithm design*, in Algorithmica, 19 (1997), pp. 369–389.

[11] N. Nisan, *Lower bounds for non-commutative computation*, in Proceedings, Symposium on the Theory of Computing, ACM, 1991, pp. 410–418.

[12] V. Ramachandran and H.-H. Yang, *An efficient parallel algorithm for the general planar monotone circuit value problem*, SIAM J. Comput., 25 (1996), pp. 312–339.

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

[14] W. L. Ruzzo, *Tree-size bounded alternation*, J. Comput. System Sci., 21 (1980), pp. 218–235.
[15] W. L. Ruzzo, *On uniform circuit complexity*, J. Comput. System Sci., 22 (1981), pp. 365–383.
[16] S.-H. Teng, *The construction of Huffman-equivalent prefix code is in $\mathcal{NC}$*, SIGACT News, 18 (1987), pp. 54–61.
[17] J. D. Ullman and A. V. Gelder, *Parallel Complexity of Logical Query Programs*, Technical report, Department of Computer Science, Stanford University, Palo Alto, CA, 1985.
[18] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, *Fast parallel computation of polynomials using few processors*, SIAM J. Comput., 12 (1983), pp. 641–644.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1689–1721

THE RELATIONSHIP BETWEEN BREAKING THE DIFFIE–HELLMAN PROTOCOL AND COMPUTING DISCRETE LOGARITHMS*

UELI M. MAURER† AND STEFAN WOLF†

Abstract. Both uniform and nonuniform results concerning the security of the Diffie–Hellman key-exchange protocol are proved. First, it is shown that in a cyclic group G of order $|G| = \prod p_i^{e_i}$, where all the multiple prime factors of $|G|$ are polynomial in $\log |G|$, there exists an algorithm that reduces the computation of discrete logarithms in G to breaking the Diffie–Hellman protocol in G and has complexity $\sqrt{\max\{\nu(p_i)\}} \cdot (\log |G|)^{O(1)}$, where $\nu(p)$ stands for the minimum of the set of largest prime factors of all the numbers d in the interval $[p - 2\sqrt{p} + 1, p + 2\sqrt{p} + 1]$. Under the unproven but plausible assumption that $\nu(p)$ is polynomial in $\log p$, this reduction implies that the Diffie–Hellman problem and the discrete logarithm problem are polynomial-time equivalent in G. Second, it is proved that the Diffie–Hellman problem and the discrete logarithm problem are equivalent in a uniform sense for groups whose orders belong to certain classes: there exists a polynomial-time reduction algorithm that works for all those groups. Moreover, it is shown that breaking the Diffie–Hellman protocol for a small but nonnegligible fraction of the instances is equally difficult as breaking it for all instances. Finally, efficient constructions of groups are described for which the algorithm reducing the discrete logarithm problem to the Diffie–Hellman problem is efficiently constructible.

Key words. public-key cryptography, Diffie–Hellman protocol, discrete logarithms, finite fields, elliptic curves

AMS subject classifications. 11T71, 68Q15

PII. S0097539796302749

1. Introduction.

Two challenging open problems in cryptography are to prove or disprove that breaking the Diffie–Hellman (DH) protocol [13] is computationally equivalent to computing discrete logarithms in the underlying group and that breaking the Rivest–Shamir–Adleman system [40] is computationally equivalent to factoring the modulus. This paper is concerned with the first of these problems.

1.1. The discrete logarithm problem.

Let G be a finite cyclic group (written multiplicatively) generated by g. The *discrete logarithm* (DL) *problem* for the group G can be stated as follows: Given g and $a \in G$, find the unique integer s in the interval $[0, |G| - 1]$ such that $g^s = a$. The number s is called the discrete logarithm of a to the base g.

1.2. The DH key-exchange protocol and the DH problem.

The DH protocol [13] allows two parties, Alice and Bob, connected by an authenticated but otherwise insecure channel (for instance an insecure telephone line over which Alice and Bob authenticate each other by speaker recognition), to generate a mutual secret

*Received by the editors April 29, 1996; accepted for publication (in revised form) January 20, 1998; published electronically May 7, 1999. Preliminary versions of this paper appeared as [U. M. Maurer, *Towards the equivalence of breaking the Diffie–Hellman protocol and computing discrete logarithms*, in Advances in Cryptology—CRYPTO'94, Lecture Notes in Comput. Sci. 839, Springer-Verlag, New York, 1994, pp. 271–281; U. M. Maurer and S. Wolf, *Diffie–Hellman oracles*, in Advances in Cryptology—CRYPTO'96, Lecture Notes in Comput. Sci. 1109, Springer-Verlag, New York, 1996, pp. 268–282]. This work was supported by Swiss National Science Foundation (SNF) grant 20–42105.94.

http://www.siam.org/journals/sicomp/28-5/30274.html

†Department of Computer Science, Swiss Federal Institute of Technology, ETH Zürich, CH-8092 Zürich, Switzerland (maurer@inf.ethz.ch, wolf@inf.ethz.ch).

key which appears to be computationally infeasible to determine for an eavesdropper overhearing the entire conversation between Alice and Bob.

The protocol works as follows. Let $G = \langle g \rangle$ be a cyclic group generated by g for which the DL problem is believed to be hard. In order to generate a mutual secret key, Alice and Bob secretly choose integers s_A and s_B, respectively, at random from the interval $[0, |G| - 1]$. Then they compute secretly $a_A = g^{s_A}$ and $a_B = g^{s_B}$, respectively. Note that there exist efficient algorithms for exponentiation in groups. Finally, they exchange these group elements over the insecure public channel and compute $a_{AB} = a_B^{s_A} = g^{s_A s_B}$ and $a_{BA} = a_A^{s_B} = g^{s_B s_A}$, respectively. Since $a_{AB} = a_{BA}$, this quantity can be used as a secret key shared by Alice and Bob. More precisely, they need to apply a function mapping elements of G to the key space of a cryptosystem.

It is unknown whether a group exists for which the DL problem is hard, but several candidate groups have been proposed. Examples are the multiplicative groups of large finite fields (prime fields [13] or extension fields), the multiplicative group of residues modulo a composite number [31], [32], elliptic curves over finite fields [36], [21], the Jacobian of a hyperelliptic curve over a finite field [20], and the class group of imaginary quadratic fields [7].

The security of the DH protocol is based on the assumptions that the DL problem is hard to solve in G and that this implies that it is hard to compute $g^{s_A s_B}$ from g^{s_A} and g^{s_B}. We will refer to the problem of computing $g^{s_A s_B}$ from g^{s_A} and g^{s_B} as the *DH problem*. This paper is mainly concerned with the relationship between the DH and DL problems. It is clear that the DH problem cannot be more difficult than the DL problem because exponentiation in a group is efficient. Conversely, even when we are using a group for which the DL problem is hard, this does not immediately imply that the DH protocol is secure. However, we will show that, for every group whose order is not divisible by the square of a large prime number, the DH problem cannot be substantially easier than the DL problem. Moreover, for certain classes of groups an efficient algorithm reducing the DL problem to the DH problem not only exists but is efficiently constructible.

1.3. Outline of the paper. The paper is organized as follows. In section 2 a general index-search problem is defined and investigated, and some algorithms for computing discrete logarithms are described. In sections 3 and 4 a technique for proving the equivalence of the DH and DL problems, using so-called auxiliary groups, is presented, and examples of suitable auxiliary groups—for instance, elliptic curves or subgroups of the multiplicative group of a finite field—are described. These two sections contain the main results of this paper. More precisely, a generalization of the result of [26] is proved in section 3, which states that the DH and DL problems are equivalent for groups G for which appropriate auxiliary groups are given. The first result of section 4 is a nonuniform reduction of the DL problem to the DH problem: it is shown (under an unproven but plausible number-theoretic conjecture) that there exists, for every group whose order does not contain a multiple large prime factor, a polynomial-time algorithm computing discrete logarithms and making calls to an oracle solving the DH problem. The second result of section 4 is a list of smoothness conditions (depending on $|G|$) which make the DH and DL problems equivalent in a uniform sense; i.e., an efficient reduction algorithm not only exists but also can be found efficiently. In section 5, several variants of the DH problem are defined, and it is shown that they are (almost) as hard as the original DH problem. For instance, breaking the DH problem with small probability is equally hard as breaking it with arbitrarily high probability.

In Appendix A we describe an algorithm for finding generating sets of Abelian groups. Appendix B contains some basic facts about Gröbner basis computations which are required in section 4. In Appendix C we obtain results which are stronger than those of sections 3 and 4 under the assumption that efficient *algorithms* exist for solving the DH problem in certain groups. In Appendix D, we show how to construct DH groups for which the DH and DL problems are provably equivalent.

1.4. Related work. Considerations on related topics can be found in [3], [4], [10], [11], [12], [28], [29], [42], and [45]. In [4], the notion of a black-box field is introduced, which makes more explicit the concept of computation with implicitly represented elements presented in [26]. Furthermore, the existence of a uniform reduction of the DL problem to the DH problem of subexponential complexity was proved in [4], using methods related to those of [26] and of section 3 and Appendix C of this paper.

In [45], the hardness of the DH problem (and hence of the DL problem) is proved in the generic model, i.e., for general-purpose algorithms that do not exploit any special property of the representation of the group elements. However, it was shown in [28] that the DH and DL problems are *not* computationally equivalent in a generic sense if the group order contains multiple large prime factors. In [11], the hardness of the DL and DH problems modulo p is proved in special computational models. For example it was shown that the DH function cannot be interpolated by a low-degree polynomial.

An alternative construction to that of section 5 for correcting a faulty oracle solving the DH problem is described in [45]. Finally, a comparison of the security of different DL based systems is given in [42].

2. The index-search problem and algorithms for computing discrete logarithms.

2.1. The index-search and DL problems. Let $A = (a_i)_{i=0,\dots,n-1}$ be a list of elements of some set S such that for a given i it is easy to compute a_i. We call the problem of computing for a given $b \in S$ an index i such that $b = a_i$ the *index-search problem*. It can trivially be solved by exhaustive search which requires at most n comparisons. If the list has the property that the permutation $\sigma : a_i \mapsto a_{i+1}$ (where the index is reduced modulo n) can efficiently be computed, then the search can be sped up by a time-memory trade-off known as *the baby-step giant-step* algorithm. Using a table of size M to store the sorted list of values $b, \sigma(b), \dots, \sigma^{M-1}(b)$, one can compute the elements $a_0, a_M, a_{2M}, \dots$ until one of them, say, a_{iM}, equals an element $\sigma^j(b)$ contained in the table. Then the index of b is $iM - j$. For the choice $M := \lceil \sqrt{n} \rceil$, the running time of the algorithm is $O(\sqrt{n} \log n)$.

The DL problem in a cyclic group H of order $|H|$ with generator h is the index-search problem for the list $(1, h, \dots, h^{|H|-1})$. Multiplication with h corresponds to the above-mentioned permutation σ. Hence the baby-step giant-step algorithm is applicable for solving the DL problem. It is a general-purpose algorithm that uses no particular properties of the representation of the group elements other than the uniqueness of the representation.

2.2. The Pohlig–Hellman algorithm. We describe a generic algorithm due to Pohlig and Hellman [37] which reduces the computation of discrete logarithms to the same problem in the minimal nontrivial subgroups. It plays a central role in this paper.

THEOREM 1 (see [37]). *Let $H = \langle h \rangle$ be a cyclic group with order $|H| = \prod_{i=1}^{r} q_i^{f_i}$, and let $a = h^x \in H$ be given. The discrete logarithm x of a can be computed by $O(\sum f_i(\log |H| + q_i))$ group operations and equality tests of group elements. If memory space for storing $\lceil \sqrt{q} \rceil$ group elements (where q is the greatest prime factor of $|H|$) is available, the running time can be reduced to $O(\sum f_i(\log |H| + \sqrt{q_i} \log q_i))$.*

Proof. To solve $a = h^x$ for x, we first compute x modulo $q_i^{f_i}$ for all i. This is done by determining, modulo q_i, the coefficients x_{ij} of the q_i-adic representation of x modulo $q_i^{f_i}$,

$$x \equiv \sum_{j=0}^{f_i-1} x_{ij} q_i^j \pmod{q_i^{f_i}}.$$

The number x_{i0} is the discrete logarithm of $a^{|H|/q_i} = h^{x \cdot |H|/q_i} = h^{x_{i0} \cdot |H|/q_i}$ in the group $H^{(i)} := \langle h^{|H|/q_i} \rangle$ of order q_i. Assume now that $x_{i0}, \ldots, x_{i,k-1}$ have already been computed. The number x_{ik} is the discrete logarithm of

$$\left(a \cdot h^{-(x_{i0} + \cdots + x_{i,k-1} q_i^{k-1})} \right)^{|H|/q_i^{k+1}} = h^{x_{ik} \cdot |H|/q_i}$$

in the same group $H^{(i)}$. The computation of a discrete logarithm in $H^{(i)}$ has complexity $O(q_i)$ with exhaustive search and can be sped up by a factor M when a table of size M is used (that can be sorted in time $O(M \log M)$).

Given x modulo $q_i^{f_i}$ for all i, Chinese remaindering yields the discrete logarithm x of a modulo $|H|$. The complexity of the entire algorithm is

$$O\left(\sum f_i(\log |H| + q_i) \right)$$

or

$$O\left(\sum f_i(\log |H| + \sqrt{q_i} \log q_i) \right)$$

when the baby-step giant-step algorithm with $M = \lceil \sqrt{q_i} \rceil$ is used. □

The algorithm is efficient only if $|H|$ is smooth, i.e., if $q_i \le B$ for a small *smoothness bound* B. If this condition is satisfied, we have in the worst case that $q_i \approx B$ for all i; i.e., the number of factors is $\log |H| / \log B$, and the complexity is

$$O\left((\log |H|)^2 + \frac{B}{\log B} \log |H| \right)$$

or

$$O\left((\log |H|)^2 + \sqrt{B} \log |H| \right)$$

when the baby-step giant-step trade-off is used.

It is crucial in the following that the algorithm be generic, i.e., that it use operations in H and equality tests of group elements only. Shoup showed in [45] that no general-purpose algorithm can solve the DL problem faster than the Pohlig–Hellman algorithm together with the baby-step giant-step trade-off. For special groups such as the multiplicative group of a finite field, more efficient algorithms are known. We refer the reader to [33] for a detailed discussion of the DL problem and algorithms for solving it.

3. A general technique for reducing the DL problem to the DH problem. In this section we describe a technique that allows us to reduce the DL problem to the DH problem efficiently in groups G (more precisely, in all groups of certain orders) which satisfy certain conditions.

In section 3.1 we define the notion of a DH oracle, and the subsequent sections deal with the problem of computing discrete logarithms in a group G when given such an oracle for G. As a preparation for this, it is investigated in section 3.2 what kind of computations are possible in the exponents (i.e., in the unknown discrete logarithms) of group elements when given a DH oracle. In section 3.3, the concept of auxiliary groups is defined, and in sections 3.4 and 3.5 it is shown that these auxiliary groups are a tool for reducing the DL problem to the DH problem.

3.1. Computing discrete logarithms with an oracle solving the DH problem. In order to prove results concerning the equivalence of breaking the DH protocol and computing discrete logarithms we assume the availability of an oracle that solves the DH problem.

DEFINITION 1. *A DH oracle for a group G with respect to a given generator g takes as inputs two elements $a, b \in G$ (where $a = g^u$ and $b = g^v$) and returns the element g^{uv}.*

In the following we describe a polynomial-time reduction of the DL problem to the DH problem for certain classes of groups. Let G be a cyclic group generated by g for which the prime factorization of the order $|G|$ is known, and let $a = g^s$ be a given group element for which we want to compute the discrete logarithm s using a DH oracle for G. It is sufficient to compute s modulo each prime factor of $|G|$ (or modulo the prime powers if $|G|$ contains multiple prime factors) and to combine these values by Chinese remaindering. Only large prime factors are relevant because the Pohlig–Hellman algorithm allows us to compute s modulo powers of small prime factors of $|G|$. Hence we can restrict our attention to the problem of computing s modulo p for a large prime factor p of $|G|$. We assume that p is a *single* prime factor of $|G|$; the case of $|G|$ having multiple large prime factors is discussed in section 3.5. Let x be the element of $GF(p)$ defined by $s \equiv x \pmod{p}$. In the following sections, the problem of computing x from the group element g^s is investigated.

3.2. Computing with implicit representations using a DH oracle. Every element y of the field $GF(p)$ can be interpreted as corresponding to a set of elements of G, namely, those whose discrete logarithm is congruent to y modulo p. Every element of this set is then a representation of the field element y.

DEFINITION 2. *Let G be a cyclic group with a fixed generator g, and let p be a prime divisor of the group order. Then, a group element $a = g^{y'}$ is called an* implicit representation *(with respect to G and g) of the element $y \in GF(p)$ if $y \equiv y' \pmod{p}$. We write $y \rightsquigarrow a$.*

Note that the implicit representation of a field element is not unique if $|G| \neq p$.

The following operations on elements of $GF(p)$ can be performed efficiently on implicit representations of these elements (i.e., by operating in the group G), where the result is also in implicit form. Let y and z be elements of $GF(p)$, with

$$y \rightsquigarrow a, \quad z \rightsquigarrow b.$$

Because

$$y = z \text{ if and only if } a^{|G|/p} = b^{|G|/p},$$

equality of two implicitly represented elements of $GF(p)$ can be tested by $O(\log |G|)$ group operations. Furthermore we have

$$\begin{aligned} y + z &\rightsquigarrow a \cdot b, \\ yz &\rightsquigarrow \mathrm{DH}_g(a, b), \\ -y &\rightsquigarrow a^{-1} = a^{|G|-1} \end{aligned}$$

(where DH_g stands for the DH function with respect to the generator g), and these implicitly executed operations on $GF(p)$ elements require a group operation in G, a call to the DH oracle, and $O(\log |G|)$ group operations, respectively.

In order to simplify the notation, we also introduce the notion of an eth-*power* DH oracle ($\mathrm{PDH}_{g,e}$ oracle) that computes an implicit representation of the eth power of an implicitly represented element. A possible implementation of a $\mathrm{PDH}_{g,e}$ oracle is to use a "square and multiply" algorithm for obtaining an implicit representation of y^e, denoted by $\mathrm{PDH}_{g,e}(a)$, by $O(\log e)$ calls to a normal DH oracle (remember that $y \rightsquigarrow a$). In particular we can compute inverses of implicitly represented elements because

$$y^{-1} \rightsquigarrow \mathrm{PDH}_{g,p-2}(a) \ .$$

We call addition, subtraction, multiplication, division, and equality testing in $GF(p)$ *algebraic* operations. Any efficient computation in $GF(p)$ can be performed equally efficiently on implicit representations whenever it makes use only of algebraic operations. Examples are the evaluation of a rational function, testing quadratic residuosity of y by comparing

$$(\mathrm{PDH}_{g,(p-1)/2}(a))^{|G|/p} \text{ and } \ g^{|G|/p} \ ,$$

or the computation of square roots using an algorithm of Massey [25]. We will crucially rely on the fact that algorithms based on exhaustive search (for example generic algorithms for solving the index-search problem, in particular the DL problem) can be executed on implicitly represented arguments and lead to explicit results.

3.3. Auxiliary groups. When given a DH oracle for G, the computation of x is shown to work efficiently if an auxiliary group H over $GF(p)$ with certain properties exists. (Remember that $s \equiv x \pmod p$, where p is a large prime factor of $|G|$, and that s is the discrete logarithm we want to compute.) The basic idea is to embed the unknown x into an implicitly represented element c of H and to compute the discrete logarithm of this element explicitly. We now define a first required property of the auxiliary group H.

DEFINITION 3. *A finite group H is said to be* defined (m, α)-algebraically over $GF(p)$ *if the elements of H can be represented as m'-tuples (for some $m' \leq m$) of elements of $GF(p)$ such that the group operation in this representation can be carried out by at most α algebraic operations in $GF(p)$.*

We will need the following stronger property for auxiliary groups.

DEFINITION 4. *A group H is* defined strongly (m, α)-algebraically over $GF(p)$ *if H is defined (m, α)-algebraically over $GF(p)$ and if there exist two algorithms,* EMBED *and* EXTRACT*, with the following properties.*

1. *For all $(x, e) \in GF(p)^2$ the* EMBED *algorithm with input (x, e) either outputs a group element c of H or reports failure.*

2. *If the* EMBED *algorithm is run with the input* (x, e) *for fixed* x *and randomly chosen* e *until the algorithm does not fail, then the expected running time until an element* $c \in H$ *is computed is at most* α *algebraic operations in* $GF(p)$.
3. *If the* EMBED *algorithm does not fail for the input* (x, e), *then*

$$\text{EXTRACT}(\text{EMBED}(x, e), e) = x.$$

4. *The* EXTRACT *algorithm runs in time at most* α.

In the examples of section 4, the EMBED algorithm computes a group element c that contains $x + e$ as a coordinate, and the EXTRACT procedure outputs this particular coordinate minus e.

In the next section we show how an Abelian group H with bounded rank, defined strongly algebraically over $GF(p)$, and with smooth order can be used as an auxiliary group in the reduction of the computation of discrete logarithms modulo p in G to break the DH protocol for G.

3.4. The reduction algorithm. First we extend the definition of implicit representations from elements of $GF(p)$ to m-tuples over $GF(p)$.

DEFINITION 5. *Let* p *and* G *be as above and let* $a_i \in G$ *and* $y_i \in GF(p)$ *(for* $i = 1, \ldots, m$*). We say that* $(a_1, \ldots, a_m)$ *is an implicit representation of* $(y_1, \ldots, y_m)$ *if* $y_i \rightsquigarrow a_i$ *for* $1 \le i \le m$.

THEOREM 2. *Let* P *be a fixed polynomial. Let* G *be a cyclic group with generator* g *such that* $|G|$ *and its factorization* $|G| = \prod_{i=1}^{s} p_i^{e_i}$ *are known. If there exist* m, α, *and* B, *all upper bounded by* $P(\log |G|)$, *such that every prime factor* p *of* $|G|$ *greater than* B *is single, and for every such* p, *a finite Abelian group* H_p *with rank* $r = O(1)$, *defined strongly* (m, α)*-algebraically over* $GF(p)$, *exists whose order* $|H_p|$ *is* B*-smooth, then breaking the DH protocol for* G *with respect to* g *is probabilistic polynomial-time equivalent to computing discrete logarithms in* G *to the base* g.

The expected complexity of the computation of a discrete logarithm in G *when given a DH oracle for* G *is* $O(m^2 B^r (\log |G|)^2 / \log B + m^2 \alpha (\log |G|)^3)$ *group operations in* G, $O(m^2 \alpha (\log |G|)^3)$ *calls to the DH oracle for* G, *and* $O(m^2 \alpha (\log |G|)^3 + m \alpha B^r (\log |G|)^2 / \log B)$ *field operations in* $GF(p)$ *for* $p \le |G|$. *The complexities can be reduced by a time-memory trade-off.*

Proof.[1] Let p be a single prime factor of $|G|$ larger than B. Assume that an auxiliary group H is given that is defined strongly (m, α)-algebraically over $GF(p)$ with B-smooth order $|H| = \prod q_i^{f_i}$. It is clear that H has the property that when given an implicitly represented field element $x \in GF(p)$, an implicitly represented *group* element c of H (and an explicit element e of $GF(p)$) can be found efficiently with the property that from the *explicit* representation of c (and from e), the EXTRACT algorithm leads to the element x. The reason is that because the EMBED procedure uses only algebraic operations, it works also on implicitly represented inputs (where the group element of the output is also implicitly represented). This fact allows us to reduce the computation of discrete logarithms in G (modulo p) to the same problem in the group H. The field element x is computed from an implicit representation of x in four steps.

Step 1. Use the EMBED algorithm to obtain, when given an implicit representation of x and a random $e \in GF(p)$, an implicit representation of a group element c of H.

[1] The reader may wish to consult the survey paper [27], where a special case of this theorem is proved. More precisely, the proof is given under the assumption that all the auxiliary groups are cyclic elliptic curves over $GF(p)$.

Step 2. Compute the discrete logarithm of c in H (with respect to some generator set).

Step 3. Compute c explicitly.

Step 4. Use the EXTRACT algorithm to obtain x explicitly:

$$x = \text{EXTRACT}(c, e) \ .$$

We have to prove the stated complexity bounds for Step 2. The group H is Abelian of rank r; i.e., H is isomorphic to $\mathbf{Z}_{n_1} \times \cdots \times \mathbf{Z}_{n_r}$ for some $n_1, \ldots, n_r$ satisfying $\prod_{j=1}^{r} n_j = |H|$ and such that n_{j+1} divides n_j for $j = 1, \ldots, r-1$. Let $h_1, \ldots, h_r$ be a set of generators of H such that $|\langle h_j \rangle| = n_j$ and H is the internal product of the cyclic subgroups $\langle h_1 \rangle, \ldots, \langle h_r \rangle$:

$$H = \langle h_1 \rangle \times \cdots \times \langle h_r \rangle \ .$$

(If no generator set for H is known, it can be computed by the method described in Appendix A.)

The element $c \in H$ has a unique representation:

$$c = \sum_{j=1}^{r} k_j h_j, \qquad 0 \le k_j < n_j.$$

(The group H is written additively.) We address the problem of computing the coefficients k_j. This can be done by a generalization of the Pohlig–Hellman algorithm (see section 2), applied to implicitly represented group elements. The following is repeated for every prime q dividing $|H|$. We describe the first and second iteration step of an algorithm that computes k_j modulo the highest power of q dividing n_j for all $j = 1, \ldots, r$. The algorithm uses v_j $(j = 1, \ldots, r)$ as local variables (initialized by $v_j \leftarrow 0$).

For the first step, let α_1 be the number of generators h_j whose order contains the same number of factors q as n_1. In other words, $(n_1/q)h_j$ is different from the unity e of H exactly for $j = 1, \ldots, \alpha_1$. Because H is defined algebraically over $GF(p)$, an implicit representation of

$$\frac{n_1}{q} c$$

can be efficiently computed from an implicit representation of c. For all $(t_1, \ldots, t_{\alpha_1}) \in \{0, \ldots, q-1\}^{\alpha_1}$, we compute (explicitly)

$$\frac{n_1}{q} t_1 h_1 + \cdots + \frac{n_1}{q} t_{\alpha_1} h_{\alpha_1},$$

transform the coordinates to an implicit representation, and test equality with $(n_1/q)c$. Equality indicates that the t_j are congruent to the coefficients k_j modulo q. We set $v_j \longleftarrow t_j$ for these t_j, and for $1 \le j \le \alpha_1$.

For the second step, let α_2 be the number of elements h_j whose order contains at most one factor q less than n_1, i.e., $(n_1/q^2)h_j \neq e$ exactly for $j = 1, \ldots, \alpha_2$. Implicit representations of the group elements

$$\frac{n_1}{q^2}(t_1 q + v_1)h_1 + \cdots + \frac{n_1}{q^2}(t_{\alpha_1} q + v_{\alpha_1})h_{\alpha_1} + \frac{n_1}{q^2} t_{\alpha_1+1} h_{\alpha_1+1} + \cdots + \frac{n_1}{q^2} t_{\alpha_2} h_{\alpha_2}$$

are computed for all $(t_1, \ldots, t_{\alpha_2}) \in \{0, \ldots, q-1\}^{\alpha_2}$ until equality with the implicitly represented element

$$\frac{n_1}{q^2} c$$

holds. Then assign

$$v_j \longleftarrow t_j q + v_j \quad (j = 1, \ldots, \alpha_1) \ ,$$

$$v_j \longleftarrow t_j \quad (j = \alpha_1 + 1, \ldots, \alpha_2) \ .$$

After repetition of this process up to the maximal q power, q^g dividing n_1, the resulting v_j satisfy

$$\frac{n_1}{q^g} c = \sum_{j=1}^{r} \frac{n_1}{q^g} v_j h_j;$$

i.e., k_j is congruent to v_j modulo the highest power of q dividing $n_j = \operatorname{ord} h_j$ for $j = 1, \ldots, r$.

After running the algorithm for all primes q dividing $|H|$, one can compute the coefficients k_j modulo $\operatorname{ord} h_j$ by Chinese remaindering. The complexity of the algorithm is

$O((\log |H|)^2)$ operations in H with implicitly represented elements,
$O(m \frac{B^r}{r \log B} \log |H| \log |G| + \alpha \log |G|)$ operations in G,
$O(\alpha \log |G|)$ calls to the DH oracle for G, and
$O(r(\log |H|)^2 + \log |H| \frac{B^r}{\log B}$ explicit operations in H.

The first part of the number of group operations comes from the comparisons of implicitly represented elements of H. Note that $|H| \leq p^m$ because H is defined (m, α)-algebraically over $GF(p)$. The implicit and explicit operations in H can be further reduced to operations and DH oracle calls in G and operations in $GF(p)$. Then, one obtains the following complexities.

$O(m^2 \frac{B^r}{r \log B} \log p \log |G| + m^2 (\log p)^2 \alpha \log |G|)$ group operations in G,
$O(m^2 (\log p)^2 \alpha \log |G|)$ calls to the DH oracle for G, and
$O((m^2 (\log p)^2 + m \log p \frac{B^r}{r \log B}) \cdot \alpha \log p)$ field operations in $GF(p)$.

The complexities can be reduced by a time-memory trade-off if memory space is available. The running time is polynomial in $\log |G|$ because m, α, and B are polynomial in $\log |G|$, and because $r = O(1)$. □

3.5. The case of multiple large prime factors in $|G|$. In the previous sections we assumed that all the large prime factors of $|G|$ are single. Under certain additional conditions one can also treat the case of multiple large prime factors of $|G|$. If p^e divides $|G|$ (with $e > 1$), the discrete logarithm s must be computed explicitly modulo p^e instead of modulo p. This can be done if either an additional DH oracle for a certain subgroup of G is given (Case 1) or pth roots can efficiently be computed in G (Case 2).

Case 1. Assume that a DH oracle for the group $\langle g^{|G|/p} \rangle$ is given. We write

$$x \equiv \sum_{i=0}^{e-1} x_i p^i \pmod{p^e}$$

with $x_i \in GF(p)$ for $i = 0, \ldots, e-1$. Let $k \leq e-1$, assume that $x_0, \ldots, x_{k-1}$ are already computed (note that x_0 can be computed as described in the previous section), and consider the problem of computing x_k. Let $a' := a \cdot g^{-x_0 - \cdots - x_{k-1}p^{k-1}}$. Then

$$\begin{aligned} a' &= g^{x_0 + x_1 p + \cdots + x_{e-1}p^{e-1}} \cdot g^{-x_0 - \cdots - x_{k-1}p^{k-1}} \\ &= g^{x_k p^k + x_{k+1}p^{k+1} + \cdots + x_{e-1}p^{e-1}} = \left(g^{p^k}\right)^{x_k + p \cdot l} \end{aligned}$$

for some l. From a', compute

$$a'' := (a')^{|G|/p^{k+1}} = \left(g^{|G|/p}\right)^{x_k},$$

and from a'', x_k can be computed as described in the previous section by using the DH oracle for $\langle g^{|G|/p} \rangle$. More generally, this also works when a DH oracle for any group $\langle g^{d \cdot p^{e-1}} \rangle$, where $d \cdot p^{e-1}$ divides $|G|/p$, is given.

Case 2. Assume that a' (see Case 1) is computed. If an element a''' of the form

$$a''' = g^{x_k + p \cdot l'}$$

for some l' is computed, x_k can again be obtained as in section 3.4, with the DH oracle for $\langle g \rangle$. Such an element a''' can be obtained by computing a p^kth root, i.e., k times the pth root, of a'. Any p^kth root of a' is of the required form because p divides $|G|/p^k$.

However, it has been shown that in the model of *generic* algorithms, it is *not* possible to compute discrete logarithms in a group G more efficiently than in time $\Omega(\sqrt{p})$ with a DH oracle for G, if p is a *multiple* prime factor of $|G|$ [28], [49]. The model of generic algorithms was introduced by Shoup [45]. Intuitively, a generic algorithm is a general-purpose algorithm that works for all groups of a certain order and does not make use of any particular property of the representation of the group elements. Of course this result implies that in the generic model a DH oracle cannot be efficiently used to construct the required subgroup oracles of Case 1 (a result which had already been proved in [45]) and that for large p, pth roots cannot be computed efficiently by a generic algorithm in a group of which the order is divisible by p^2, even when a DH oracle is given for this group (Case 2) [28], [49].

4. Applicable auxiliary groups over $GF(p)$. In this section, two classes of possible auxiliary groups satisfying the requirements specified in the previous section are described: elliptic curves over finite fields and subgroups of the multiplicative groups of finite fields. The applicability of Jacobians of hyperelliptic curves (see [20] and [9]) as auxiliary groups was demonstrated in [48].

Two types of results are derived as a consequence of the applicability of these classes of groups as auxiliary groups. First, a nonuniform reduction of the DL to the DH problem is shown. Under an unproven assumption on the existence of smooth numbers in small intervals, the complexity of this reduction is polynomial in $\log |G|$; i.e., for every group (if no squares of large primes divide the order) there exists an algorithm for computing discrete logarithms in polynomial time if it is allowed to make calls to a DH oracle for this group. As mentioned already, such a reduction does not exist (in the model of generic algorithms) if the group order contains multiple large prime factors.

Moreover, we give a list of expressions $A(p)$ in p with the property that an auxiliary group H_p with order $A(p)$ over $GF(p)$ can efficiently be constructed. Theorem 2 then implies that if for each prime factor p of $|G|$ one of the expressions in this list is smooth, then breaking the DH protocol and computing discrete logarithms are equivalent for G (if $|G|$ has no multiple large prime factors). The equivalence of the DH and DL problems holds in a uniform sense for these groups because an efficient reduction algorithm, whose existence is guaranteed by the nonuniform result, can even be found efficiently.

4.1. Elliptic curves.

4.1.1. Applicability as auxiliary groups. Let $\mathbf{F}$ be a field (whose characteristic is not 2 or 3) and let $A, B \in \mathbf{F}$ with $4A^3 + 27B^2 \neq 0$ (in $\mathbf{F}$). The elliptic curve $E_{A,B}(\mathbf{F})$ (with parameters A and B in $\mathbf{F}$) is the set

$$\{(x, y) \in \mathbf{F}^2 : y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}.$$

(The additional point $\mathcal{O}$ is called "point at infinity.") Together with a certain operation on the set of points, $E_{A,B}(\mathbf{F})$ forms an Abelian group of rank at most 2. We refer to [35] for an introduction to elliptic curves.

Here we show that an elliptic curve E over the field $GF(p)$ is defined strongly $(2, O((\log p)^2))$-algebraically over $GF(p)$. Therefore it can be used as an auxiliary group if it has smooth order. Note that the order of an elliptic curve can be computed in polynomial time [44],[6]. The points of E can be represented as pairs of $GF(p)$-elements, and the group operation can be executed in this representation by a constant number of additions, multiplications, divisions, and equality tests in $GF(p)$. We describe the EMBED algorithm. Let $x, e \in GF(p)$ be given. First the expression $D = (x + e)^3 + A(x + e) + B$ is computed and its quadratic residuosity is tested. If D is not a quadratic residue, the algorithm reports failure (and a new value for e is chosen). If D is a quadratic residue, then a square root y of D is computed by an algorithm due to Massey [25] (see also Lemma 3 in section 5.4). Then the EMBED algorithm outputs $c = (x + e, y)$. The necessary executions of the EMBED algorithm require $O((\log p)^2)$ algebraic operations in $GF(p)$.

One can show in a completely analogous manner that an elliptic curve over an extension field $GF(p^n)$ of $GF(p)$, where n is polynomial in $\log p$, can also be used as an auxiliary group.

4.1.2. Existence. It is well-known that for any $A, B \in GF(p)$

$$p - 2\sqrt{p} + 1 \leq |E_{A,B}(GF(p))| \leq p + 2\sqrt{p} + 1 \ ,$$

and that for every $d \in [p - 2\sqrt{p} + 1, p + 2\sqrt{p} + 1]$, there exists a cyclic elliptic curve over $GF(p)$ with order d [41]. This implies the following non-uniform reduction of the DL problem to the DH problem.

DEFINITION 6. *For a number n, let $\nu(n)$ be the minimum, taken over all d in the interval $[n - 2\sqrt{n} + 1, n + 2\sqrt{n} + 1]$, of the largest prime factor of d.*

THEOREM 3. *Let P be a fixed polynomial. For every finite cyclic group G with order $|G| = \prod p_i^{e_i}$ and such that all multiple prime factors p_i of $|G|$ are smaller than $B := P(\log |G|)$, there exists an algorithm that makes calls to a DH oracle for G and computes discrete logarithms of elements of G in time*

$$\max\{\nu(p_i)\} \cdot (\log |G|)^{O(1)} \ .$$

The quantity $\nu(p)$ is directly linked with the existence of a smooth number in the interval $[p-2\sqrt{p}+1, p+2\sqrt{p}+1]$. Unfortunately, very little is known about smooth numbers in such intervals. However, it is known [8] that for every fixed u,

$$\psi(n, n^{1/u}) = n/u^{(1+o(u))u}, \tag{1}$$

where $\psi(n, y)$ denotes the number of y-smooth integers $\leq n$. This fact suggests that $\nu(n)$ is polynomial in $\log n$.

Smoothness Assumption. $\nu(n) = (\log n)^{O(1)}$.

This assumption implies that the algorithms of Theorem 3 run in time polynomial in $\log |G|$, and this yields a polynomial-time nonuniform reduction of the DL problem to the DH problem for all groups whose orders are free of multiple large prime factors. Moreover, the reduction algorithms are generic, i.e., they depend only on the group order $|G|$ of G, and they have a description of length linear in $\log |G|$, namely the large prime factors of $|G|$ and parameters of suitable elliptic curves.

COROLLARY 4. *Let P be a fixed polynomial. If the smoothness assumption is true, then for every group $G = \langle g \rangle$ whose order is free of multiple prime factors greater than $B := P(\log |G|)$, there exists a side-information string S of length at most $3 \log |G|$ such that when given S, breaking the DH protocol for G is polynomial-time equivalent to computing discrete logarithms in G.*

Remark. The group order of Jacobians of hyperelliptic curves of genus 2 varies in a larger interval of size $[n-\Theta(n^{3/4}), n+\Theta(n^{3/4})]$, but the results about the distribution of the orders which are proved in [1] are not sufficient to prove the existence of the side-information string without unproven assumption. The reason is that in [1] the existence of Jacobians with *prime* order is proved, whereas Jacobians with *smooth* order are required for our purpose.

In the model of generic algorithms the results described in section 3.5 (see [28], [45]) and in this section imply the following complete characterization of group orders n for which there exists an efficient generic algorithm computing discrete logarithms, making calls to a DH oracle for the same group.

COROLLARY 5. *If the smoothness assumption is true, then there exists a polynomial-time generic algorithm computing discrete logarithms in cyclic groups of order n, making calls to a DH oracle for the same group, if and only if all the multiple prime factors of n are of order $(\log n)^{O(1)}$.*

4.1.3. Construction of elliptic curves. For certain expressions $A(p)$, elliptic curves over $GF(p)$ with order $A(p)$ can explicitly be constructed. The curve over $GF(p)$ defined by the equation

$$y^2 = x^3 - Dx \tag{2}$$

has order $p+1$ if $p \equiv 3 \pmod 4$, and the curve

$$y^2 = x^3 + D \tag{3}$$

has also order $p+1$ if $p \equiv 2 \pmod 3$. Thus if $p \not\equiv 1 \pmod{12}$, elliptic curves of order $p+1$ are explicitly constructible over $GF(p)$. (We will show in the next section that the subgroup of order $p+1$ of $GF(p^2)^*$ is a useful auxiliary group for all p.) The following statements about the orders of curves of the form (2) or (3) in the case they are *not* $p+1$ are proved in [19] (see also [18]).

If $p \equiv 1 \pmod 4$, then p can uniquely be represented as the sum of two squares, i.e., $p = a^2 + b^2$. Then the curves $y^2 = x^3 - Dx$ have the orders

$$p + 1 \pm 2a \ , \quad p + 1 \pm 2b \ , \tag{4}$$

and the four orders occur equally often over the choices of D.

If $p \equiv 1 \pmod 3$, then p can uniquely be represented as $p = a^2 - ab + b^2$ with $a \equiv 2 \pmod 3$ and $b \equiv 0 \pmod 3$. Then the curves $y^2 = x^3 + D$ have the orders

$$p + 1 \pm 2a \ , \quad p + 1 \pm a \mp 2b \ , \quad p + 1 \pm (a + b) \ , \tag{5}$$

and the six orders occur equally often over the choices of D.

If $p \equiv 1 \pmod 4$ or $p \equiv 1 \pmod 3$, curves with the orders listed in (4) and (5) are explicitly constructible by varying D.

4.2. Subgroups of the multiplicative group of an extension field of $GF(p)$. In this section we investigate under what conditions a subgroup H of $GF(p^n)^*$ satisfies the properties of an auxiliary group in the technique for reducing the DL problem to the DH problem.

4.2.1. Representation with normal bases. We refer to [24] or [34] for an introduction to finite fields. For a prime power q, the field $GF(q^n)$ is an n-dimensional vector space over $GF(q)$ and hence its elements can be represented as n-tuples of $GF(q)$-elements with respect to some basis. Let α be an element of $GF(q^n)$, and let $\alpha_i := \alpha^{q^i}$ for $i = 0, \ldots, n-1$. Then $\{\alpha_0, \ldots, \alpha_{n-1}\}$ is called a *normal basis* if it is linearly independent in which case α is called a *normal element*. Let $\vec{\alpha} := (\alpha_0, \ldots, \alpha_{n-1})$. The matrix T in $(GF(q))^{n \times n}$ satisfying $\alpha_0 \vec{\alpha} = \vec{\alpha} T$ is called the *multiplication table* of the basis.

Normal elements can be found efficiently by trial and error, and when given q, n, and a normal element $\alpha \in GF(q^n)$, the multiplication table can be determined by solving a system of linear equations over $GF(q)$.

4.2.2. The use of subgroups H of $GF(p^n)^*$ as auxiliary groups. Let H be a subgroup of $GF(p^n)^*$. We derive conditions under which such subgroups are defined strongly algebraically over $GF(p)$. The group operation of H is a multiplication in $GF(p^n)^*$ and requires, in a normal basis representation, $O(n^3)$ multiplications in $GF(p)$. We conclude that every subgroup of $GF(p^n)^*$ (for n polynomial in $\log p$) is defined $(n, (\log p)^{O(1)})$-algebraically over $GF(p)$. For all n, $GF(p^n)^*$ is a cyclic group. This implies that a subgroup of $GF(p^n)^*$ is uniquely determined by its order $|H|$, or more precisely, for every divisor d of $|H|$ there exists exactly *one* subgroup of $GF(p^n)^*$ with $|H| = d$. Furthermore, all these subgroups are cyclic.

The next theorem states conditions on n and $|H|$ under which H is defined *strongly* algebraically over $GF(p)$.

THEOREM 6. *Let P be a fixed polynomial and c be a fixed constant. Let H be the subgroup of $GF(p^n)^*$ of order $|H|$. Then H is defined strongly (m, α)-algebraically over $GF(p)$ for $m, \alpha = (\log p)^{O(1)}$ if one of the following two conditions is satisfied.*

Condition 1. *$n \le P(\log p)$, and there exists a divisor $k < n$ of n such that*

$$|H| = \frac{p^n - 1}{p^k - 1} = p^{n-k} + p^{n-2k} + \cdots + p^k + 1 \ .$$

Condition 2. *$n \le c$, and there exists a non-constant polynomial $f(x)$ (with integer coefficients) dividing $x^n - 1$ such that $|H| = f(p)$.*

Remark. An alternative formulation of Condition 2 is that $|H|$ is a multiple of $\Phi_n(p)$ for some $n = O(1)$, where Φ_n stands for the n-th cyclotomic polynomial (see [24] and Appendix B). Examples are

$$\Phi_6(p) = p^2 - p + 1 \ ,$$
$$\Phi_8(p) = p^4 + 1 \ ,$$
$$\Phi_9(p) = p^6 + p^3 + 1 \ .$$

The alternating sums

$$p^{2l} - p^{2l-1} + - \cdots - p + 1$$

also satisfy Condition 2 for $l = O(1)$.

Proof. We show that if one of the conditions is satisfied there exists an EMBED algorithm that takes as input two elements x and e of $GF(p)$ and computes coordinates $\beta_1, \ldots, \beta_{n-1}$ (still in the normal basis representation) in $GF(p)$ by a polynomial number of algebraic operations such that $\beta = (x + e, \beta_1, \ldots, \beta_{n-1}) \in H$.

One possibility of designing the EMBED algorithm is to express membership of an element $\beta = (\beta_0, \ldots, \beta_{n-1})$ to the subgroup H by an equation (or a system of equations) in the coordinates. Then, the element $x + e$ can be assigned to one of the coordinates, say β_0, and the equation is solved for the remaining coordinates (by using only algebraic operations in the field $GF(p)$).

For an element β of $GF(p^n)^*$, we have that $\beta \in H$ if and only if $\beta^{|H|} = 1$. Clearly, this equation corresponds to a set of polynomial equations (with coefficients in $GF(p)$) in the coordinates β_i.

We will show that if the first condition is satisfied, then it is sufficient to solve one univariate polynomial equation over a subfield $GF(p^k)$ of $GF(p^n)$ for finding such a β, and that this can be reduced to a polynomial number of algebraic operations in the field $GF(p)$. The situation when the second condition is satisfied is more difficult. Here, a *system of multivariate* polynomial equations over $GF(p)$ has to be solved by algebraic operations. This can be achieved by Gröbner basis computations. The proof that Condition 2 is sufficient is given in Appendix B.

Proof that Condition 1 is sufficient. The EMBED algorithm works as follows in this situation. Let $x, e \in GF(p)$ be given. For $l := n/k$ let $\{\alpha_0, \ldots, \alpha_{k-1}\}$ and $\{\alpha'_0, \ldots, \alpha'_{l-1}\}$ be normal bases of $GF(p^k)$ over $GF(p)$ and of $GF(p^n)$ over $GF(p^k)$, respectively. For an element $\beta = (\beta_0, \ldots, \beta_{l-1}) \in GF(p^n)$ (with $\beta_i \in GF(p^k)$), we have that $\beta \in H$ is equivalent to

$$\beta^{(p^n-1)/(p^k-1)} = 1 \ . \tag{6}$$

Equation (6) is equivalent to

$$\left(\sum_{i=0}^{l-1} \beta_i \alpha'_i\right)^{p^{(l-1)k}+p^{(l-2)k}+\cdots+p^k+1} = 1 \ . \tag{7}$$

Now, we have $(\beta_i)^{p^{jk}} = 1$ (because $\beta_i \in GF(p^k)$) and $(\alpha'_i)^{p^{jk}} = \alpha'_{i+j}$ (where the index is reduced modulo l) by the definition of the normal basis. Hence (7) is equivalent to

$$\left(\sum_{i=0}^{l-1} \beta_i \alpha'_{i+l-1}\right) \cdot \left(\sum_{i=0}^{l-1} \beta_i \alpha'_{i+l-2}\right) \cdots \left(\sum_{i=0}^{l-1} \beta_i \alpha'_i\right) = 1 \tag{8}$$

(where the indices are reduced modulo l).

Because $(\beta^{(p^n-1)/(p^k-1)})^{p^k-1} = \beta^{p^n-1} = 1$, $\beta^{(p^n-1)/(p^k-1)}$ is an element of the subfield $GF(p^k)$ of $GF(p^n)$. Such elements are easy to characterize in terms of their coordinates. An element $(\gamma_0, \ldots, \gamma_{l-1})$ is an element of $GF(p^k) \subset GF(p^n)$ if and only if $\gamma_0 = \gamma_1 = \cdots = \gamma_{l-1}$. The reason for this fact is that $\alpha'_0 + \alpha'_1 + \ldots + \alpha'_{l-1}$ (the trace $\mathrm{Tr}(\alpha'_0)$ of α'_0) is also an element of $GF(p^k)$. Because both $\beta^{(p^n-1)/(p^k-1)}$ and 1 are elements of $GF(p^k)$, they are equal if and only if their first coordinates are equal. The equation coming from (8), restricted to the first coordinate, is equivalent to

$$g(\beta_0, \ldots, \beta_{l-1}) = 1/\mathrm{Tr}(\alpha'_0) \tag{9}$$

for some l-degree polynomial g with $GF(p^k)$-coefficients.

The construction of a group element β of H with the desired property now works as follows. Let the first coordinate $\beta_{0,0}$ of β_0 (note that β_i corresponds to a k-tuple $(\beta_{i,0}, \ldots, \beta_{i,k-1})$ of $GF(p)$-elements with respect to the normal basis $\alpha_0, \ldots, \alpha_{k-1}$) be equal to $x+e$. Choose the coefficients $\beta_{0,1}, \ldots, \beta_{0,k-1}$ and the coefficients $\beta_1, \ldots, \beta_{l-2}$ randomly in $GF(p)$ and in $GF(p^k)$, respectively. Then (9) is equivalent to a polynomial equation for β_{l-1} with coefficients in $GF(p^k)$.

The roots of a polynomial $f(\gamma)$ over a finite field $GF(p^k)$ can be computed in probabilistic polynomial time by the Cantor-Zassenhaus algorithm (see [34], [24]). The key idea of this algorithm is to factor the polynomial $f(\gamma)$ into

$$\gcd(f(\gamma), (\gamma+\delta)^{\frac{p^k-1}{2}} - 1) \quad \text{and} \quad \gcd(f(\gamma), (\gamma+\delta)^{\frac{p^k-1}{2}} + 1)$$

for random $\delta \in GF(p^k)$. This is repeated with different δ and leads to the linear factors of $f(\gamma)$.

The computation of polynomial gcd's, and thus the entire root-finding algorithm, require only algebraic operations in $GF(p^k)$, and the latter can be reduced to algebraic operations (and equality tests) in $GF(p)$ (with respect to the normal basis representation). The expected number of solutions for β_{l-1} is roughly 1 because $|H|/p^n \approx 1/p^k$. If no solution is found, then failure is reported.

Because the Cantor-Zassenhaus algorithm has probabilistic running time polynomial in n and $\log p$ and uses only algebraic operations in $GF(p)$, the required executions of the EMBED procedure run in a probabilistic polynomial (in $\log |G|$) number of algebraic operations if n is polynomial in $\log p$. □

4.3. Summary. The following corollary is an immediate consequence of Theorem 2, combined with the results of this section.

COROLLARY 7. *Let P be a fixed polynomial, let G be a cyclic group with generator g, and let $B := P(\log |G|)$. Then there exists a list of expressions $A(p)$ in p with the following property: if every prime factor p of $|G|$ greater than B is single and if for every such prime factor at least one of the expressions $A(p)$ is B-smooth, then breaking the DH protocol in G with respect to g is polynomial-time equivalent to computing discrete logarithms in G to the base g. The list contains the following expressions:*

$$p-1\ ,\ p+1\ ,\ p+1\pm 2a\ ,$$

if $p \equiv 1 \pmod 4$, where $p = a^2 + b^2$,

$$p+1\pm 2a\ ,\ p+1\mp a\pm 2b\ ,\ p+1\pm(a+b)\ ,$$

if $p \equiv 1 \pmod 3$, *where* $p = a^2 - ab + b^2$, $a \equiv 2 \pmod 3$, *and* $b \equiv 0 \pmod 3$,

$$\frac{(p^k)^l - 1}{p^k - 1} = (p^k)^{l-1} + \cdots + p^k + 1 \ ,$$

where $k, l = (\log p)^{O(1)}$, *and* $f(p)$, *where* $f(x) \in \mathbf{Z}[x]$ *is a nonconstant polynomial dividing* $x^n - 1$ *for some* $n = O(1)$. □

5. Equivalence between variants of the DH problem.

5.1. Introduction. In the previous sections we have proved results concerning the relationship between the security of the DH protocol and the hardness of the DL problem. However, in order to prove that the DH protocol is secure for a group in which the DL problem is hard, one has to show that the DH problem cannot be solved efficiently even with small probability of, say, 1%. Motivated by this, we show in this section that the assumption of a *perfect* DH oracle for the reduction process is unnecessarily strong and can be relaxed in many ways. In 5.2 we prove that a (probabilistic) DH oracle answering correctly with small probability is virtually as strong as a perfect DH oracle. For example, an oracle answering correctly with probability 1% can efficiently be transformed into an oracle that answers correctly with arbitrarily high probability.

In section 5.3, it is shown that the same holds for a DH oracle that answers correctly for the input (g^u, g^v) only if $u = v$. Finally, the relationship between the DH problem in G and in subgroups of G is investigated in section 5.4.

5.2. ε-DH-oracles. This section deals with DH oracles that answer correctly only with small (but nonnegligible) probability. It is shown that such oracles are virtually as strong as perfect DH oracles. The problem of correcting faulty DH oracles was considered independently by Shoup [45], who described a quite different approach. We introduce the notion of an ε-DH-oracle for a cyclic group G with respect to a generator g. Note that such an "oracle" is probabilistic in general rather than deterministic.

DEFINITION 7. *For* $\varepsilon > 0$, *an* ε-DH-oracle *is a probabilistic oracle which returns for an input* (g^u, g^v) *the correct answer* g^{uv} *with probability at least* ε *if the input is uniformly distributed over* $G \times G$.

The offset *of the oracle's answer* g^t *to the input* (g^u, g^v) *is defined as* $t - uv \pmod{|G|}$. *A* translation-invariant ε*-DH-oracle is an* ε*-DH-oracle whose offset distribution is the same for every input* (g^u, g^v).

A special case of (nontranslation-invariant) ε-DH-oracles are *deterministic* oracles answering correctly for a fraction ε of all inputs. We proceed in two steps to prove that an ε-DH-oracle can be transformed into a virtually perfect DH oracle. First, the oracle is made translation-invariant by randomization of the input, and then, the translation-invariant oracle is "amplified" to an (almost) perfect oracle.

LEMMA 1. *An* ε*-DH-oracle for a cyclic group* G *with order* $|G|$ *can efficiently be transformed into a translation-invariant* ε*-DH-oracle. More precisely, implementing one call to the latter requires one call to the former and* $O(\log |G|)$ *group operations.*

Proof. Given the group elements $a = g^u$ and $b = g^v$ we can randomize the input by choosing r and s at random from $[0, |G| - 1]$, providing the oracle with $a' = ag^r$ and $b' = bg^s$ and multiplying the oracle's answer $g^{(u+r)(v+s)+t} = g^{uv+rv+su+rs+t}$ with $(a^{-1})^s \cdot (b^{-1})^r \cdot g^{-rs} = g^{-(rv+su+rs)}$ to obtain g^{uv+t}. Note that a' and b' are random group elements and statistically independent of a and b. The ε-DH-oracle with randomized input is thus a translation-invariant ε-DH-oracle. □

Remark. If $|G|$ is unknown the input can also be randomized, where the random numbers are chosen from a larger interval. The resulting ε-DH-oracle is then "almost translation-invariant" and applicable in the proof of Theorem 8 if the interval is of size at least $2 \cdot |G|/(\varepsilon^2 \cdot \min\{s, 0.1\})$ (where s is as in Theorem 8). This is the reason for the greater number of group operations for this case in Theorem 8.

In the proof of Theorem 8 it is shown that a translation-invariant ε-DH-oracle can be transformed into an almost-perfect DH oracle. The straightforward approach to using a translation-invariant ε-DH-oracle may at first sight appear to be to run it $O(1/\varepsilon)$ times until it produces the correct answer. However, because the Diffie–Hellman decision problem is difficult, a more complicated approach must be used. (The Diffie–Hellman decision problem, which was first mentioned in [5], is, for given g^u, g^v, and g^w, to decide whether $g^w = g^{uv}$, and is of course at most as difficult as the DH problem.)

THEOREM 8. *For every cyclic group G with generator g and known order $|G|$ and for every $\beta > 0$ there exists an algorithm for solving the DH problem in G which makes calls to an ε-DH-oracle and whose answer is correct with probability at least $1 - \beta$. The number of required oracle calls is $O(\log(1/\beta\varepsilon)/\varepsilon^4)$. If the order of G is unknown, then the reduction is also possible if all the prime factors of $|G|$ are greater than $(1 + s)/\varepsilon$ for some $s > 0$. The number of required calls to the ε-DH-oracle is then*

$$O\left(\frac{1}{(\varepsilon^2 \cdot \min\{s, 1\})^2} \cdot \log \frac{1}{\beta\varepsilon}\right) .$$

The number of required group operations is $O(\log |G|)$ times the number of oracle calls if $|G|$ is known and $O(\log(|G|/(\varepsilon^2 \cdot \min\{s, 1\})))$ times this number if $|G|$ is not known, respectively.

For the proof of Theorem 8 we need the following lemma.

LEMMA 2. *Let $X_1, X_2, X_3, \ldots$ be independent binary random variables with identical distribution P_{X_i} with $P_{X_i}(1) = p$. Let further $\alpha, \delta' > 0$. If t is the smallest number such that the event*

$$\frac{X_1 + \cdots + X_t}{t} \in [p - \delta', p + \delta']$$

has probability at least $1 - \alpha$, then $t = O(\log(1/\alpha)/\delta'^2)$.

Proof. Since the random variables X_i are independent, we have

$$\frac{\mathrm{Var}(X_1 + \cdots + X_t)}{t} = \frac{t \cdot \mathrm{Var}(X_i)}{t^2} = \Theta(1/t) .$$

Hence the number of standard deviations corresponding to δ' is of order $\Theta(\delta'\sqrt{t})$. The normal approximation of the binary distribution (see for example [15]) leads to $\delta'\sqrt{t} = \Theta((\log(1/\alpha))^{1/2})$ or $t = \Theta(\log(1/\alpha)/\delta'^2)$. □

Proof of Theorem 8. The basic idea of the amplification of the DH oracle is as follows. In a precomputation phase, which is independent of the actual input, the oracle's offset distribution is determined. Then, the oracle is called with the given input to compute the correct solution with overwhelming probability.

More precisely, the reduction from an ε-DH-oracle to an oracle answering correctly with high probability consists of the following steps which we first describe intuitively.

Step 1. The ε-DH-oracle is transformed into a *translation-invariant* ε-DH-oracle.

Step 2. We compute an estimate ε' for the probability that the (translation-invariant) oracle answers correctly.

Step 3. A list L_1 of group elements g^e is computed with the property that g^e is contained in L_1 if and only if the probability of the offset e is close to ε'.

Step 4. A second list L_2 of group elements is generated which contains exactly those group elements that occur with frequency close to ε' when the oracle is called with the input (g^u, g^v).

Step 5. The lists L_1 and L_2 have (with high probability) the property that the elements of L_2 are exactly the elements of L_1 multiplied by the group element g^{uv} (which is itself contained in L_2). In order to determine this switch element, the lists aL_1 are generated for all elements a in L_2 (the list aL_1 contains exactly the elements al_1, where l_1 is contained in L_1). The list L_2 is compared to all the lists aL_1, and equality yields a candidate a for g^{uv}.

Step 6. In case of one single candidate a for g^{uv}, this is the output of the algorithm. In the case of *several* candidates and if the group order $|G|$ is known, the discrete logarithms of all the candidates and of g^u and g^v are determined modulo the smooth part of $|G|$. This yields the correct candidate for g^{uv}, which is then the output of the algorithm.

Note that the first three steps are a precomputation which is independent of the particular input (g^u, g^v). The list L_1 which is generated in these steps is a reference list describing the offset behavior of the faulty oracle. We describe the steps in detail and analyze their correctness and efficiency.

Step 1. According to Lemma 1, one can construct a translation-invariant ε-DH-oracle which uses $O(\log |G|)$ group operations and one call to an ε-DH-oracle per call if $|G|$ is known. If $|G|$ is unknown, the number of group operations is $O(\log(|G|/(\varepsilon^2 \cdot \min\{s, 1\})))$.

Step 2. Let $\alpha := \beta\varepsilon/8$. An event with probability at least $1 - \alpha$ will be called *almost certain*. Let $\delta := \varepsilon/10$ and $\delta' := \delta\varepsilon/100 = \varepsilon^2/1000$.[2] If ε is not known, we take a lower bound. In order to determine the probability of a correct answer, the translation-invariant oracle is called repeatedly with the input (g^0, g^0), and ε' is the fraction of correct answers g^0. The number t of oracle calls is such that the true probability of a correct answer lies almost certainly in the interval $[\varepsilon' - \delta', \varepsilon' + \delta']$. It follows from Lemma 2 that $t = O(\log(1/\alpha)/\delta'^2)$.

Step 3. In this step the reference list L_1 is generated as follows. The faulty oracle is called t times (for the same value of t as in the previous step), and all the occurring group elements are stored. Let the list L_1 consist of those group elements whose fraction in the set of all answers lies in the interval $[\varepsilon' - (\delta + \delta'), \varepsilon' + (\delta + \delta')]$. According to Lemma 2, and because $2/\varepsilon$ is an upper bound on the number of offsets occurring with probability at least $\varepsilon/2$, with probability $(1 - \alpha)^{4/\varepsilon}$ the following two statements are both true.

1. If e is an offset with probability in $[\varepsilon' - \delta, \varepsilon' + \delta]$, then g^e is contained in L_1.
2. If g^e is in L_1, then the offset e has probability in $[\varepsilon' - (\delta + 2\delta'), \varepsilon' + (\delta + 2\delta')]$.

Step 4. The translation-invariant faulty oracle is called repeatedly wth the input (g^u, g^v), where (g^u, g^v) is the input to the DH algorithm for G. Let the list L_2 then consist of those group elements which occur as answers of the oracle with a frequency in $[\varepsilon' - (\delta + 3\delta'), \varepsilon' + (\delta + 3\delta')]$. Then, for the same number of trials t as in the previous step, with probability at least $(1 - \alpha)^{4/\varepsilon}$ the following statements are true.

1. If e is an offset with probability in $[\varepsilon' - (\delta + 2\delta'), \varepsilon' + (\delta + 2\delta')]$, then g^{uv+e} is contained in L_2.

[2] The proof does not depend on the choice of the constants (e.g., 1/10), which is somewhat arbitrary. Intuitively, we need that $\delta \ll \varepsilon$ and $\delta' \ll \varepsilon\delta$.

2. If g^{uv+e} is in L_2, then the offset e has probability in $[\varepsilon'-(\delta+4\delta'), \varepsilon'+(\delta+4\delta')]$.

Step 5. With high probability, the list L_2 is equal to L_1, switched by g^{uv} (which is itself in L_2). This allows to determine g^{uv}. More precisely, it follows from the analysis of steps 1 to 4 that the probability that L_1 contains *all* the offsets which have their probability in the interval $[\varepsilon' - \delta, \varepsilon' + \delta]$, that all the offsets of L_1 also occur in L_2, and that all the offsets of L_2 have probability in $[\varepsilon' - (\delta + 4\delta'), \varepsilon' + (\delta + 4\delta')]$ is at least

$$(1-\alpha)^{8/\varepsilon} \geq 1 - \frac{8\alpha}{\varepsilon} = 1 - \beta \; . \tag{10}$$

If this is fulfilled, then L_2 contains more elements than L_1 only if there exists an offset whose probability is in the set

$$[\varepsilon' - (\delta + 4\delta'), \varepsilon' - \delta] \cup [\varepsilon' + \delta, \varepsilon' + (\delta + 4\delta')] \; . \tag{11}$$

In this case we replace δ by $\delta + i \cdot 5\delta'$ (for an integer i randomly chosen in $[-2/\varepsilon, 2/\varepsilon]$), leave δ' unchanged, and run the entire algorithm (except Steps 1 and 2) again. Because the sets (11) are disjoint for different i, and because there can be at most $2/\varepsilon$ offsets with probability at least $\varepsilon/2$, L_1 and L_2 contain the same number of elements for at least half of the possible choices for i.

If the lists L_1 and L_2 have equal length, then with probability at least $1 - \beta$ we have that g^{uv} is contained in L_2 and $L_2 = g^{uv}L_1$, i.e., L_2 contains exactly the elements $g^{uv}l_1$ for l_1 in L_1. The lists aL_1 are computed for all elements a of L_2 and compared to L_2. If equality holds, then a is a candidate for g^{uv}.

Step 6. Let c be the number of elements of L_1 and L_2. If there exists only *one* candidate for g^{uv}, then this group element is the output of the algorithm. If there exist *several* such elements, this means that the lists have a nontrivial translation symmetry, or more precisely, that they are invariant under a multiplication with $g^{|G|/c'}$ for a divisor c' of c and $|G|$. Let c' be the maximal number with this property. Note that $|G|$ has a factor $c' \leq c \leq 1/(\varepsilon' - (\delta + 2\delta'))$ in this case. There are c' candidates for g^{uv}, namely

$$g^{uv}, \; g^{uv+\frac{|G|}{c'}}, \ldots, \; g^{uv+(c'-1)\frac{|G|}{c'}} \; .$$

We show that if $|G|$ is known, the correct candidate can be determined. Let $p_1, \ldots, p_l$ be the distinct prime factors of c' (they can be found in time $O((\log(1/\varepsilon))^2/\varepsilon)$ because $c' = O(1/\varepsilon)$), i.e., $c' = \prod_{i=1}^{l} p_i^{f_i}$. Let further $d = \prod_{i=1}^{l} p_i^{e_i}$ be the product of the maximal powers of the p_i dividing $|G|$. The number d can be computed in time $O((\log|G|)^3)$ and is $(2/\varepsilon)$-smooth because $c' \leq c \leq 2/\varepsilon$. Hence u and v (and consequently uv) are computable modulo d from g^u and g^v by the Pohlig–Hellman algorithm by $O((\log|G|)^2 + \log|G|/\varepsilon)$ group operations. Analogously, we can also compute the discrete logarithms of all the candidates modulo d.

The discrete logarithm of exactly one of the candidates has the correct remainder with respect to d. This is true because for every i, exactly every $p_i^{f_i}$-th candidate has the correct remainder with respect to $p_i^{e_i}$. The primes are distinct; thus the $p_i^{f_i}$ are relatively prime, and hence every $\prod_{i=1}^{l} p_i^{f_i}$-th candidate, that is *exactly one* of them, has the correct remainder. This group element is the output of the algorithm.

In the case where $|G|$ is *not* known, this last step of finding the correct candidate does not work. The only possibility is to choose a smaller value for δ. This is always successful if all the prime factors of $|G|$ are greater than $(1 + s)/\varepsilon$ for some positive

s. Then δ must be chosen smaller than $s\varepsilon/2$, such that $1/(\varepsilon' - (\delta + 2\delta')) < (1+s)/\varepsilon$ holds. The last inequality implies that such a symmetry of the lists L_1 and L_2 (this symmetry is a necessary condition for the case of more than one candidate for g^{uv}) is not possible. □

Remark. Examples of ε-DH-oracles which can *not* be transformed into perfect oracles with our method when $|G|$ is unknown are those which answer the input (g^u, g^v) by one of the values $g^{uv+i|G|/z}$, where $z \leq 1/\varepsilon$ is a factor of $|G|$, and where all the values of i between 0 and $z-1$ are equally likely.

Note that a DH oracle as obtained in Theorem 8 is virtually equivalent to a perfect DH oracle in a polynomial-time (or subexponential-time) reduction of the DL to the DH problem because the correctness of the output of a probabilistic algorithm computing discrete logarithms can be tested, and because only a polynomial (or subexponential) number of oracle calls is required for the computation of a discrete logarithm.

5.3. The squaring oracle. We describe an example of an oracle that is weaker than an ε-DH-oracle with respect to the fraction of correctly answered inputs. Nevertheless, the oracle turns out to be as strong as the perfect oracle. We call an oracle that answers the input g^u by $g^{(u^2)}$ (where u and u^2 are in $\mathbf{Z}_{|G|}$) a *squaring-DH-oracle*.

From g^u and g^v one can compute $g^{u+v} = g^u \cdot g^v$, and with the squaring-DH-oracle

$$g^{(u+v)^2} \cdot \left(g^{(u^2)}\right)^{-1} \cdot \left(g^{(v^2)}\right)^{-1} = g^{(u+v)^2-u^2-v^2} = g^{2uv} = (g^{uv})^2 \ . \tag{12}$$

When given $|G|$, the square root g^{uv} of $(g^{uv})^2$ can efficiently be computed. If $|G|$ is odd, the square root is unique, but if $|G|$ is even, there exist two square roots,

$$g^{uv} \quad \text{and} \quad g^{uv+\frac{|G|}{2}}$$

which can be computed efficiently (see Lemma 3). Let $|G|$ be even, and let 2^e be the maximal power of 2 dividing $|G|$. From g^u and g^v, one can compute u and v, and hence uv, modulo 2^e with $O((\log|G|)^2)$ group operations by the Pohlig–Hellman algorithm. Because $|G|/2$ is not a multiple of 2^e, we have

$$uv \not\equiv uv + \frac{|G|}{2} \pmod{2^e} \ ,$$

and one can determine the correct root g^{uv} by computing the discrete logarithms of one of the roots modulo 2^e. Hence a squaring-DH-oracle is equally powerful as a perfect DH oracle in a group G whose order is known.

A probabilistic squaring-DH-oracle for a group with known order that answers correctly only with probability ε (an *ε-squaring-DH-oracle*) can be transformed into a translation-invariant ε^3-DH-oracle by randomizing the inputs in (12). The complexity is $O((\log|G|)^2)$ group operations per call. This proves the following theorem.

THEOREM 9. *For every cyclic group G with generator g and known order $|G|$ and for every $\beta > 0$ there exists an algorithm solving the DH problem in G which makes calls to an ε-squaring-DH-oracle and whose answer is correct with probability at least $1-\beta$. The number of oracle calls is $O(\log(1/\beta\varepsilon^3)/\varepsilon^{12})$. The number of required group operations is $O((\log|G|)^2)$ times the number of oracle calls.*

5.4. The security of subgroups. Throughout this section we assume that the order of G and its factorization are known. We address the question whether a

subgroup is more or less secure than the entire group with respect to the DH protocol. Although the statement of Corollary 12 below is very intuitive (and an analogous result holds for the computation of discrete logarithms), the proofs of Theorems 10 and 11 are not trivial. First we give a criterion when a DH oracle for $\langle g\rangle$ can be efficiently transformed into a DH oracle for $\langle g^r\rangle$. More precisely, we will show that a subgroup of G is at most as secure as G with respect to the DH protocol if every large prime factor of the index of the subgroup occurs with the same multiplicity in the index and in the group order. We need the following lemma on the computation of p-th roots in a cyclic group G if p is a multiple prime factor of $|G|$. Note that for *single* prime factors p of $|G|$, a p-th root can be obtained by computing the z-th power for $z :\equiv p^{-1} \pmod{|G|/p}$.

LEMMA 3. *Let G be a cyclic group with generator g, and let p be a multiple prime divisor of $|G|$. One of the p-th roots of a p-th power in G can be computed in time $O((\log|G|)^2 + p\log|G|)$.*

Proof. The square root algorithm of Massey [25] can be generalized as follows. Let $|G| = p^j s$ (where $j \geq 2$ and $(p,s) = 1$), and let h be a p-th power in G. By the Pohlig–Hellman algorithm we can compute the remainder k of the discrete logarithm of h to the base g with respect to p^j. Note that k is a multiple of p because h is a p-th power. Let $d :\equiv -s^{-1} \pmod{p}$. The element

$$\left(g^{s\cdot\frac{k}{p}\cdot d}\right)^{-1} \cdot h^{\frac{sd+1}{p}}$$

is a p-th root of h. This algorithm requires $O((\log|G|)^2 + p\log|G|)$ operations in G. □

Remark. When memory space is available, this algorithm can be sped up to $O(\sqrt{p}\cdot(\log|G|)^{O(1)})$ by the baby-step giant-step trade-off in the Pohlig–Hellman algorithm. This running time is optimal: it was shown in [28] that no generic algorithm can compute p-th roots substantially faster in a group whose order is divisible by p^2 (even when given a DH oracle for this group).

THEOREM 10. *Let P be a fixed polynomial. Let G be a cyclic group with generator g. If the number r is such that every prime factor of r is either smaller than $B := P(\log|G|)$ or has at least the same multiplicity in r as in G, then there exists an algorithm solving the DH problem in the group $\langle g^r\rangle$, making one call to the DH oracle for $\langle g\rangle$ and using a polynomial number of group operations per call.*

Remark. Again, the conditions of the theorem are optimal. Shoup [45] has shown that if the conditions are not satisfied, then the construction of a subgroup oracle from an oracle for G is hard in the generic model.

Proof. Let $|G| = \prod p_i^{e_i}$ and $r = \prod p_i^{f_i}$ (where $f_i > e_i$, $e_i = 0$, or $f_i = 0$ is possible). The DH algorithm for the group $\langle g^r\rangle$ takes as inputs two elements $(g^r)^a$ and $(g^r)^b$ and must output $(g^r)^{ab}$. Using the DH oracle for the group $G = \langle g\rangle$ with the same input, one obtains g^{r^2ab}, i.e., the r-th power of g^{rab}. Now, g^{rab} is computed from g^{r^2ab} by computing the r-th root. More precisely, the $p_i^{f_i}$-th root of g^{r^2ab} has to be computed for all i with $f_i > 0$, and the correct root, i.e., the particular root that is a power of g^{rab}, must be determined. Assume that we have already computed

$$g^{p_1^{f_1}\cdots p_{i-1}^{f_{i-1}} p_i^{2f_i}\cdots p_s^{2f_s} ab} = g^{cp_i^{2f_i}ab} =: d_i\ ,$$

where c is explicitly known. We describe the computation of the correct $p_i^{f_i}$-th root of this group element separately for the cases $f_i \geq e_i$ and $p_i \leq B$.

Case 1: $f_i \geq e_i$. We compute z with

$$z :\equiv (p_i^{f_i})^{-1} \pmod{|G|/p_i^{e_i}}$$

and d_i^z, which is the desired group element. First, it is a $p_i^{f_i}$-th root of d_i. Additionally, it is the only $p_i^{f_i}$-th root of this element which is a power of $g^{p_i^{f_i}}$ (the $p_i^{e_i} - 1$ different roots are

$$g^{cp_i^{2f_i}abz+i|G|/p_i^{e_i}}$$

for $i = 1, \ldots, p_i^{e_i} - 1$, and they are not even powers of $g^{p_i^{e_i}}$).

Case 2: $p_i \leq B$ and $f_i < e_i$. Here we repeat the following two steps f_i times.

Step 1. Compute the p_i-th roots of the group element.

Step 2. Decide which of the roots is a power of g^{rab} and continue with this element.

Assume for some $k = 2f_i - 1, 2f_i - 2, \ldots, f_i$ that we have already computed

$$g^{p_1^{f_1} \cdots p_{i-1}^{f_{i-1}} \cdot p_i^{k+1} \cdot p_{i+1}^{2f_{i+1}} \cdots p_s^{2f_s} ab} = g^{c' p_i^{k+1} ab} ,$$

where c' is explicitly known. Then the two steps work as follows.

Step 1. According to Lemma 3 we can compute a p_i-th root of the group element in time $O((\log |G|)^2 + p_i \log |G|)$.

Step 2. Because a and b can be obtained modulo $p_i^{e_i - f_i}$ directly from g^{ra} and g^{rb} by the Pohlig–Hellman algorithm and c' is explicitly known, and because $k \geq f_i$, we can compute $c' p_i^k ab$ modulo $p_i^{e_i}$. From the root obtained in Step 1, all the roots

$$g^{c' p_i^k ab + j \cdot |G|/p_i} \qquad (j = 0, \ldots, p_i - 1)$$

can be computed. We have $j \cdot |G|/p_i \equiv 0 \pmod{p_i^{e_i}}$ only for $j = 0$, and the correct group element $g^{c' p_i^k ab}$ can be determined by computing the discrete logarithms of the candidates modulo $p_i^{e_i}$, using the Pohlig–Hellman algorithm.

The entire procedure, executed for all prime factors p_i of r, ends up with g^{rab}, and the running time of the algorithm is polynomial in $\log |G|$. □

Remark. It has been pointed out in a preliminary version of [4] that in case of a generator change, i.e., if $(r, |G|) = 1$, it is not even necessary to know r. Let $h = g^r$, and let DH_g and DH_h be the DH functions in G with respect to the generator g and h, respectively. Then

$$\begin{aligned} \mathrm{DH}_h(h^a, h^b) &= h^{ab} = g^{rab} = \mathrm{DH}_g(g^{r^2 ab}, g^{r^{-1}}) \\ &= \mathrm{DH}_g(\mathrm{DH}_g(h^a, h^b), \mathrm{PDH}_{g,\varphi(|G|)-1}(h)) , \end{aligned}$$

and the last expression can be computed by $O(\log |G|)$ applications of the oracle with respect to the basis g.

In many cases a DH oracle for a subgroup of G or a set of such oracles can be transformed into a DH oracle for the entire group, and the following theorem gives a criterion for when this is the case.

THEOREM 11. *Let P be a fixed polynomial. Let G be a cyclic group with generator g and order $|G| = \prod_{i=1}^{r} p_i^{e_i}$, and let $B := P(\log |G|)$ be a smoothness bound. If for all $p_i > B$ a number s_i, where p_i does not divide s_i, and a DH oracle for the group $\langle g^{s_i} \rangle$ is given, then there exists a polynomial-time algorithm solving the DH problem in G with respect to g which calls each oracle for such a subgroup once.*

Proof. Let g^u and g^v be given. We compute g^{uv} using the available oracles for subgroups. Let $m_i := p_i^{e_i}$, $M_i := |G|/m_i$, and $N_i :\equiv M_i^{-1} \pmod{m_i}$. For prime factors $p_i \leq B$, u and v, and hence also uv, can be computed in polynomial time modulo m_i by the Pohlig–Hellman algorithm. For a prime factor $p_i > B$, assume that a DH oracle for the subgroup $\langle g^{s_i} \rangle$ is given, where p_i does not divide s_i. We apply the oracle for $\langle g^{s_i} \rangle$ to $(g^{s_i})^u = (g^u)^{s_i}$ and $(g^{s_i})^v$ to obtain $(g^{s_i})^{u \cdot v}$, where u, v and $u \cdot v$ are modulo $|G|/s_i$. Let $z_i :\equiv s_i^{-1} \pmod{m_i}$ and

$$U_i := \left(g^{s_i(u \cdot v)}\right)^{M_i \cdot z_i} = g^{M_i \cdot (u \cdot v)} \ ,$$

where $u \cdot v$ is modulo m_i. Finally, g^{uv} is computable by Chinese remaindering with implicitly represented arguments by applying only group operations in G:

$$g^{uv} = g^{\sum_i M_i N_i (u \cdot v)} = \prod_i U_i^{N_i}. \qquad \square$$

The following result is an immediate consequence of the above theorems.

COROLLARY 12. *Consider a group $G = \langle g \rangle$ and a subgroup $H = \langle g^k \rangle$ of G with $(\log |G|)^{O(1)}$-smooth index. The DH problem for H is polynomial-time equivalent to the DH problem for G.*

6. Concluding remarks. We have presented a technique for reducing the DL problem in a group G to the DH problem in the same group efficiently when suitable auxiliary groups are given. One conclusion of this fact is that, under a plausible but unproven assumption on the existence of smooth numbers, for every group whose order does not contain a multiple large prime factor there exists a polynomial-time algorithm computing discrete logarithms and making calls to a DH oracle for the same group. In the generic model, it was proven that such a reduction cannot exist for groups whose order is divisible by the square of a large prime. A second conclusion is that solving the DH and DL problems is computationally equivalent for many classes of groups in a uniform sense. These are the groups for which suitable auxiliary groups can be efficiently constructed.

Throughout this paper, we have assumed to know the group order and its factorization. Let p be a large prime factor of $|G|$. If an appropriate auxiliary group over $GF(p)$ such as a subgroup of the multiplicative group of a finite field or an elliptic curve is given that has smooth order, then p can be found efficiently as a factor of $|G|$ (see [23] and [2]). This fact indicates a close relationship between the problems of integer factoring and proving the equivalence between the DH and DL problems.

In Appendix C we describe a technique, presented in [48] and independently considered in [4], for obtaining stronger results under the assumption that efficient algorithms exist for solving the DH problem in certain groups, and which use only algebraic operations. The idea is to execute these algorithms on implicitly represented arguments. This allows to iterate the technique by computing with multiply implicitly represented elements. It is then no longer necessary that for every large prime factor p of $|G|$ a smooth auxiliary group H_p is known. For example, a cyclic auxiliary group H_p whose order contains a large prime factor q and a smooth auxiliary group H_q over $GF(q)$ are sufficient under the assumption that a polynomial-time DH algorithm exists for H_p which uses only algebraic operations in $GF(p)$.

Appendix A: Finding generator sets of the auxiliary groups. We show how a generator set can be found efficiently in a (additively written) finite Abelian

group H of rank r and with B-smooth order $|H| = \prod_{i=1}^{l} q_i^{f_i}$. Suppose $H \cong \mathbf{Z}_{n_1} \times \cdots \times \mathbf{Z}_{n_r}$ (such that n_{j+1} divides n_j for $j = 1, \ldots, r-1$), where the numbers r and $n_1, \ldots, n_r$ are not known a priori, and suppose that we have already found n_i and points h_i with order n_i in $H/\langle h_1, \ldots, h_{i-1}\rangle$ for $i = 1, \ldots, j-1$. Let $n_1 = \prod q_i^{g_i}$, and let π_{j-1} denote the canonical projection to the quotient group $H/\langle h_1, \ldots, h_{i-1}\rangle$, i.e., $\pi_{j-1}(u)$ is the element of $H/\langle h_1, \ldots, h_{j-1}\rangle$ containing u. For the construction of h_j such that $\pi_{j-1}(h_j)$ has maximal order in $H/\langle h_1, \ldots, h_{j-1}\rangle$, we choose $O(\log\log|H|)$ points h in H at random and compute $\mathrm{ord}_{H/\langle h_1,\ldots,h_{j-1}\rangle}\pi_{j-1}(h)$ by comparing

$$\frac{n_1}{q_i^k}\pi_{j-1}(h) = \pi_{j-1}\left(\frac{n_1}{q_i^k}h\right) \quad (\text{for } i = 1, \ldots, l \text{ and } k = g_i, g_i - 1, \ldots, 0)$$

with the unity $e_{H/\langle h_1,\ldots,h_{j-1}\rangle}$ of the quotient group. Comparing $\pi_{j-1}(h')$ and $e_{H/\langle h_1,\ldots,h_{j-1}\rangle}$ is equivalent to deciding if $h' \in \langle h_1, \ldots, h_{j-1}\rangle$, which is done by the generalized Pohlig–Hellman DL algorithm described in section 3. This leads to an element h_j with maximal order in $H/\langle h_1, \ldots, h_{j-1}\rangle$.

It is possible that the algorithm makes a mistake here, i.e., that the generated element does not have maximal order. Such an error occurs only with probability exponentially small in the number of trials and can be detected as follows. In the case where $\mathrm{ord}_{H/\langle h_1,\ldots,h_{j-1}\rangle}\pi_{j-1}(h_j)$ does not divide $\mathrm{ord}_{H/\langle h_1,\ldots,h_{j-2}\rangle}\pi_{j-2}(h_{j-1})$, the process must be restarted because one of the preceding points has not had maximal order. The same holds if $j > \max\{f_i\}$. The latter is a bound for the rank r of H.

The algorithm stops if $\langle h_1, \ldots, h_r\rangle = H$, that is

$$H/\langle h_1, \ldots, h_r\rangle = \{e\} \ ,$$

and $\{h_1, \ldots, h_r\}$ is then a generator set of H. Every element c of H has a unique representation $c = \sum_{j=1}^{r} k_j h_j$ with $k_j \in \{0, \ldots, n_j - 1\}$ with respect to this set. The expected number of operations in H to determine the generator set is

$$O\left(r^2 \log\log|H| \cdot \frac{\sqrt{B^r}}{\log B}(\log|H|)^3\right)$$

(using the time-memory trade-off in the Pohlig–Hellman algorithm).

Appendix B: Gröbner basis computations and the completion of the proof of Theorem 6. The goal of this appendix is to complete the proof of Theorem 6, i.e., to show that the second condition also implies that H is defined strongly algebraically over $GF(p)$. In the first part of the proof of Theorem 6 (given in section 4.2), the key argument was that the Cantor-Zassenhaus algorithm allows to solve a univariate polynomial equation over a finite field efficiently and with algebraic operations only. This led to an EMBED algorithm with the required properties.

For the second part of this proof the result is required that a *system* of such equations can be solved. In section B.1 we shortly describe the concept of Gröbner bases, which are a tool for solving such systems. Section B.2 completes the proof of Theorem 6.

B.1 Gröbner bases. Let $\mathbf{F}$ be a field and R be the ring $\mathbf{F}[x_1, \ldots, x_n]$ of the polynomials in $x_1, \ldots, x_n$ over $\mathbf{F}$. Let further $p_i = 0$ (where $i = 1, \ldots, l$ and $p_i \in R$ for all i) be a system of polynomial equations. We also write $P = 0$, where $P := \{p_i\}$. Every basis of the generated ideal $\langle P\rangle$ in the ring R leads to an equivalent system of

equations. Gröbner bases with respect to the lexicographic term ordering have the property that the system can be solved if univariate equations can be solved. The lexicographic term ordering is defined as follows:

$$\prod x_j^{i_j} <_L \prod x_j^{i'_j}$$

if and only if $i_j = i'_j$ for $j = 1, \ldots, l-1$ and $i_l < i'_l$ for some l.

We motivate the definition of Gröbner bases[3]. Let f and g be polynomials, and let t be the leading term of g. One can reduce f modulo g if a monomial of f is a multiple of t, $f = \alpha t + r$. The reduction of f modulo g is then

$$f - \frac{\alpha t}{\mathrm{M}(g)} \cdot g \ , \tag{13}$$

where $\mathrm{M}(g)$ denotes the leading monomial of g. Let Q be a set of polynomials. The *reducer set* of the polynomial f with respect to Q are the polynomials g in Q with the property that the leading monomial of f can be reduced modulo g. There exists a simple algorithm for a maximal reduction of a polynomial f modulo a set Q of polynomials based on (13). Since R is not a principal ideal domain (if $n > 1$), the maximal reduction is not unique, and an element q of $\langle Q \rangle$ can be irreducible modulo Q. A Gröbner basis G (with respect to a term ordering) is defined and characterized by the following equivalent conditions:

1. Maximal reductions modulo G are unique.
2. If $f \in \langle G \rangle$, then f reduces to 0 modulo G.
3. For all f and g in G,

$$\mathrm{lcm}\,(\mathrm{M}(f), \mathrm{M}(g)) \cdot \left(\frac{f}{\mathrm{M}(f)} - \frac{g}{\mathrm{M}(g)} \right) =: \text{s-poly}(f, g)$$

reduces to 0 modulo G.

For given P, the third criterion leads to a simple algorithm for the computation of a Gröbner basis G of $\langle P \rangle$ by extending P.

Algorithm (Buchberger) *Choose any pair (f_1, f_2) in $P \times P$ and compute a maximal P-reduction of* s-poly(f_1, f_2). *If it is different from zero, extend P by this polynomial. Repeat the process for all pairs, including the pairs with components added to P during the execution of the algorithm.*

This algorithm can be improved by criteria stating whether s-poly(f, g) reduces to 0, such that the number of s-polynomial reductions is decreased. The complexity of Gröbner basis computations is a subject of ongoing research. If the system $P = 0$ has only finitely many solutions over $\mathbf{C}$, the computation of a lexicographic Gröbner basis for $\langle P \rangle$ has complexity $O(D^{n^2})$, where n is a bound for the number of variables and polynomials and D is the maximal degree. The degrees of the polynomials in the Gröbner basis are of order $O(D')$, where

$$D' := (nD)^{(n+1)^{2^{s+1}}} , \tag{14}$$

and s is the dimension of the ideal, $s \leq n$.

The following are key properties of Gröbner bases. Let P be a set of polynomials and G a monic Gröbner basis for $\langle P \rangle$ (where monic means that the coefficients of the leading monomials of all the polynomials are 1).

[3]For an introduction to Gröbner bases, see for example [16].

Property 1. $P = 0$ has a solution if and only if $1 \notin G$.

Property 2. Let H be the set of all leading terms occurring in G. Then the following statements are equivalent:

1. P has finitely many solutions over $\mathbf{C}$,
2. For all i, there exists m_i such that $(x_i)^{m_i} \in H$.

The first property is a criterion for solvability, and the second property implies, when using the lexicographic ordering, that a subset of the equations coming from the polynomials of the Gröbner basis is a system of triangular form and can be solved if univariate polynomial equations can be solved. The fact that $(x_i)^{m_i}$ is the leading term of a polynomial implies that the variables $x_1, \ldots, x_{i-1}$ do *not* occur in the polynomial. For example the polynomial with $(x_n)^{m_n}$ as leading term is *univariate* (with the only variable x_n). Analogously, there is a polynomial containing x_{n-1} and x_n only, etc.

B.2 Completing the proof of Theorem 6.

Proof that Condition 2 *is sufficient.* Let $|H| = f(p)$ for some nonconstant polynomial (with integer coefficients) $f(x)$ dividing $x^N - 1$, where $N = O(1)$.

We show first that we can assume without loss of generality that $f(x)$ equals a cyclotomic polynomial $\Phi_n(x)$ for some $n = O(1)$. The cyclotomic polynomials are the irreducible factors of the polynomials $x^N - 1$ over the ring $\mathbf{Z}$ of integers (see for example [24]). More precisely, we have

$$x^N - 1 = \prod_{d|N} \Phi_d(x) \ ,$$

and the polynomials Φ_d are irreducible over $\mathbf{Z}$. The degree of $\Phi_n(x)$ is $\varphi(N)$, where φ is Euler's totient function. Because the cyclotomic polynomials are irreducible over $\mathbf{Z}$, the (nonconstant) polynomial $f(x)$ (that divides $x^N - 1$) must be a multiple of at least one cyclotomic polynomial $\Phi_n(x)$.

We show that a subgroup H of $GF(p^n)^*$ with $|H| = \Phi_n(p)$ (for $n = O(1)$) is defined strongly (n, α)-algebraically over $GF(p)$ for some $\alpha = (\log p)^{O(1)}$. This proves the second part of Theorem 6, because a group which has a subgroup with this property has the property itself (the same EMBED algorithm can be used). Let

$$\Phi_n(x) = \sum_{j=0}^{\varphi(n)} c_j x^j$$

(with $c_j \in \mathbf{Z}$). Let further $\alpha_0, \ldots, \alpha_{n-1}$ be a normal basis of $GF(p^n)$ over $GF(p)$.

We describe the EMBED algorithm for H. Let $x, e \in GF(p)$ be given. We compute, by a polynomial number of algebraic operations in $GF(p)$, an element $\beta = (\beta_0, \ldots, \beta_{n-1})$ such that $x+e$ is one of the coordinates of β, for instance $x+e = \beta_0$. Again, we need an alternative characterization of the fact that $\beta \in H$ in terms of the $GF(p)$-coordinates of β. The following conditions are equivalent for $\beta = \sum \beta_i \alpha_i$.

$$\begin{aligned} \beta \in H &\Leftrightarrow \beta^{|H|} = 1 \\ &\Leftrightarrow \left(\sum_{i=0}^{n-1} \beta_i \alpha_i \right)^{\sum_{j=0}^{\varphi(n)} c_j p^j} = 1 \end{aligned}$$

$$\Leftrightarrow \prod_{j=0}^{\varphi(n)} \left(\left(\sum_{i=0}^{n-1} \beta_i \alpha_i \right)^{p^j} \right)^{c_j} = 1$$

$$\Leftrightarrow \prod_{j=0}^{\varphi(n)} \left(\sum_{i=0}^{n-1} \beta_i \alpha_{i+j} \right)^{c_j} = 1$$

$$\Leftrightarrow \prod_{c_j \geq 0} \left(\sum_{i=0}^{n-1} \beta_i \alpha_{i+j} \right)^{c_j} - \prod_{c_j < 0} \left(\sum_{i=0}^{n-1} \beta_i \alpha_{i+j} \right)^{-c_j} = 0 \ .$$

In the fourth step, we have made use of $\beta_i^{p^j} = \beta_i$ (because $\beta_i \in GF(p)$) and $\alpha_i^{p^j} = \alpha_{i+j}$ (by the definition of the normal basis). The last condition corresponds to a system of n polynomial equations (with $GF(p)$-coefficients) in the β_i, where the maximal degree D of the polynomials is bounded by

$$D \leq \max \left\{ \sum_{c_j > 0} c_j, \sum_{c_j < 0} |c_j| \right\} \leq \varphi(n) \cdot \max_j |c_j| \ .$$

As in the first part of the proof, the EMBED algorithm assigns $x + e$ to one of the β_i's, random values to some of the other β_i's, and solves the arising equations over $GF(p)$ for the remaining β_i's. Because $|H|/p^n \approx 1/p^{n-\varphi(n)}$, i.e., approximately every $p^{n-\varphi(n)}$-th element β of $GF(p^n)$ is also an element of H, we have to solve the equations for $n - \varphi(n)$ different coordinates β_i simultaneously in order to have an expectation of one solution. (If no solution is found, the algorithm reports failure.)

Using Gröbner bases, this system of polynomial equations can now be transformed into an equivalent system of triangular form (see section B.1). The computation of the Gröbner basis uses only algebraic operations in $GF(p)$, and its complexity is of order $O(D^{n^2})$ (see [16]). The triangular system of equations can be solved by the Cantor-Zassenhaus algorithm for solving *univariate* polynomial equations (see in the first part of the proof of Theorem 6).

According to the result of Gianni and Kalkbrener (see [16]), it suffices to solve a subset of $n - \varphi(n)$ equations. The first polynomial has to be solved once, the second one D' times (where D' is defined as in (14); the reason is that in the worst case, the first polynomial has D' different solutions), the third one $(D')^2$ times, etc. This yields $O((D')^n)$ executions of the Cantor-Zassenhaus algorithm. (The *effective* number of executions will be much smaller in a typical case, since only about one solution is expected.)

The expected complexity of the required executions of the EMBED algorithm is polynomial in $\log p$ (and the algorithm uses only algebraic operations in $GF(p)$) if $n = O(1)$. □

Appendix C: Algebraic algorithms solving the DH problem. The results described in this appendix are based on the following observation. Assume that not only a DH oracle for a group G, but also an efficient *algorithm* which solves the DH problem in an entire class of groups, such as elliptic curves over a finite field or the groups $GF(p)^*$, is given. If this algorithm additionally has the property that it uses only algebraic operations in the underlying field, then it can be executed on inputs that are not explicitly known, but only implicitly represented (in the sense of section 3). This allows to iterate the reduction algorithm described in section 3,

i.e., computing discrete logarithms in G is reduced to the same problem in a group $GF(p)^*$, which is further reduced to the DL problem of another group $GF(q)^*$, and so on.

We give an example. Assume that a uniform polynomial-time algorithm exists for solving the DH problem in all the groups $GF(p)^*$, and that these algorithms use algebraic operations in $GF(p)$ only. Let again $B = (\log |G|)^{O(1)}$ be a smoothness-bound, p_0 a prime factor of $|G|$ greater than B, and let p_1 be the only prime factor of $p_0 - 1$ greater than B. Assume further that p_i is the only prime factor of $p_{i-1} - 1$ greater than B for all $i = 2, \ldots, k$, and that $p_k - 1 =: T = \prod_i r_i^{n_i}$ is B-smooth, and $k = O(1)$. Given $a = g^s$, it is possible to compute $x_0 \equiv s \pmod{p_0}$, $x_0 \in GF(p_0)$, in polynomial time as follows when given a DH oracle for G. Let

$$h_0 := \frac{|G|}{p_0}, \quad h_i := \frac{p_{i-1} - 1}{p_i} \quad (\text{for } i = 1, \ldots, k) \ ,$$

and let

$$GF(p_i)^* = \langle c_i \rangle \quad (\text{for } i = 0, \ldots, k-1) \ .$$

If $x_0 \neq 0$, then $x_0 = c_0^{w_0}$ (in $GF(p_0)$), and g^s is an implicit representation of x_0. Since $p_0 - 1$ has a large prime factor p_1, w_0 modulo p_1 cannot be obtained directly. But $x_1 \equiv w_0 \pmod{p_1}$ (with $x_1 \in GF(p_1)$) can be written as $x_1 = c_1^{w_1}$ (in $GF(p_1)$, if $x_1 \neq 0$), and g^s is a "double-implicit" representation of x_1. Our assumptions allow efficient computation with these elements of $GF(p_1)$ which are "double-implicitly" represented. For example, an addition of two $GF(p_1)$-elements requires multiplication of the corresponding implicitly represented $GF(p_0)^*$-elements and can be obtained by a call to the DH oracle for G. A multiplication in $GF(p_1)^*$ is done by an oracle call for $GF(p_0)^*$ with implicitly represented arguments and an implicitly represented answer. This works (in polynomial time) because of the stated properties of the DH algorithm for $GF(p_0)^*$.

Analogously, computation with $(k+1)$-times implicitly represented arguments is possible in the smooth group $GF(p_k)^*$. The index-search problem for the list

$$\left(g^{h_0 c_0^{h_1 c_1^{\cdots^{h_k c_k^{\frac{T}{r_i} t}}}}} \right)_{t=0,\ldots,r_i-1}$$

and the element

$$g^{h_0 c_0^{h_1 c_1^{\cdots^{h_k c_k^{\frac{T}{r_i} w_k}}}}}$$

which can be obtained in polynomial time by computation with multiply implicitly represented arguments, is solved and leads to w_k modulo r_i. When this is done for all prime powers $r_i^{n_i}$, w_k is computable modulo T. Then $x_k = c_k^{w_k}$ (in $GF(p_k)$), and one can get w_{k-1} modulo $p_{k-1} - 1$ in polynomial time because the other prime factors of $p_{k-1} - 1$ are smaller than B. Finally, we obtain w_0 modulo $p_0 - 1$ and x_0.

Remark. The reason for assuming that $p_{i-1} - 1$ has *only one* large prime factor p_i is that otherwise it would not be possible to find the factors of $p_{i-1} - 1$ in polynomial time. When these factors are given, then the condition is unnecessary.

THEOREM 13. *Let P be a fixed polynomial. Let G be a cyclic group with the property that all prime factors p_0 of $|G|$ greater than $B := P(\log|G|)$ are single, and that for all such prime factors there exist $k = O(1)$ and primes p_i $(i = 1, \ldots, k)$ such that p_i is the only prime factor of $p_{i-1} - 1$ greater than B for $i = 1, \ldots, k$, and $p_k - 1$ is B-smooth. Assume further that a polynomial-time algorithm is given which solves the DH problem in the groups $GF(p)^*$ and uses algebraic operations in $GF(p)$ only. Then, breaking the DH protocol and computing discrete logarithms are polynomial-time equivalent in G.* □

The process works in an analogous way if some of the used groups are cyclic elliptic curves or Jacobians, provided an efficient algebraic (with respect to the underlying field $GF(p)$) DH algorithm is given for these groups.

Appendix D: Construction of groups for which a reduction of the DL problem to the DH problem is efficiently constructible. It appears desirable to use a group G in the DH protocol for which the algorithm reducing the DL problem to the DH problem can easily be found. However, such reasoning should be used with care because it is conceivable that knowledge of the auxiliary groups makes computing discrete logarithms easier. There are three possible scenarios:

1. When given G it is easy (also for the opponent) to find suitable auxiliary groups.
2. The designer of the group G knows suitable auxiliary groups but they are difficult to find for an opponent.
3. The designer of the group G knows that suitable auxiliary groups exist, without knowing them.

Note that the second case can always be transformed into the first by publishing the suitable auxiliary groups. Of course, because this information can only help an opponent in breaking the DH protocol, there is no reason for the designer of the group to make it public.

Constructing a group G of the third type is not difficult: choose a (secret) arbitrary large smooth number m and search for a prime p in the interval $[m - 2\sqrt{m} + 1, m + 2\sqrt{m} + 1]$. A group G whose order contains only such large prime factors satisfies the third property. Note that it is easy to construct, for a given n, a group G for the DH protocol whose order is a multiple of n. One possibility is to find a multiple l of n (where l/n is small) such that $l + 1$ is prime and to use $G = GF(l + 1)^*$. An alternative is to use the construction of Lay and Zimmer [22] for finding an elliptic curve of order n.

The second case is somewhat more involved. Primes p for which the designer knows an auxiliary group over $GF(p)$ can be obtained by choosing a large smooth number m and using the method of Lay and Zimmer [22] for constructing a prime p together with an elliptic curve of order m. When given such prime factors of the group order, a group G can be found as described.

We now consider efficient constructions for the first case. We generalize an algorithm, presented in [47] by Vanstone and Zuccherato, for constructing a large prime p such that either a quarter of the curves $y^2 = x^3 - Dx$ or every sixth curve of the form $y^2 = x^3 + D$ have smooth order. First, we construct primes $p = a^2 + (k \pm 1)^2$ (for a fixed k with l digits) such that $a^2 + k^2$, which is then one of the possible orders of the curves $y^2 = x^3 - Dx$ over $GF(p)$ (see (4)), is smooth.

Let l'-digit numbers x_1, x_2, y_1, and y_2 be chosen at random. Define

$$u + vi := (x_1 + y_1 i)(x_2 + y_2 i);$$

that is,

$$u = x_1x_2 - y_1y_2, \quad v = x_1y_2 + x_2y_1 \ .$$

Then u and v have at most $2l'$ digits. If $\gcd(u, v)$ divides k (otherwise choose again), one can compute numbers c and d (of at most $2l' + l$ digits) such that

$$cv + du = k \ .$$

Define

$$a := cu - dv \ ,$$

and restart the process if a is even. Then

$$a + ki = (c + di)(u + vi) = (c + di)(x_1 + y_1i)(x_2 + y_2i)$$

and

$$a^2 + k^2 = (c^2 + d^2)(x_1^2 + y_1^2)(x_2^2 + y_2^2) \ .$$

The process is repeated until a^2+k^2 is s-digit-smooth, which happens with probability approximately

$$\left(\frac{4l' + 2l}{s}\right)^{-\frac{4l'+2l}{s}} \cdot \left(\frac{2l'}{s}\right)^{-\frac{2l'}{s}} \cdot \left(\frac{2l'}{s}\right)^{-\frac{2l'}{s}}$$

(according to (1)), and smoothness can be tested with the elliptic curve factoring algorithm [23]. Because a and k are odd, exactly one of the expressions $a + (k \pm 1)i$ is congruent to 1 modulo $2 + 2i$. Let $\alpha := a + (k \pm 1)i$, respectively. Repeat the computations until

$$p := \alpha\overline{\alpha} = a^2 + (k \pm 1)^2$$

is prime. According to (4), a quarter of the curves $y^2 = x^3 - Dx$ over $GF(p)$ have smooth order $a^2 + k^2$. Hence p is an $(8l' + 2l)$-digit prime such that an elliptic curve with s-digit-smooth order is constructible over $GF(p)$. The expected number of trials is

$$O\left(\left(\frac{4l' + 2l}{s}\right)^{\frac{4l'+2l}{s}} \cdot \left(\frac{2l'}{s}\right)^{\frac{4l'}{s}} \cdot (8l' + 2l)\right) \ . \tag{15}$$

In a similar way, primes can be constructed such that curves of type $y^2 = x^3 + D$ have smooth order. More precisely, one can generate primes $p = a^2 - a(k\pm1) + (k\pm1)^2$ such that $a^2 - ak + k^2$ is one of the possible orders of the curves $y^2 = x^3 + D$ over $GF(p)$ (see (5)) and s-digit-smooth.

In case of a small k, an L-digit prime p such that an s-digit-smooth curve is constructible over $GF(p)$ can be found by

$$O\left(\left(\frac{L}{\sqrt{8} \cdot s}\right)^{\frac{L}{s}} \cdot L\right)$$

trials instead of

$$O\left(\left(\frac{L}{s}\right)^{\frac{L}{s}} \cdot L\right)$$

trials when varying p among L-digit numbers until p is prime and one of the considered curves is s-digit-smooth. For example, a 100-digit prime p such that a 10-digit-smooth curve over $GF(p)$ is constructible can be found by approximately $3 \cdot 10^6$ trials (instead of about 10^{11} trials when using the straightforward strategy).

Acknowledgment. The authors thank Dan Boneh, Dima Grigoriev, Hendrik Lenstra, Markus Metzger, Victor Shoup, and Igor Shparlinsky for interesting discussions on the subject of this paper, and two anonymous referees for their very helpful comments for improving the presentation.

REFERENCES

[1] L. M. Adleman and M. A. Huang, *Primality Testing and Abelian Varieties over Finite Fields*, Lecture Notes in Math. 1512, Springer-Verlag, New York, 1992.
[2] E. Bach and J. Shallit, *Factoring with cyclotomic polynomials*, Math. Comp., 52 (1989), pp. 201–219.
[3] D. Boneh, *Studies in computational number theory with applications to cryptography*, Ph. D. Thesis, Princeton Univ., Princeton, NJ, 1996.
[4] D. Boneh and R. J. Lipton, *Algorithms for black-box fields and their application to cryptography*, Advances in Cryptology—CRYPTO '96, Lecture Notes in Computer Science 1109, Springer-Verlag, 1996, pp. 283–297.
[5] S. Brands, *An Efficient Off-line Electronic Cash System Based on the Representation Problem*, Tech. Rep. CS-R9323, CWI, Amsterdam, 1993.
[6] J. Buchmann and V. Müller, *Computing the number of points of elliptic curves over finite fields*, Proceedings ISSAC '91, ACM Press, New York, 1991, pp. 179–182.
[7] J. Buchmann and H. C. Williams, *A key-exchange system based on imaginary quadratic fields*, J. Cryptology, 1 (1988), pp. 107–118.
[8] E. R. Canfield, P. Erdös, and C. Pomerance, *On a problem of Oppenheim concerning "Factorisatio Numerorum,"* J. Number Theory, 17 (1983), pp. 1–28.
[9] D. G. Cantor, *Computing in the Jacobian of a hyperelliptic curve*, Math. Comp., 48 (1987), pp. 95–101.
[10] M. A. Cherepnev, *On the connection between discrete logarithms and the Diffie–Hellman problem*, Discrete Math. Appl., 1996.
[11] D. Coppersmith and I. E. Shparlinsky, *On polynomial approximation and the parallel complexity of the discrete logarithm problem and breaking the Diffie–Hellman cryptosystem*, preprint, Nov. 1996.
[12] B. den Boer, *Diffie–Hellman is as strong as discrete log for certain primes*, Advances in Cryptology—CRYPTO '88, Lecture Notes in Comput. Sci. 403, Springer-Verlag, New York, 1989, pp. 530–539.
[13] W. Diffie and M. E. Hellman, *New directions in cryptography*, IEEE Trans. Inform. Theory, 22 (1976), pp. 644–654.
[14] T. El-Gamal, *A public key cryptosystem and a signature scheme based on the discrete logarithm*, IEEE Trans. Inform. Theory, 31 (1985), pp. 469–472.
[15] W. Feller, *An Introduction to Probability Theory and its Applications*, John Wiley & Sons, 1968.
[16] K. O. Geddes, S. R. Czapor, and G. Labhan, *Algorithms for Computer Algebra*, Kluwer Academic Publisher, 1992.
[17] S. Goldwasser and J. Kilian, *Almost all primes can be quickly certified*, Proceedings of the 18th Annual ACM Symposium on the Theory of Computing, 1986, pp. 316–329.
[18] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, Oxford University Press, Oxford, 1979.
[19] K. Ireland and M. Rosen, *A Classical Introduction to Modern Number Theory*, Springer-Verlag, New York, 1982.
[20] N. Koblitz, *Hyperelliptic cryptosystems*, J. Cryptology, 1 (1989), pp. 139–150.

[21] N. Koblitz, *Elliptic curve cryptosystems*, Math. Comp., 48 (1987), pp. 203–209.
[22] G.-J. Lay and H. G. Zimmer, *Constructing elliptic curves with given group order over large finite fields*, in Proceedings of ANTS-I, Lecture Notes in Comput. Sci. 877, Springer-Verlag, New York, 1994, pp. 250–263.
[23] H. W. Lenstra, Jr., *Factoring integers with elliptic curves*, Annals of Mathematics, 126 (1987), pp. 649–673.
[24] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Application*, Cambridge University Press, Cambridge, UK, 1986.
[25] J. L. Massey, *Advanced Technology Seminars Short Course Notes*, Zürich, 1993, pp. 6.66–6.68.
[26] U. M. Maurer, *Towards the equivalence of breaking the Diffie–Hellman protocol and computing discrete logarithms*, in Advances in Cryptology—CRYPTO '94, Lecture Notes in Comput. Sci. 839, Springer-Verlag, New York, 1994, pp. 271–281.
[27] U. M. Maurer and S. Wolf, *The Diffie–Hellman protocol*, in Des. Codes Cryptog., Special Issue "20 Years of Public-Key Cryptography," to appear.
[28] U. M. Maurer and S. Wolf, *Lower Bounds on Generic Algorithms in Groups*, in Advances in Cryptology—ENDOCRYPT '96, Lecture Notes in Comput. Sci. 1403, Springer-Verlag, New York, 1998, pp. 72–84.
[29] U. M. Maurer and S. Wolf, *On the complexity of breaking the Diffie–Hellman protocol*, Tech. Rep. 244, Computer Science Department, ETH Zürich, Switzerland, 1996, pp. 27–29.
[30] U. M. Maurer and S. Wolf, *Diffie–Hellman oracles*, in Advances in Cryptology—CRYPTO '96, Lecture Notes in Comput. Sci. 1109, Springer-Verlag, New York, 1996, pp. 268–282.
[31] U. M. Maurer and Y. Yacobi, *Non-interactive public-key cryptography*, Des. Codes Cryptog., 9 (1996), pp. 305–316.
[32] K. S. McCurley, *A key distribution system equivalent to factoring*, J. Cryptology, 1 (1988), pp. 95–105.
[33] K. S. McCurley, *The discrete logarithm problem*, in Cryptology and Computational Number Theory, Proc. Sympos. Appl. Math. 42, C. Pomerance, ed., American Mathematical Society, Providence, RI, 1990, pp. 49–74.
[34] A. J. Menezes, ed., *Applications of Finite Fields*, Kluwer Academic Publishers, Norwell, MA, 1992.
[35] A. J. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, Norwell, MA, 1993.
[36] V. Miller, *Uses of elliptic curves in cryptography*, in Advances in Cryptology—CRYPTO '85, Lecture Notes in Comput. Sci. 218, Springer-Verlag, New York, 1986, pp. 417–426.
[37] S. C. Pohlig and M. E. Hellman, *An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance*, IEEE Trans. Inform. Theory, 24 (1978), pp. 106–110.
[38] J. M. Pollard, *Monte-Carlo methods for index computation mod p*, Math. Comp., 32 (1978), pp. 918–924.
[39] J. M. Pollard, *Theorems on factorization and primality testing*, Proceedings of the Cambridge Philosophical Society, 76 (1974), pp. 521–528.
[40] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, 21 (1978), pp. 120–126.
[41] H. Rück, *A note on elliptic curves over finite fields*, Math. Comp., 49 (1987), pp. 301–304.
[42] K. Sakurai and H. Shizuya, *Relationships among the computational powers of breaking discrete log cryptosystems*, Advances in Cryptology—EUROCRYPT '95, Lecture Notes in Comput. Sci. 921, Springer-Verlag, New York, 1995, pp. 341–355.
[43] C. P. Schnorr, *Efficient identification and signatures for smart cards*, in Advances in Cryptology—CRYPTO '89, Lecture Notes in Comput. Sci. 435, Springer-Verlag, New York, 1990, pp. 239–252.
[44] R. Schoof, *Elliptic curves over finite fields and the computation of square roots mod p*, Math. Comp., 44 (1985), pp. 483–494.
[45] V. Shoup, *Lower bounds for discrete logarithms and related problems*, in Advances in Cryptology—EUROCRYPT '97, Lecture Notes in Comput. Sci. 1233, Springer-Verlag, New York, 1997, pp. 256–266.
[46] I. E. Shparlinsky, *Computational Problems in Finite Fields*, Kluwer Academic Publishers, Norwell, MA, 1992.
[47] S. A. Vanstone and R. J. Zuccherato, *Elliptic curve cryptosystems using curves of smooth order over the ring $\mathbf{Z}_n$*, IEEE Trans. Inform. Theory, 43 (1997), pp. 1231–1237.
[48] S. Wolf, *Diffie–Hellman and discrete logarithms*, Diploma Thesis, Department of Computer Science, ETH Zürich, Zürich, Switzerland, 1995.

[49] S. Wolf, *Unconditionally and Computationally Secure Key Agreement in Cryptography*, Ph.D. thesis, ETH Zürich, Zürich, Switzerland, 1999.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1722–1758

SELECTING THE MEDIAN*

DORIT DOR† AND URI ZWICK†

Abstract. Improving a long-standing result of Schönhage, Paterson, and Pippenger [*J. Comput. System Sci.*, 13 (1976), pp. 184–199] we show that the *median* of a set containing n elements can always be found using at most $c \cdot n$ comparisons, where $c < 2.95$.

Key words. median selection, comparison algorithms, concrete complexity

AMS subject classifications. 68Q25, 68R05, 06A07

PII. S0097539795288611

1. Introduction.

The *selection problem* is defined as follows: Given a set X containing n distinct elements drawn from a totally ordered domain, and given a number $1 \leq i \leq n$, find the ith *order statistic* of X, i.e., the element of X larger than exactly $i-1$ elements of X and smaller than the other $n-i$ elements of X. The *median* of X is the $\lceil n/2 \rceil$th order statistic of X.

The selection problem is one of the most fundamental problems of computer science and it has been extensively studied. Selection is used as a building block in the solution of other fundamental problems such as sorting and finding convex hulls. It is somewhat surprising therefore that only in the early 1970s was it shown, by Blum et al. [BFP+73], that the selection problem can be solved in $O(n)$ time. As $\Omega(n)$ time is clearly needed to solve the selection problem, the work of Blum et al. completely solves the problem. Or does it?

A very natural setting for the selection problem is the *comparison model*. An algorithm in this model can access the input elements only by performing pairwise comparisons between them. The algorithm is charged only for these comparisons; all other operations are free. The comparison model is one of the few models in which *exact* complexity results may be obtained. What then is the exact comparison complexity of finding the median?

The comparison complexity of many comparison problems is exactly known. It is clear, for example, that exactly $n-1$ comparisons are needed, in the worst case, to find the maximum or minimum of n elements. Exactly $n + \lceil \log n \rceil - 2$ comparisons are needed to find the second largest (or second smallest) element (Schreier [Sch32], Kislitsyn [Kis64]). Exactly $\lceil 3n/2 \rceil - 2$ comparisons are needed to find both the maximum and the minimum of n elements (Pohl [Poh72]). Exactly $2n-1$ comparisons are needed to merge two sorted lists each of length n (Stockmeyer and Yao [SY80]). Finally, $n \log n + O(n)$ comparisons are needed to sort n elements (e.g., Ford and Johnson [FJ59]).

A relatively large gap, considering the fundamental nature of the problem, still remains, however, between the known lower and upper bounds on the exact complexity of finding the median. After presenting a basic scheme by which an $O(n)$ selection algorithm can be obtained, Blum et al. [BFP+73] try to optimize their algorithm and

*Received by the editors July 5, 1995; accepted for publication (in revised form) May 20, 1997; published electronically May 13, 1999.
http://www.siam.org/journals/sicomp/28-5/28861.html
†Department of Computer Science, School of Mathematical Sciences, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv 69978, Israel (ddorit@math.tau.ac.il, zwick@math.tau.ac.il).

present a selection algorithm that performs at most $5.43n$ comparisons. They also obtain the first nontrivial lower bound and show that $1.5n$ comparisons are required, in the worst case, to find the median. The result of Blum et al. is subsequently improved by Schönhage, Paterson, and Pippenger [SPP76], who present a beautiful algorithm for the selection of the median, or any other element, using at most $3n + o(n)$ comparisons. In this work we improve the long-standing result of Schönhage, Paterson, and Pippenger and present a selection algorithm that uses at most $2.95n$ comparisons.

Bent and John [BJ85] (see also John [Joh88]), improving previous results of Kirkpatrick [Kir81], Munro and Poblete [MP82], and Fussenegger and Gabow [FG78], obtained a $(1 + H(\alpha))\cdot n - o(n)$ lower bound on the number of comparisons needed to select the αnth element of a set of n elements, where $H(\alpha) = \alpha \log \frac{1}{\alpha} + (1-\alpha) \log \frac{1}{1-\alpha}$ is the binary entropy function (all logarithms in this paper are taken to base 2). We have shown recently [DZ96a] (using somewhat different methods from the ones used here) that the αnth element can be selected using at most $(1 + \alpha \log \frac{1}{\alpha} + O(\alpha \log\log \frac{1}{\alpha}))\cdot n$ comparisons. This for small values of α is almost optimal. The bound of Bent and John gives in particular a $2n - o(n)$ lower bound on the number of comparisons needed to find the median. We have recently improved this lower bound slightly to $(2+\epsilon)n$ ([DZ96b]; see also [Dor95] and [DHUZ96]).

Our work slightly narrows the gap between the best known lower and upper bounds on the comparison complexity of the median problem. Although our improvement is quite modest, many new ideas were required to obtain it. These new ideas shed some more light on the intricacy of the median finding problem.

Algorithms for selecting the ith element for small values of i were obtained by Hadian and Sobel [HS69], Hyafil [Hya76], Yap [Yap76], and Ramanan and Hyafil [RH84]. See also Aigner [Aig82] and Eusterbrock [Eus93].

All the results mentioned so far deal with the number of comparisons needed in the *worst case*. Floyd and Rivest [FR75] showed that the ith element can be found using an *expected* number of $n + i + o(n)$ comparisons. Cunto and Munro [CM89] had shown that the bound of Floyd and Rivest is tight.

The central idea used by Schönhage, Paterson, and Pippenger in their $3n + o(n)$ median algorithm is the idea of *factories*. They use factories for the mass production of certain partial orders at a much reduced cost. To obtain our results we extend the notion of factories. We introduce *green* factories and perform an *amortized* analysis of their production costs. We obtain improved green factories, with which we can improve the $3n + o(n)$ result of Schönhage, Paterson, and Pippenger.

The performance of a green factory is mainly characterized by two parameters u_0 and u_1 (the *upper* and *lower* element costs). Using a green factory with parameters u_0 and u_1 we obtain an algorithm for the selection of the αnth element using at most $(u_0\alpha + u_1(1-\alpha))\cdot n + o(n)$ comparisons. To select the median, we use a factory with $u_0, u_1 \approx 2.95$. Actually, there is a trade-off between the lower and upper costs of a factory. For every $0 < \alpha \le 1/2$ we may choose a factory that minimizes $u_0\alpha + u_1(1-\alpha)$. We can select the $n/4$th element, for example, using at most $2.69n$ comparisons, by using a factory with $u_0 \approx 4$ and $u_1 \approx 2.25$. In this paper, we concentrate on factories for median selection. It is easy to verify that the algorithm described here, as the median finding algorithms of both Blum et al. and Schönhage, Paterson, and Pippenger, can be implemented in linear time in the RAM model.

The best green factories that we have explicitly constructed have lower and upper element costs $u_0, u_1 \simeq 2.942$, yielding a median selection algorithm that uses at

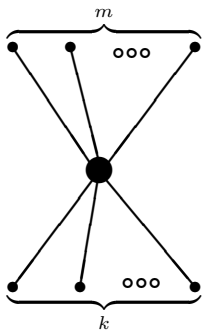


FIG. 2.1. *The partial order* S_k^m.

most $2.942n$ comparisons. These factories are extremely complicated. They employ in particular 16 different subfactories (subfactories are introduced in section 5). A full description of these factories is too long for a journal paper but may be found in [Dor95]. We describe instead a simpler, although still quite complicated, construction that uses only four different subfactories and has $u_0, u_1 \simeq 2.956$. This simpler construction depicts all the ideas used in the construction of the more complicated factories. After describing these simpler factories in full, we give a partial description of the more complicated factories.

A preliminary version of this paper appeared in [DZ95]. There we sketched the construction of a factory that uses only two subfactories and achieves $u_0, u_1 \simeq 2.968$. These are about the simplest factories with which the $3n$ median algorithm of Schönhage, Paterson, and Pippenger can be improved. They do not demonstrate, however, all the ideas required to obtain the $u_0, u_1 \simeq 2.956$ and $u_0, u_1 \simeq 2.942$ factories.

We believe that further small improvements can be obtained by building more complicated factories that use even more subfactories. It seems, however, that new ideas will be needed to obtain a more substantial improvement (see in particular the comments at the end of section 9).

In the next section we describe in more detail the concept of factory production and introduce our notion of a green factory. We also state the properties of the improved factories that we obtain. In section 3 we explain the way in which green factories are used to obtain efficient selection algorithms. The selection algorithm that we describe is a generalization of the median algorithm of [SPP76] and is similar to the selection algorithm that we describe in [DZ96a]. The subsequent sections are then devoted to the construction of our improved green factories. We end with some concluding remarks and open problems.

2. Factory production. Denote by S_k^m a partial order composed of a *center* element, m elements larger than the center and k elements smaller than the center (see Figure 2.1). An S_k^m is sometimes referred to as a *spider*. Schönhage, Paterson, and Pippenger [SPP76] show that producing l disjoint copies of S_k^m usually requires fewer comparisons than l times the number of comparisons required to produce a single S_k^m. The best way, prior to this work, for producing a single S_k^k, for example, required about $6k$ comparisons (find the median of $2k+1$ elements using the $3n+o(n)$ median algorithm). The cost per copy can be cut by almost half if the S_k^k's are mass-produced using factories.

A factory for a partial order P is a comparison algorithm with continual input and output streams. The input stream of a simple factory consists of single elements. When enough elements are fed into the factory, a new disjoint copy of P is produced. A factory is characterized by the following quantities: the *initial cost* I, which is the number of comparisons needed to initialize the factory; the *unit cost* U, which is the number of comparisons needed to generate each copy of P; and finally the *production residue* R, which is the maximum number of elements that can remain in the factory when lack of inputs stops production. For every $l \geq 0$, the cost of generating l disjoint copies of P is at most $I + l \cdot U$. Schönhage, Paterson, and Pippenger [SPP76] construct factories with the following characteristics.

THEOREM 2.1. *There is a factory F_k for S_k^k with initial cost I_k, unit cost U_k, and production residue R_k satisfying $U_k \sim 3.5k$, $I_k = O(k^2)$, $R_k = O(k^2)$.*

The notation $U_k \sim 3.5k$ here means that $U_k = 3.5k + o(k)$. Schönhage, Paterson, and Pippenger also show that if there exist factories F_k, for S_k^k's, satisfying $U_k \sim Ak$, for some $A > 0$, and $I_k, R_k = O(k^2)$, then the median of n elements can be found using at most $An + o(n)$ comparisons. The above theorem therefore immediately implies the existence of a $3.5n + o(n)$ median algorithm.

The way factories are used by selection algorithms is described in the next section. For now, we just mention that most S_k^m's generated by a factory employed by a selection algorithm are eventually broken, with either their upper elements eliminated and their lower elements returned to the factory or vice versa. While constructing an S_k^m, a factory may have compared elements that turned out to be on the same side of the center. If such elements are ever returned to the factory, the known relations among them may save the factory some of the comparisons it has to perform. To capture this, we extend the definition of factories and define green factories (factories that support the recycling of known relations). This extension is implicit in the work of [SPP76]. Making this notion explicit simplifies the analysis of our factories. The $3n + o(n)$ median algorithm of Schönhage, Paterson, and Pippenger is in fact obtained by replacing the factory F_k of Theorem 2.1 by a simple green factory.

A green factory for S_k^m's is mainly characterized by the following two quantities: the *lower element cost* u_0 and the *upper element cost* u_1. Using these quantities, the *amortized* production costs of the factory can be calculated as follows: the amortized production cost of an S_k^m whose upper m elements are eventually returned (together) to the factory is $k \cdot u_0$. The amortized production cost of an S_k^m whose lower k elements are eventually returned (together) to the factory is $m \cdot u_1$. The amortized production cost of an S_k^m such that none of its elements is returned to the factory is $k \cdot u_0 + m \cdot u_1$. Note that in this accounting scheme we attribute all the production cost to elements that are *not* returned to the factory. The initial cost I and the production residue R of a green factory are defined as before. A somewhat different definition of green factory was given by us in [DZ96a]. The definition given here uses amortized costs per *element*, whereas our previous definition used amortized costs per partial order. A green factory does not know in advance whether the lower or upper part of a generated S_k^m will be recycled. This is set by an adversary. Although not stated explicitly, the following result is implicit in [SPP76].

THEOREM 2.2. *There is a green factory G_k for S_k^k with lower and upper element costs $u_0, u_1 \sim 3$, initial cost $I_k = O(k^2)$, and production residue $R_k = O(k^2)$.*

The notation $u_0, u_1 \sim 3$ here means that $u_0, u_1 = 3 + o(1)$, where the $o(1)$ is with respect to k.

We shall see in the next section that a green factory for S_k^k with lower and upper

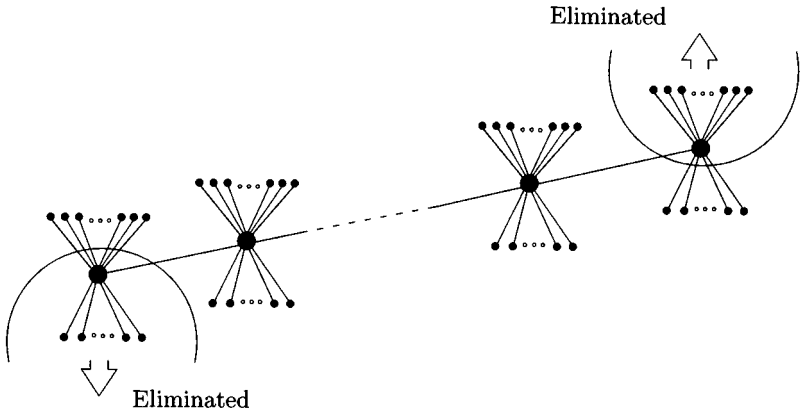


FIG. 3.1. *The ordered list of* $\tilde{\mathcal{S}}_k^k$*'s.*

element costs u_0 and u_1 yields a $(u_0+u_1)/2n+o(1)$ median algorithm. To improve the algorithm of Schönhage, Paterson, and Pippenger it is enough therefore to construct an S_k^k factory with $(u_0+u_1)/2 < 3$. Unfortunately, we are not able to construct such a factory.

However, we are able to reduce the upper and lower element costs if we allow variation among the partial orders generated by the factory. Let $\tilde{\mathcal{S}}_k^k = \{S_{k'}^{k''} : k \le k' \le 2k\ ,\ k \le k'' \le 2k\}$. We construct improved green factories that generate partial orders that are members of $\tilde{\mathcal{S}}_k^k$. These factories can be easily incorporated into the selection algorithm described in the next section. To obtain our $2.95n$ median algorithm we use green $\tilde{\mathcal{S}}_k^k$ factories $\mathcal{G}_k$ with the following characteristics.

THEOREM 2.3. *There is a green* $\tilde{\mathcal{S}}_k^k$ *factory* $\mathcal{G}_k$ *with* $u_0, u_1 \simeq 2.942$, $I_k = O(k^2)$, $R_k = O(k^2)$.

An outline of the ideas used to construct the factories $\mathcal{G}_k$ is given in section 5. The full details are then given in sections 6 through 9.

3. Selection algorithms. In this section we describe our selection algorithm. This algorithm uses an $\tilde{\mathcal{S}}_k^k$ factory. The complexity of the algorithm is completely determined by the characteristics of the factory used. This algorithm is a generalization of the median algorithm of Schönhage, Paterson, and Pippenger and is a slight variation of the selection algorithm that we describe in [DZ96a].

THEOREM 3.1. *Let* $0 < \alpha \le 1/2$. *Let* $\mathcal{F}_k$ *be an* $\tilde{\mathcal{S}}_k^k$ *factory with lower element cost* u_0, *upper element cost* u_1, *initial cost* $I_k = O(k^2)$, *and production residue* $R_k = O(k^2)$. *Then, the* αn*th smallest element, among* n *elements, can be selected using at most* $(\alpha \cdot u_0 + (1-\alpha)\cdot u_1)\cdot n + o(n)$ *comparisons.*

Proof. We refer to the αnth smallest element among the n input elements as the percentile element. The algorithm uses the factory $\mathcal{F}_k$ where $k = \lfloor n^{1/4} \rfloor$. The n input elements are fed into this factory, as singletons, and the production of partial orders $S \in \tilde{\mathcal{S}}_k^k$ commences. The centers of the generated S's are inserted, using binary insertion, into an ordered list L, as shown in Figure 3.1. When the list L is long enough we know either, as we shall soon show, that the center of the upper (i.e., last) S in L and the elements above it are too large to be the percentile element, or that the

center of the lower (i.e., first) S and the elements below it are too small to be the percentile element. Elements too large or too small to be the percentile element are eliminated. The lower elements of the upper S and the upper elements of the lower S are returned to the factory for recycling.

Let t be the current length of the list L and let r be the number of elements currently in the factory. The number of elements that have not yet been eliminated is therefore $N = \Theta(k)\cdot t + r$. Let i be the rank of the percentile element among the noneliminated elements. Initially $N = n$ and $i = \lceil \alpha n \rceil$.

The number of elements in the list known to be smaller than or equal to the center of the upper S of the list is $N_0 = \Theta(k)\cdot t$. The number of elements known to be greater than or equal to the center of the lowest S of the list is $N_1 = \Theta(k)\cdot t$. Note that $N_0 + N_1 = N + t - r$ as the centers of all the S's in the list satisfy both these criteria, the r elements currently in the factory satisfy neither, and all the other noneliminated elements satisfy exactly one of these criteria.

The algorithm consists of the following interconnected processes:

(i) Whenever sufficiently many elements are supplied to the factory $\mathcal{F}_k$, a new partial order $S \in \tilde{\mathcal{S}}_k^k$ is produced and its center is inserted into the list L using binary insertion.

(ii) Whenever $N_0 > i$, the center of the upper partial order $S \in \tilde{\mathcal{S}}_k^k$ in the list and the elements above it are eliminated, because they are too big to be the percentile element. The lower elements of S are recycled.

(iii) Whenever $N_1 > N - i + 1$, the center of the lowest partial order $S \in \tilde{\mathcal{S}}_k^k$ in the list and the elements below it are eliminated, because they are too small to be the percentile element. The upper elements of S are recycled. The value of i is updated accordingly, i.e., i is decremented by the number of elements in the lower part of S (including the center).

If (ii) and (iii) are not applicable, then $N_0 \le i$ and $N_1 \le N - i + 1$. Thus $N + t - r = N_0 + N_1 \le N + 1$ and $t - 1 \le r$. If (i) is not applicable, then by the factory definition we have $r \le R_k$. When none of (i), (ii), and (iii) can be applied we get that $t - 1 \le r \le R_k = O(k^2)$. At this stage $N = O(k^3)$, which is $O(n^{3/4})$, and the ith element among the surviving elements is found using any linear selection algorithm.

We now analyze the comparison complexity of the algorithm. Whenever (ii) is performed, the upper partial order $S \in \tilde{\mathcal{S}}_k^k$ of the list is broken. Its center and upper elements are eliminated and its lower elements are returned to the factory. The amortized production cost of the partial order S is at most u_1 comparisons per each element above the center.

Whenever (iii) is performed, the lowest partial order $S \in \tilde{\mathcal{S}}_k^k$ of the list is broken. Its center and lower elements are eliminated and its upper elements are returned to the factory. The amortized production cost of the partial order S is at most u_0 comparisons per each element below the center.

The algorithm can eliminate at most $(1-\alpha)n$ elements larger than the percentile element and at most αn elements smaller than the percentile element. The total production cost of all partial orders $S \in \tilde{\mathcal{S}}_k^k$ that are eventually broken is therefore at most $(\alpha u_0 + (1-\alpha)u_1)\cdot n + o(n)$. At most $O(k^2)$ generated partial orders $S \in \tilde{\mathcal{S}}_k^k$ are not broken. Their total production cost is $O(k^3)$. The initial production cost is $O(k^2)$. The total number of comparisons performed by the factory is therefore $(\alpha u_0 + (1-\alpha)u_1)\cdot n + o(n)$.

Let t^* be the final length of the list L (when none of (i), (ii), and (iii) is applicable). The total number of partial orders generated by $\mathcal{F}_k$ is at most $n/k + t^*$, as at least k elements are eliminated whenever a partial order is removed from L. The total cost of the binary insertions into the list L is at most $O((n/k + t^*) \cdot \log n) = O((n/k + k^2) \log n)$ which is $o(n)$. The total number of comparisons performed by the algorithm is therefore at most $(\alpha u_0 + (1 - \alpha)u_1) \cdot n + o(n)$, as required. □

Using the factories of Theorem 2.3, we obtain our main result, as follows.

THEOREM 3.2. *Any element, among n elements, can be selected using at most $2.942n + o(n)$ comparisons.*

4. Basic principles of factory design. In this section we give some of the basic principles used to construct efficient factories. The section is divided into three subsections. In the first subsection we remind the reader what *hyperpairs* are and what their *pruning cost* is. In the second subsection we describe the notion of *grafting*. In the third subsection we sketch the construction of the S_k^k factories of Schönhage, Paterson, and Pippenger [SPP76]. These factories are briefly described to illustrate the basic design principles. All the results of this section, except for Theorems 4.5 and 4.6, which are new, are essentially taken from [SPP76]. Some of the proofs are therefore omitted.

Before going into details, we describe a clever accounting principle introduced by Schönhage, Paterson, and Pippenger to simplify the complexity analysis of factories. The information we care to remember on the elements that pass through the factory can always be described using a Hasse diagram. Each comparison made by the algorithm adds an edge to the diagram and possibly deletes some edges that become redundant. At some stages we may decide to "forget" the result of some comparisons, and the edges that correspond to them are removed from the diagram. Schönhage, Paterson, and Pippenger noticed that instead of counting the number of comparisons made, we can count the number of edges cut! To this we should add the number of edges in the eliminated parts of the partial orders as well as the edges that remain in the factory when the production stops. The second number, in our factories, is at most a constant times the production residue of the factory and it can be attributed to the initial cost.

4.1. Hyperpairs. A factory usually starts the production of a partial order from $\tilde{\mathcal{S}}_k^k$ by producing a large partial order, a hyperpair, that contains a partial order from $\tilde{\mathcal{S}}_k^k$.

DEFINITION 4.1. *A hyperpair P_w, where w is a binary string, is a finite partial order with a distinguished element, the center, defined recursively by* (i) *P_λ is a single element (λ here stands for the empty string)*; (ii) *P_{w1} is obtained from two disjoint P_w's by comparing their centers and taking the larger as the new center. P_{w0} is obtained in the same way but taking the smaller of the two centers as the new center.*

The Hasse diagrams of some small hyperpairs are shown in Figure 4.1 (the meaning of the notation H_r will become clear later). Some basic properties of hyperpairs are given in the following lemma.

LEMMA 4.2. *Let c be the center of a hyperpair P_w. Let w_i be the prefix of w of length i. Let h_0 be the number of 0's in w and h_1 be the number of 1's in w. Then*

(1) *the center c together with the elements greater than it form a $P_{0^{h_0}}$ with center c. The elements greater than c form a disjoint set of hyperpairs P_λ, P_0, ..., $P_{0^{h_0-1}}$. The center c together with the elements smaller than it form a $P_{1^{h_1}}$ with center c. The elements smaller than c form a disjoint set of hyperpairs P_λ, P_1, ..., $P_{1^{h_1-1}}$.*

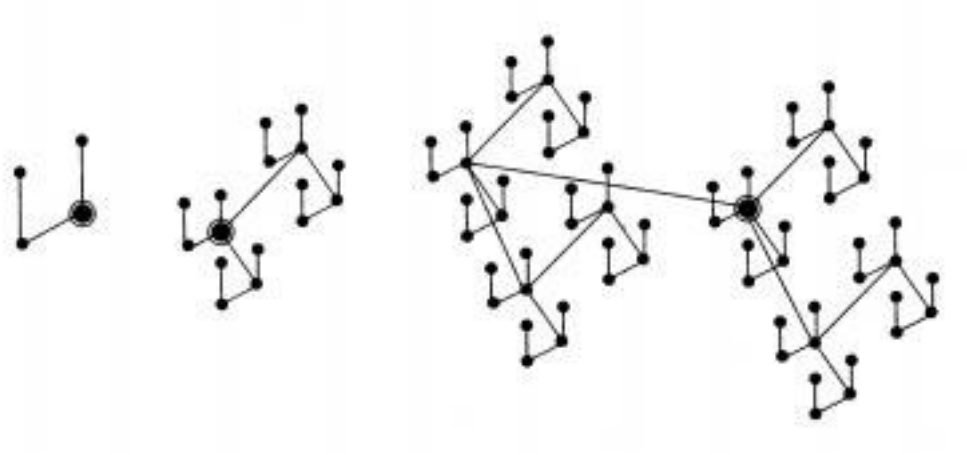


FIG. 4.1. *Some small hyperpairs* ($H_2 = P_{01}$, $H_4 = P_{0110}$, *and* $H_6 = P_{011010}$).

(2) *the hyperpair* P_w *can be parsed into its center* c *and into a disjoint set* $\{P_{w_i} \mid 0 \leq i < |w|\}$ *of smaller hyperpairs. Moreover, the center of* P_{w_i} *is above* c *if* w_{i+1} *ends with* 0 *and below* c *if* w_{i+1} *ends with* 1.

The lemma can be easily proved by induction. Note, in particular, that if $m < 2^{h_0}$ and $k < 2^{h_1}$, then P_w contains an S_k^m. No edges are cut during the construction of hyperpairs. But, before outputting an S_k^m contained in a hyperpair, all the edges connecting the elements of this S_k^m with elements not contained in this S_k^m have to be cut. This rather costly operation is referred to as *pruning*.

The *downward pruning* of a hyperpair P_w is the operation of removing from the hyperpair all the elements that are not known to be smaller or equal to its center c. The *downward pruning cost* $PR_0(w)$ of a hyperpair P_w is the cost of this operation. This cost is equal to the number of edges of the hyperpair that connect the center of P_w and the elements below it to the other elements of P_w. The *upward pruning* and the *upward pruning cost* $PR_1(w)$ of a hyperpair P_w are defined analogously. Let w be a binary string and let h_0 and h_1 be the number of 0's and 1's in it. The pruning costs can be computed using the following recursive relations.

LEMMA 4.3.

(i) $PR_0(\lambda) = 0$, $\quad PR_0(w0) = PR_0(w) + 1$, $\quad PR_0(w1) = 2PR_0(w)$.

(ii) $PR_1(\lambda) = 0$, $\quad PR_1(w0) = 2PR_1(w)$, $\quad PR_1(w1) = PR_1(w) + 1$.

Proof. A P_{w0} is composed of two P_w's and an edge connecting their two centers. If c_1 and c_2 are the centers of these P_w's and $c_1 < c_2$, then c_1 is the center of the hyperpair P_{w0}. To downward prune the hyperpair P_{w0}, all we have to do is downward prune the hyperpair P_w whose center is c_1 and then cut the edge between c_1 and c_2. The cost of this operation is therefore $PR_0(w) + 1$. The other cases are proved similarly. □

We also define the amortized, per element, pruning costs of a hyperpair P_w. Let h_0 and h_1 be the number of 0's and 1's in w and let $h = h_0 + h_1$. We define $pr_0(w) = PR_0(w)/2^{h_1}$ and $pr_1(w) = PR_1(w)/2^{h_0}$ to be the *lower element pruning cost* and the *upper element pruning cost* of w. An immediate consequence of Lemma 4.3 is the following lemma, which is easily proved by induction.

LEMMA 4.4. *Let* $w = \gamma_1\gamma_2 \ldots \gamma_h$ *be a binary string, let* w_i *be the prefix of* w *of length* i, *and let* $h_0(w_i)$ *and* $h_1(w_i)$ *be the number of* 0*'s and* 1*'s, respectively, in* w_i.

Then,

$$pr_0(w) = \sum_{i|\gamma_i=0} 2^{-h_1(w_i)}, \qquad pr_1(w) = \sum_{i|\gamma_i=1} 2^{-h_0(w_i)}.$$

Note, in particular, that if w_i is a prefix of w, then $pr_0(w_i) \leq pr_0(w)$ and $pr_1(w_i) \leq pr_1(w)$.

Usually, especially if grafting processes are also used, we do not want to prune all the elements above or below the center c of a hyperpair P_w. We next estimate the cost of pruning k elements below or above the center of a hyperpair P_w, i.e., the cost of pruning the hyperpair P_w so that all that remains of it is its center and k elements either above or below the center. We show that the cost of pruning k elements below the center of P_w is at most $k \cdot pr_0(w) + h$ while the cost of pruning k elements above the center of P_w is at most $k \cdot pr_1(w) + h$, where h is the length of w. Note that h is also the number of edges connected to the center c of the hyperpair P_w. The h terms in the above estimates will usually be negligible compared to the other terms.

By Lemma 4.2, a hyperpair P_w with center c can be parsed into h_0 hyperpairs $\{P_i^1 \mid 0 \leq i \leq h_0 - 1\}$ whose centers are above c and h_1 hyperpairs $\{P_i^0 \mid 0 \leq i \leq h_1 - 1\}$ whose centers are below c. It is easy to check that the number of elements in P_i^1 which are above the center of this P_i^1, and the number of elements in P_i^0 which are below the center of this P_i^0, are both 2^i (this follows from the fact that the string corresponding to the hyperpair P_i^1 contains exactly i 0's and the string corresponding to the hyperpair P_i^0 contains exactly i 1's). To upward prune $k = k_0 2^0 + k_1 2^1 + \cdots + k_{h-1} 2^{h-1}$ elements from P_w, we upward prune all P_i^1's for which $k_i = 1$ and cut the edges connecting c with the centers of all the other hyperpairs. As the string corresponding to P_i^1 is a prefix of w, the cost of upward pruning P_i^1 is at most $2^i \cdot pr_1(w)$. The cost of cutting the other edges is at most h. Thus, the cost of pruning k elements above the center c is at most $\sum_{i|k_i=1} 2^i \cdot pr_1(w) + h = k \cdot pr_1(w) + h$. All the "wastes" of this process are hyperpairs whose strings are prefixes of w and they can be used therefore for the construction of the next P_w. The cost of pruning k elements below the center is estimated in the same way.

To produce partial orders from $\tilde{\mathcal{S}}_k^k$ for larger and larger values of k, we have to construct larger and larger hyperpairs. When we design a family $\{\mathcal{F}_k\}_{k=1}^{\infty}$ of factories, we usually choose a (semi-) infinite binary string $\mathcal{W}$ and in each member $\mathcal{F}_k$ of this family we construct a hyperpair whose string is a long enough prefix of $\mathcal{W}$. Let w_i be the finite prefix of $\mathcal{W}$ of length i. The lower and upper element pruning costs of an infinite string $\mathcal{W}$ are defined as the limits $pr_0(\mathcal{W}) = \lim_{i\to\infty} pr_0(w_i)$ and $pr_1(\mathcal{W}) = \lim_{i\to\infty} pr_1(w_i)$. We will see in a minute that these limits do exist (they may be $+\infty$).

Let $\mathcal{W} = \gamma_1\gamma_2\ldots$ be an infinite binary string. As before, let $w_i = \gamma_1 \ldots \gamma_i$ be the prefix of $\mathcal{W}$ of length i, and let $h_0(w_i)$ and $h_1(w_i)$ be the number of 0's and 1's in w_i, respectively. It is easy to check that Lemma 4.4 holds also for infinite strings.

Schönhage, Paterson, and Pippenger base their factories on the infinite string $\mathcal{W} = 01(10)^\omega$ for which, as can be easily verified, $pr_0(\mathcal{W}) = pr_1(\mathcal{W}) = 1.5$. (Here and in what follows, we let x^ω denote the infinite string obtained by concatenating an infinite number of copies of the string x, and we let X^ω denote the set of infinite strings obtained by concatenating an infinite number of strings from the set X.) In our factories, we also need hyperpairs with cheaper lower element pruning cost and, alas, more expensive upper element pruning cost, or vice versa. For an infinite binary string $\mathcal{W}$, we let $pr(\mathcal{W}) = pr_0(\mathcal{W}) + pr_1(\mathcal{W})$. The next theorem shows that for every string $\mathcal{W}$ we have $pr(\mathcal{W}) \geq 3$. Theorem 4.6 then shows that for any real number $1 \leq a \leq 2$, there exists an infinite string $\mathcal{W}$ for which $pr_0(\mathcal{W}) = a$ and $pr_1(\mathcal{W}) = 3 - a$.

THEOREM 4.5. *For any $\mathcal{W} \in \{0,1\}^\omega$ we have $pr(\mathcal{W}) \geq 3$ with equality holding if and only if $\mathcal{W} \in \{01,10\}^\omega$.*

Proof. Let $\mathcal{W}$ be an infinite string. As before, we let w_i stand for the prefix of $\mathcal{W}$ of length i. If $\mathcal{W}$ contains only a finite number of 0's or a finite number of 1's, then $pr(\mathcal{W}) = +\infty$. Assume therefore that $\mathcal{W}$ contains an infinite number of 0's and an infinite number of 1's. Using Lemma 4.4 we get that

$$pr(\mathcal{W}) = \sum_{i|\gamma_i=0} 2^{-h_1(w_i)} + \sum_{i|\gamma_i=1} 2^{-h_0(w_i)} = \sum_{i=1}^{\infty} 2^{-h_1(w_i)} + \sum_{i=1}^{\infty} 2^{-h_0(w_i)} - 2,$$

since $\sum_{i|\gamma_i=1} 2^{-h_1(w_i)} = \sum_{i|\gamma_i=0} 2^{-h_0(w_i)} = 1$. Let x_i be the index of the ith 0 in $\mathcal{W}$ and let y_i be the index of the ith 1 in $\mathcal{W}$. Also let $x_0 = y_0 = 1$. It is easy to check that

$$\sum_{i=1}^{\infty} 2^{-h_0(w_i)} = \sum_{i=0}^{\infty} 2^{-i}(x_{i+1} - x_i) = \sum_{i=1}^{\infty} 2^{-i} x_i - 1.$$

A similar relation holds, of course, for $\sum_{i=1}^{\infty} 2^{-h_1(w_i)}$. As a consequence we get that

$$pr(\mathcal{W}) = \sum_{i=1}^{\infty} 2^{-i} x_i + \sum_{i=1}^{\infty} 2^{-i} y_i - 4.$$

This expression is clearly minimized when for every $1 \leq i < j$ we have $x_i, y_i \leq x_j, y_j$. Because all the elements in the sequences $\{x_i\}$ and $\{y_i\}$ are integral and distinct, we get that the minimum is attained when for every $i \geq 1$, either $x_i = 2i-1$ and $y_i = 2i$ or vice versa. This corresponds to strings from $\{01,10\}^\omega$. It is easy to check that for a string $\mathcal{W} \in \{01,10\}^\omega$ we have $pr(\mathcal{W}) = 3$. □

THEOREM 4.6. *For any real number $1 \leq a \leq 2$ there exists a binary sequence $\mathcal{W} \in \{01,10\}^\omega$ for which $pr_0(\mathcal{W}) = a$ and $pr_1(\mathcal{W}) = 3 - a$.*

Proof. Let $0.\alpha_1\alpha_2\ldots$ be the binary representation of $a-1$. Let $\mathcal{W} = \overline{\alpha}_1\alpha_1\overline{\alpha}_2\alpha_2\ldots$, where $\overline{\alpha}$ is the complement of the bit α. Using Lemma 4.4, we get that

$$pr_0(\mathcal{W}) = \sum_{i=1}^{\infty} 2^{-(i-\alpha_i)} = \sum_{i=1}^{\infty} 2^{-i} + \sum_{i|\alpha_i=1} 2^{-i} = 1 + (a-1) = a.$$

Since $\mathcal{W} \in \{01,10\}^\omega$, using the previous theorem we get that $pr_1(\mathcal{W}) = 3 - a$. □

Note, as an example, that if $a = 1.5$, then $a - 1$ is either $0.1000\ldots$ or $0.0111\ldots$ and both $\mathcal{W} = 01(10)^\omega$ and $\mathcal{W} = 10(01)^\omega$ satisfy $pr_0(\mathcal{W}) = pr_1(\mathcal{W}) = 1.5$.

We are already in a position to describe a simple but complete S_k^k factory. Select a string $\mathcal{W}$. Construct a hyperpair P_w that contains the partial order S_k^k, where w is a long enough prefix of $\mathcal{W}$. Prune k elements above and k elements below the center of this P_w. These $2k+1$ elements form a copy of S_k^k. By Lemma 4.2(ii), the remaining elements of P_w form a disjoint collection of partial orders each of the form P_{w_i}, where w_i is some prefix of w. These partial orders are used to construct a new copy of P_w that will be used to construct the next S_k^k. Before we output an S_k^k, we cut the $2k$ edges it contains. When some part of an S_k^k generated by the factory is recycled, the elements returned to the factory (as singletons) are used again for the construction of hyperpairs. It is easy to check that the lower and upper element costs of this simple factory are both $u_0, u_1 \sim pr_0(\mathcal{W}) + pr_1(\mathcal{W}) + 2 = pr(\mathcal{W}) + 2$. For any $\mathcal{W} \in \{01,10\}^\omega$ we get that the lower and upper element costs are $u_0, u_1 \sim 5$.

The above factory can be turned into a green factory by noticing that the elements can be returned to the factory as pairs and not as singletons. The edges of a generated S_k^k are not cut when this S_k^k leaves the factory. Instead, we wait until the elements on one of the sides of this S_k^k are eliminated by the selection algorithm. We then cut the k edges connecting the eliminated elements to the center. By cutting about $k/2$ edges we can break the k noneliminated elements into about $k/2$ pairs, which we then return to the S_k^k factory. These pairs can be used for the construction of the next large hyperpair. The lower and upper element costs of this simple green factory are $u_0, u_1 \sim pr(\mathcal{W}) + 1.5$. For any $\mathcal{W} \in \{01, 10\}^\omega$ we get $u_0, u_1 \sim 4.5$. A slightly more careful consideration shows that the elements of one of the sides can actually be recycled as quartets (P_{00}'s or P_{11}'s), thus cutting only $k/4$ edges. This gives us $u_0 \sim 4.25$ and $u_1 \sim 4.5$ or vice versa.

4.2. Grafting. The costs of the simple factories described above can be significantly improved using *grafting*. We can cheaply find elements that are smaller than the center, or elements that are larger than the center, but usually not both. The process of finding such elements is called grafting. Pruning is then used to obtain elements on the opposite side.

We demonstrate this notion using a simple example, the grafting of singletons. Take an element x, not contained in the hyperpair, and compare it to the center c of the hyperpair. Continue is this way, comparing new elements to the center, until either k elements above the center or k elements below the center are found. Note that no edges are cut in this process. All the grafted elements are put in the output partial order. The pruning process is then used to complete the partial order into an S_k^k. Adding this process to our simple factory for S_k^k, the upper and lower element costs are reduced to $u_1, u_0 \sim \max\{pr_0(\mathcal{W}), pr_1(\mathcal{W})\} + 2$ (note that now we have to prune elements from at most one side). Thus $u_0, u_1 \sim 3.5$ if we take $\mathcal{W} = 01(10)^\omega$ or $\mathcal{W} = 10(01)^\omega$. This supplies a proof to Theorem 2.1. Note that the obtained factory is a degenerate green factory since no relations are recycled. At least one side of each generated S_k^k is composed of singletons, and if this side is recycled, no comparisons can be reused.

4.3. The factories of Schönhage, Paterson, and Pippenger. We now sketch the operation of the green factories G_k obtained by Schönhage, Paterson, and Pippenger [SPP76]. These factories improve upon the simple factories described above by grafting and recycling pairs. They are described here using a new terminology that we also use in the next sections to describe our improved factories.

The factories of Schönhage, Paterson, and Pippenger use two simple pair grafting processes which we refer to as the P_0 and P_1 grafting processes. The two processes are mirror images of each other.

We start with the description of the P_0 grafting process. Let $x < y$ be a pair. Compare x with c, the center of the hyperpair. If $c < x$, then stop. Otherwise, compare y with c. The three possible outcomes of this process are (i) $c < x < y$, (ii) $x < y < c$, and (iii) $x < c < y$. These three outcomes are shown on the left of Figure 4.2 and are denoted, respectively, by X_0^2, X_2^0, and X_1^1. Note that in the second case, the result $x < c$ of the first comparison becomes redundant and the edge corresponding to it, shown dashed in the figure, is cut. Similarly, in the third case, the relation $x < y$ becomes redundant and the corresponding edge is cut.

The P_1 grafting process is, as mentioned, the mirror image of the P_0 grafting process. Let $x < y$ be a pair. Compare y with c and if $c < y$, also compare x with c. The three possible outcomes of this process, denoted by Y_2^0, Y_0^2, and Y_1^1, are shown

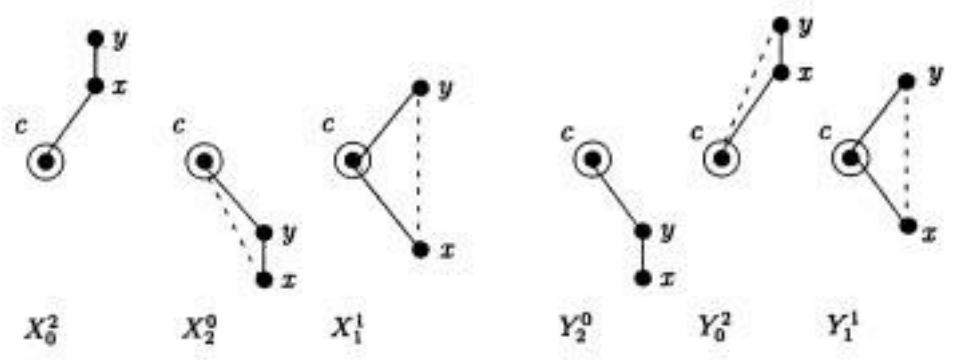


FIG. 4.2. *Possible outcomes of the simple pair grafting processes.*

TABLE 4.1
The costs of the possible outcomes of the simple pair grafting processes.

Outcome	Above	Below	*gen*	rec_0	rec_1
X_0^2	P_0	—	0	2	1
X_2^0	—	P_0	1	1	2
X_1^1	P_λ	P_λ	1	2	2
Outcome	Above	Below	*gen*	rec_0	rec_1
Y_2^0	—	P_0	0	1	2
Y_0^2	P_0	—	1	2	1
Y_1^1	P_λ	P_λ	1	2	2

on the right of Figure 4.2. Note that the outcome of Y_1^1 of the P_1 grafting process is identical to the outcome X_1^1 of the P_0 grafting process. The outcomes Y_2^0 and X_2^0, and Y_0^2 and X_0^2 have the same forms but different costs are associated with them. No edges are cut while producing X_0^2 and Y_2^0, while a single edge is cut while producing X_2^0 and Y_0^2.

The costs associated with each one of the outcomes obtained by these processes are summarized in Table 4.1. If X is an outcome of a grafting process, then $gen(X)$ is the number of edges cut during the generation of X, $rec_0(X)$ is the number of edges cut when the lower part of the spider to which the outcome X belongs is recycled while the upper part is eliminated, and $rec_1(X)$ is defined analogously as the number of edges cut when the upper part of X is recycled and the lower part eliminated.

The factory G_k starts by producing hyperpairs corresponding to prefixes of the string $\mathcal{W} = 01(10)^\omega$ (the string $\mathcal{W} = 10(01)^\omega$ could also be used). Let w_i be the prefix of $\mathcal{W}$ of length i and let $H_i = P_{w_i}$. Some small H_i's are shown in Figure 4.1. By Lemma 4.2, an H_{2r}, where $r = \lceil \log(k+1) \rceil$, contains an S_k^k. After constructing an H_{2r}, the factory initiates the pair grafting processes. The algorithm keeps two counters n_0 and n_1 of the number of grafted elements already obtained above and below the center c of the hyperpair H_{2r}. If $n_0 > n_1$, then the P_1 grafting process is applied. If $n_0 \leq n_1$, then the P_0 grafting process is applied. The grafting continues until either $n_0 \geq k$ or $n_1 \geq k$. All the elements on at least one side of the spider are thus obtained using grafting. Missing elements on the opposite side are then obtained using pruning. The operation of the algorithm is summarized in Figure 4.3.

The intuition behind this algorithm is simple. If we already have many grafted

```
Construct a hyperpair H_{2r}, where r = ⌈log(k+1)⌉.
Initialize n_0, n_1 ← 0.
while n_0 < k and n_1 < k do
{
      if n_0 > n_1 then
      {
         Apply the P_1 grafting process on a pair x < y;
         if x < y < c then n_0 ← n_0 + 2 ;  /* Y_2^0 */
         if c < x < y then n_1 ← n_1 + 2 ;  /* Y_0^2 */
         if x < c < y then n_0 ← n_0 + 1 ; n_1 ← n_1 + 1 ;  /* Y_1^1 */
      }
      else
      {
         Apply the P_0 grafting process on a pair x < y;
         if c < x < y then n_1 ← n_1 + 2 ;  /* X_0^2 */
         if x < y < c then n_0 ← n_0 + 2 ;  /* X_2^0 */
         if x < c < y then n_0 ← n_0 + 1 ; n_1 ← n_1 + 1 ;  /* X_1^1 */
      }
}

if n_0 < k then prune k − n_0 elements below the center c of H_{2r}.
if n_1 < k then prune k − n_1 elements above the center c of H_{2r}.
```

FIG. 4.3. *The factory of Schönhage, Paterson, and Pippenger.*

elements below the center, then by comparing c with the larger element y of a pair $x < y$, we force the adversary to select one of the following two options: (1) give us another pair below the center; (2) give us at least one additional element above the center. In both cases we stand to gain. In the first case, the elements below the center are organized in pairs, not in singletons, and their recycling cost is therefore reduced. In the second case, we get relatively cheap elements above the center, in addition to cheap elements that were already obtained below the center.

The elements above the center of the generated S_k^k form a collection of disjoint P_{0^i}'s and the elements below the center form a collection of disjoint P_{1^i}'s. When the lower or upper part of an S_k^k is returned to the factory, some of the existing relations among the elements returned are utilized. The amortized analysis of the green factory G_k encompasses a trade-off between the cost of generating an S_k^k and the utility obtained from its lower or upper parts when these parts are recycled. Although the S_k^k's generated by the factory of Schönhage, Paterson, and Pippenger may contain P_{0^i}'s and P_{1^i}'s, where $i > 1$, their factory is capable only of utilizing pairwise disjoint relations among the elements returned to it (as their grafting processes can use only pairs). If a P_{0^i} or a P_{1^i}, with $i > 1$, is returned to the factory, it is immediately broken into 2^{i-1} P_0's or P_1's. Note that both P_0 and P_1 simply stand for a pair of elements. It can be checked (see [SPP76]) that the upper and lower element costs of this factory are $u_1, u_0 \sim 3$. This is the best factory obtained by Schönhage, Paterson, and Pippenger.

5. Advanced principles of factory design. In this section, we outline the principles used to construct our improved factories. The first of these principles was already mentioned.

- Allowing variations in the produced partial orders.
  Our factories construct partial orders from $\tilde{\mathcal{S}}_k^k$. The exact proportion between the number of elements below and above the center of a generated partial order is not fixed in advance.
- Recycling larger relations.
  The factories of Schönhage, Paterson, and Pippenger are capable only of recycling singletons and pairs (i.e., P_0's, P_1's, and P_λ's). Our factories recycle larger constructs such as quartets (P_{00}'s and P_{11}'s), octets (P_{000}'s and P_{111}'s), 16-tuples (P_{0000}'s and P_{1111}'s), as well as pairs, singletons, and other structures (e.g., I_3's, sorted lists of three elements) which are not hyperpairs (see Figure 9.1). The nonhyperpair constructs are obtained by the more sophisticated grafting processes used.

  As pairs can be used both as P_0's and P_1's, they can be used to construct a hyperpair P_w, no matter what the string w is. This is one of the reasons why recycling pairs is easier than recycling larger hyperpairs. Quartets such as P_{00}'s (or their equivalent P_{01}'s), for example, can be used to construct a hyperpair P_w only if w starts with 0. More limitations are imposed when larger hyperpairs are considered. Nonhyperpairs cannot be used directly for the construction of hyperpairs.

  To enable the recycling of larger relations, we must be able to use them for the construction of hyperpair-like relations. We should also be able to use them for grafting.
- Constructing hyper-products.
  As mentioned, our factories may receive partial orders that could not be used for the construction of hyperpairs. These partial orders are used instead for the construction of *hyper-products*. A hyper-product $P_w \circ I$, where I is some partial order with a distinguished element, which is again called a center, is a hyperpair P_w in which each of its elements is also the center of a disjoint copy of the partial order I. Hyperpairs are, of course, special cases of hyper-products as $P_w \circ P_0 = P_{0w}$ and $P_w \circ P_1 = P_{1w}$.

  Some of the partial orders I used for hyper-products construction are unbalanced. To counter this, they are composed with oppositely unbalanced hyperpairs.
- Grafting larger relations and mass-grafting.
  The factories of Schönhage, Paterson, and Pippenger use a simple pair of pair grafting processes. We use more complicated grafting processes, even for grafting pairs. For each input construct we have several different grafting processes. Some of our grafting processes use the technique of mass production.

  Our factories apply each grafting process a certain number of times and record the number of times each outcome is obtained. From time to time, they decide to place a combination of outcomes, that tend to balance each other nicely, in the output partial order. The factory algorithms ensure that for any adversary strategy, if each grafting process is applied a certain number of times, then at least one such balanced combination can be put in output partial order. The output combinations are chosen to have low "local" lower and upper unit costs. The maxima of all these local upper and lower element

costs are the overall upper and lower element costs of the factory.

- Using subfactories.
  The factories of Schönhage, Paterson, and Pippenger generate only a single family of hyperpairs (corresponding to $\mathcal{W} = 01(10)^\omega$). Our factories generate several types of hyperpairs and hyper-products, as mentioned above. The construction of each one of these hyper-products is carried out in a separate subproduction unit that we refer to as a subfactory. Different subfactories also differ in the "raw materials" that they can process.
- Using credits in an amortized complexity analysis.
  The last principle is an accounting principle. The different constructs recycled by our factories are of different quality. Some of them, like pairs and I_3's (sorted lists of three elements), can be used very efficiently for the construction of partial orders from $\tilde{\mathcal{S}}_k^k$. Others, like P_{000}'s and P_{0000}'s, are not so appropriate for this process as they are extremely unbalanced, and using them as raw materials for the construction of partial orders from $\tilde{\mathcal{S}}_k^k$ results in a much higher production cost. To equalize these costs, each construct used by our factories is assigned a *credit*, which may be either positive or negative. The credit assigned to a construct Q is denoted by $credit(Q)$. When a construct Q is recycled, extra $credit(Q)$ comparisons are charged to the partial order from $\tilde{\mathcal{S}}_k^k$ that is being broken. These $credit(Q)$ comparisons can then be deducted from the cost of the partial order from $\tilde{\mathcal{S}}_k^k$ that will be constructed using this construct Q. High-quality materials, such as I_3's, carry negative credits. The credits attached to singletons (P_λ's) must be zero since the singletons initially fed into the factory carry no credit.

In the next section we describe the general framework of our improved factories. This framework combines the principles put forth in this and in the previous sections. We also give a general description of the complexity analysis.

6. The framework of the improved factories. A factory $\mathcal{G}$ of the type we are using is composed of several subfactories $g_1, \ldots, g_r$. These subfactories are activated in "parallel." Each subfactory is either working on the construction of a partial order from $\tilde{\mathcal{S}}_k^k$ or waiting for additional raw materials. Whenever the construction of a partial order from $\tilde{\mathcal{S}}_k^k$ in one of subfactories is finished, this partial order is output and the subfactory begins to work on the construction of a new partial order from $\tilde{\mathcal{S}}_k^k$. While designing such a factory we have to make sure that if enough materials are fed into the whole factory, then at least one of its subfactories can make progress in constructing the next partial order from $\tilde{\mathcal{S}}_k^k$. The production residue of the factory $\mathcal{G}$ is the sum of the production residues of the subfactories $g_1, \ldots, g_r$.

The operation of each subfactory is essentially independent of the operation of the other subfactories. Each subfactory processes a specific type (or in some cases, specific types) of input constructs. There may be, for example, a P_{00}-processing subfactory, an I_3-processing subfactory, and so on. A construct recycled to the factory $\mathcal{G}$ is immediately fed to an appropriate subfactory. A Q-processing subfactory (a Q subfactory for short) may sometimes produce, as by-products, partial orders which are not Q's. These partial orders are immediately fed into subfactories that can consume them.

The upper and lower element costs of $\mathcal{G}$ are the maxima of the (amortized) upper and lower element costs of $g_1, \ldots, g_r$. The credits attached to the different constructs are used to equalize the costs of the different subfactories, thereby reducing their maximum. The credits selected optimize a natural trade-off between the generation cost

of an output partial order that contains a partial order Q and the cost of utilizing Q when it is recycled.

The factory $\mathcal{G}$ can be viewed as the "union" of the subfactories $g_1, \ldots, g_r$. In the next subsection we describe the structure of a generic subfactory. In subsection 6.2 we then describe the cost analysis of a generic subfactory.

6.1. The structure of a generic subfactory. Each subfactory g is composed of a hyper-product generation process, a hyper-product pruning process, a collection $A_1, A_2, \ldots, A_\ell$ of grafting processes, and a list $C_1, C_2, \ldots, C_m$ of output combinations.

The operation of each subfactory, like the simple factories described in section 4, is composed of three main phases. In the first phase a large hyper-product is constructed using the hyper-product generation process. In the second stage the grafting processes $A_1, A_2, \ldots, A_\ell$ are activated. Whenever a combination of outcomes from the list $C_1, C_2, \ldots, C_m$ is encountered, this combination is placed in the output partial order. The process continues until at least k grafted elements are obtained either above or below the center c of the hyper-product. An appropriate number of elements, determined by the output combinations that were used, is then pruned using the hyper-product pruning process. The factory then outputs the partial order generated.

The hyper-product construction process of a subfactory must be able to use all the constructs that may be fed into the subfactory as inputs. For each possible input construct there should also be at least one grafting process that can utilize it. Typically, a subfactory would have more than one such grafting process for each possible input construct. This gives the subfactory the freedom to choose different grafting processes in different circumstances and helps balance the upper and lower costs of the subfactory.

Each grafting process A_i has a list $a_{i,1}, a_{i,2}, \ldots, a_{i,l_i}$ of possible outcomes. Let $c_1(a_{i,j})$ and $c_0(a_{i,j})$ denote the upper and lower costs of $a_{i,j}$ (these costs are defined in the next subsection) and let $n_1(a_{i,j})$ and $n_0(a_{i,j})$ denote the number of elements above and below c, respectively, in the outcome $a_{i,j}$. When applying the grafting process A_i we do not know in advance which outcome will result. This is decided by the adversary.

For accounting purposes, it is convenient to view pruning as a special grafting process. We therefore refer to pruning as the 0th grafting process A_0 and let $a_{0,0}$ and $a_{0,1}$ stand for lower and upper pruned elements. The costs $c_0(a_{0,0})$ and $c_1(a_{0,1})$ are just and lower and upper element pruning costs. This special "grafting" process is different from the other grafting processes in two major respects. The first is that the algorithm, and not the adversary, chooses the outcome. The second is that elements are not pruned one at a time. Instead, two counters are used to maintain the number of elements that should be pruned above and below the center. The required numbers of elements are then pruned after all the grafting processes have finished.

An output combination $C_i = (r_{i,1} \times b_{i,1}, r_{i,2} \times b_{i,2}, \ldots, r_{i,s_i} \times b_{i,s_i})$ is a weighted list of outcomes, where $r_{i,1}, r_{i,2}, \ldots, r_{i,s_i} > 0$ are *real* numbers and each $b_{i,j}$ is an outcome $a_{i',j'}$ of one of the grafting processes A_0 and $A_1, \ldots, A_\ell$ employed by the subfactory (the use of real numbers will be justified later). Note that A_0 is the pruning process of the subfactory, disguised as a grafting process, and an output combination may therefore involve elements that should be obtained by pruning. In our factories most combinations involve only two outcomes.

For each possible outcome $a_{i,j}$ of the grafting processes, the subfactory maintains a counter $\#(a_{i,j})$ of the number of outcomes of type $a_{i,j}$ which were obtained but not

yet consumed. Initially $\#(a_{i,j}) = 0$ for every $i \geq 1$ and $j \geq 1$, and $\#(a_{0,j}) = 2k$, for $j = 0, 1$. Recall that $a_{0,0}$ and $a_{0,1}$ represent pruned elements. The initial values given to $\#(a_{0,0})$ and $\#(a_{0,1})$ reflect the fact that up to $2k$ elements can be pruned on either side of the hyper-product constructed by the subfactory. Two more counters k_0 and k_1 maintain the number of elements below and above the center c that were already placed in the output partial order. The counters k_0 and k_1 are initially set to 0.

An output combination $C_i = (r_{i,1} \times b_{i,1}, r_{i,2} \times b_{i,2}, \ldots, r_{i,s_i} \times b_{i,s_i})$ can be applied if $\#(b_{i,j}) \geq r_{i,j}$ for every $1 \leq j \leq s_i$. If these conditions are satisfied, then $r_{i,j}$ copies of outcome $b_{i,j}$, for $1 \leq j \leq s_i$, are "placed" in the output partial order and all the counters are updated accordingly. The output combination C_i contains $n_0(C_i) = \sum_{j=1}^{s_i} r_{i,j} n_0(b_{i,j})$ elements below the center c and $n_1(C_i) = \sum_{j=1}^{s_i} r_{i,j} n_1(b_{i,j})$ elements above the center c of the hyper-product. All the combinations used in our factories satisfy the condition $\frac{1}{2} < n_0(C_i)/n_1(C_i) < 2$. The partial orders produced by our factories are therefore of the form $S_{k'}^{k''}$, where $k \leq k' \leq 2k$ and $k \leq k'' \leq 2k$, and hence members of $\tilde{\mathcal{S}}_k^k$. The factor 2 used in the definition of $\tilde{\mathcal{S}}_k^k$ is arbitrary and can be increased if necessary.

The subfactory applies each grafting process A_i, where $1 \leq i \leq \ell$, sufficiently many times so that there is at least one outcome $a_{i,j}$ for which $\#(a_{i,j}) \geq \mathit{quota}$, where *quota* is some sufficiently large constant. A suitable choice for quota is the maximum $\max_{i,j} r_{i,j}$ of the coefficients that appear in the output combinations of the subfactory. The combination list of the subfactory should have the property that if for each $1 \leq i \leq l$ there exists at least one $1 \leq j \leq l_i$ such that $\#(a_{i,j}) \geq$ quota, then at least one of the output combinations can be applied. This ensures that the subfactory will never get "stuck," no matter what the outcomes of the grafting processes will be.

In the above description, we speak freely about placing $r_{i,j}$ copies of an outcome $b_{i,j}$ in the output partial order, although $r_{i,j}$ was not necessarily an integer. This was done, however, only for accounting purposes. The outcomes (or fractions of outcomes) placed in the output partial order are the outcomes whose cost was already accounted for. At most a fixed number of outcomes of each type are unaccounted for by this analysis. Their contribution to the cost, therefore, is negligible. These unaccounted-for outcomes can either be placed in the output partial order or be used to construct the next output partial order.

A pseudocode describing the operation of a generic subfactory is given in Figure 6.1.

6.2. The analysis of a generic subfactory. We now turn to the complexity analysis of a generic subfactory. We start by defining the lower and upper costs of each outcome $a_{i,j}$ of the grafting processes. Let $I_1, \ldots, I_r$ be the input constructs consumed by the flow of the grafting process A_i that produces $a_{i,j}$. Let $V_1, \ldots, V_s$ be the constructs above the center c of the hyper-product from which the outcome $a_{i,j}$ is composed, and let $\Lambda_1, \ldots, \Lambda_t$ be the constructs below the center c from which $a_{i,j}$ is composed. Let $R_1, \ldots, R_p$ be the leftovers of this process, i.e., the parts of $I_1, \ldots, I_r$ that do not become parts of either $V_1, \ldots, V_s$ or $\Lambda_1, \ldots, \Lambda_t$. The leftovers $R_1, \ldots, R_p$ are recycled to the appropriate subfactory.

The *generation cost*, $\mathit{gen}(a_{i,j})$, of an outcome $a_{i,j}$ is the number of edges cut during the generation of an $a_{i,j}$. The *lower separation cost*, $\mathit{sep}_0(a_{i,j})$, of an $a_{i,j}$ is the number of edges that have to be cut to separate the constructs $\Lambda_1, \ldots, \Lambda_t$ from the center c of the hyper-product. The *lower elimination cost*, $\mathit{elm}_0(a_{i,j})$, of an $a_{i,j}$ is the number of edges that have to be cut to turn all the elements of $\Lambda_1, \ldots, \Lambda_t$ into singletons (and disconnect them from the center). The Hasse diagram of most

1. Generate a hyper-product with at least $2k$ elements on either side of its center c.
2. Initialize the counters:
   $\#(a_{0,j}) \leftarrow 2k$ for $j = 0, 1$;
   $k_0, k_1 \leftarrow 0$;
3. Perform the following steps until $k_0 \geq k$ and $k_1 \geq k$:
   (a) `for` $i \leftarrow 1$ `to` l `do` /* Activate grafting processes */
   `while` $\max\{\#(a_{i,1}), \ldots, \#(a_{i,l_i})\} < quota$ activate grafting process A_i.
   (b) `for` $i \leftarrow 1$ `to` m `do` /* Find appropriate output combinations */
   `if` $\#(b_{i,j}) \geq r_{i,j}$ `for` $1 \leq j \leq s_i$ `then`
   {
   $\#(b_{i,j}) \leftarrow \#(b_{i,j}) - r_{i,j}$ for $1 \leq j \leq s_i$;
   $k_0 \leftarrow k_0 + \sum_{j=1}^{s_i} r_{i,j} \cdot n_0(b_{i,j})$;
   $k_1 \leftarrow k_1 + \sum_{j=1}^{s_i} r_{i,j} \cdot n_1(b_{i,j})$
   }
4. Let $p_0 \leftarrow 2k - \#(a_{0,0})$ and $p_1 \leftarrow 2k - \#(a_{0,1})$;
   Prune p_0 and p_1 elements below and above c, respectively.
5. Output the partial order and return to 1.

FIG. 6.1. *The generic subfactory algorithm.*

outcomes is acyclic and the lower elimination cost in such a case is just the number of elements contained in the constructs $\Lambda_1, \ldots, \Lambda_t$. The *upper separation cost*, $sep_1(a_{i,j})$, and the *upper elimination cost*, $elm_1(a_{i,j})$, are defined analogously.

When the lower side of an outcome $a_{i,j}$ is recycled, the upper side is eliminated and vice versa. Hence, we define the *lower recycling cost*, $rec_0(a_{i,j})$, of the outcome $a_{i,j}$ as $sep_0(a_{i,j}) + elm_1(a_{i,j})$. The *upper recycling cost*, $rec_1(a_{i,j})$, of $a_{i,j}$ is defined analogously as $sep_1(a_{i,j}) + elm_0(a_{i,j})$.

Each input construct I_l, where $1 \leq l \leq r$, used in the flow of the grafting process that generates the outcome $a_{i,j}$ carries a credit of $credit(I_l)$ units. These credit units help "finance" the generation of $a_{i,j}$. If the lower part of the partial order generated using the outcome $a_{i,j}$ is eventually eliminated and its upper part recycled, we have to attach $credit(V_l)$ credit units to each of the constructs $V_1, \ldots, V_s$. If, on the other hand, the upper part is eliminated and the lower part recycled, we have to attach $credit(\Lambda_l)$ credit units to each of the constructs $\Lambda_1, \ldots, \Lambda_t$. Similarly, we have to attach $credit(R_l)$ credit units to each of the leftover $R_1, \ldots, R_p$. The *lower cost* $c_0(a_{i,j})$ and the *upper cost* $c_1(a_{i,j})$ of the outcome $a_{i,j}$ are thus defined as

$$c_0(a_{i,j}) = gen(a_{i,j}) + rec_1(a_{i,j}) - \sum_{l=1}^{r} credit(I_l) + \sum_{l=1}^{s} credit(V_l) + \sum_{l=1}^{p} credit(R_l),$$

$$c_1(a_{i,j}) = gen(a_{i,j}) + rec_0(a_{i,j}) - \sum_{l=1}^{r} credit(I_l) + \sum_{l=1}^{t} credit(\Lambda_l) + \sum_{l=1}^{p} credit(R_l).$$

As mentioned before, our grafting procedures may use mass production. Hence, the costs $gen(a_{i,j})$, $rec_0(a_{i,j})$, and $rec_1(a_{i,j})$ are amortized costs and $I_1, \ldots, I_r$ are the constructs used for the construction of a single copy of the outcome $a_{i,j}$.

Let $C_i = (r_{i,1} \times b_{i,1}, \ldots, r_{i,s_i} \times b_{i,s_i})$ be an output combination. The *local upper element cost*, $u_1(C_i)$, and the *local lower element cost*, $u_0(C_i)$, of the combination C_i are defined, temporarily, as

$$u_0(C_i) = \sum_{j=1}^{s_i} r_{i,j} \cdot c_0(b_{i,j}) \Big/ \sum_{j=1}^{s_i} r_{i,j} \cdot n_0(b_{i,1}) \ , \ u_1(C_i) = \sum_{j=1}^{s_i} r_{i,j} \cdot c_1(b_{i,j}) \Big/ \sum_{j=1}^{s_i} r_{i,j} \cdot n_1(b_{i,1}).$$

This definition, however, is not completely adequate. So far, we have attached credits to individual constructs. We would sometimes like to attach credits to combinations of constructs. It turns out that there are some constructs that can be more efficiently utilized when constructs of a different kind are also available. As a concrete example, we will see in the next sections that quartets (P_{00}'s and P_{11}'s) can be utilized much more efficiently if some pairs (P_0's and P_1's) are also available. More specifically, we will be in a situation in which $credit(P_0) = 0$ and $credit(P_{00}) > 0$, i.e., pairs carry no credit while each quartet on its own should carry some positive credit. In contrast, a combination composed of a quartet and pair, and in fact a combination composed of up to a fixed constant $\gamma > 1$ of quartets and of a single pair, could be recycled without any credit, i.e., $credit(P_0, \gamma \times P_{00}) = 0$. The credit that needs to be attached to a collection of constructs may therefore be smaller than the sum of the individual credits that should be attached to each member of this collection.

The above temporary definition of $u_0(C_i)$ and $u_1(C_i)$ does not take such considerations into account. It is not enough to replace the sums $\sum_{l=1}^{s} credit(V_l)$, $\sum_{l=1}^{t} credit(\Lambda_l)$, and $\sum_{l=1}^{p} credit(R_l)$ in the definitions of $c_0(a_{i,j})$ and $c_1(a_{i,j})$ with $credit(V_1, \ldots, V_s)$, $credit(\Lambda_1, \ldots, \Lambda_t)$, and $credit(R_1, \ldots, R_p)$, respectively, as good combinations may be formed using constructs obtained while constructing or recycling different outcomes that participate in the output combination. As an example, pairs are perhaps obtained when the $b_{i,1}$'s outcomes that participate in the output combination are recycled while quartets are perhaps obtained when the $b_{i,2}$'s outcomes are recycled. We therefore amend the definitions of $u_0(C_i)$ and $u_1(C_i)$ in the following way:

$$u_0(C_i) = \frac{\left\lfloor \sum_{j=1}^{s_i} r_{i,j} \cdot (gen(b_{i,j}) + rec_1(b_{i,j})) \right\rfloor - \sum_{I \in \mathcal{I}_i} credit(I) + credit(\mathcal{V}_i) + credit(\mathcal{R}_i)}{\sum_{j=1}^{s_i} r_{i,j} \cdot n_0(b_{i,j})}, \tag{6.1}$$

$$u_1(C_i) = \frac{\left\lfloor \sum_{j=1}^{s_i} r_{i,j} \cdot (gen(b_{i,j}) + rec_0(b_{i,j})) \right\rfloor - \sum_{I \in \mathcal{I}_i} credit(I) + credit(\Lambda_i) + credit(\mathcal{R}_i)}{\sum_{j=1}^{s_i} r_{i,j} \cdot n_1(b_{i,j}))}, \tag{6.2}$$

where $\mathcal{I}_i$, Λ_i, $\mathcal{V}_i$, and $\mathcal{R}_i$ are the (weighted) collections of all input constructs, recycled constructs, and leftovers involved in the generation and the recycling of all the outcomes composing the combination C_i. The terms $credit(\mathcal{V}_i)$, $credit(\Lambda_i)$, and $credit(\mathcal{R}_i)$ represent the credits that should be attached to the collections $\mathcal{V}_i$, Λ_i, and $\mathcal{R}_i$.

In practice, credits are attached separately to most constructs involved in the collections Λ_i, $\mathcal{V}_i$, and $\mathcal{R}_i$. In the factory described in section 8, for example, the only exception to this is the grouping of pairs and quartets together and the formation of compound constructs of the form $(P_0, \gamma \times P_{00})$.

In general, each factory $\mathcal{G}$ has a set $\mathcal{P}$ of both simple and compound constructs to which credits are assigned. The set $\mathcal{P}$ includes all the basic constructs that can be consumed and that are recycled by the factory. Every collection Λ_i, $\mathcal{V}_i$, or $\mathcal{R}_i$ of constructs is then treated as a combination of constructs taken from $\mathcal{P}$ in a way that minimizes the total credit that should be attached to the collection.

The lower and upper element costs $u_0(g)$ and $u_1(g)$ of the subfactory g with a list $C_1, \ldots, C_m$ of output combinations are defined to be

$$u_0(g) = \max_{i=1}^{m} u_0(C_i), \qquad u_1(g) = \max_{i=1}^{m} u_1(C_i).$$

We are now in a position to state the following theorem.

THEOREM 6.1. *If $\mathcal{G}$ is a factory that employs the subfactories $g_1, g_2, \ldots, g_r$, then the lower and upper element costs of the factory $\mathcal{G}$ are*

$$u_0(\mathcal{G}) \sim \max_{j=1}^{r} u_0(g_j), \qquad u_1(\mathcal{G}) \sim \max_{j=1}^{r} u_1(g_j).$$

Proof. Let $\mathcal{P}$ be the set of basic and compound constructs recycled by the factory $\mathcal{G}$. We assume that for each construct from $\mathcal{P}$ there is at least one subfactory among $g_1, g_2, \ldots, g_r$ that can consume it. We also assume that each subfactory g_j, where $1 \le j \le r$, has an adequate list of output combinations, in the sense that if each grafting process employed by g_j is applied sufficiently many times, then at least one output combination can be used. We also assume that if enough input constructs are fed into the factory, then at least one subfactory can generate a partial order from $\tilde{\mathcal{S}}_k^k$.

Each construct from $\mathcal{P}$ has a specific amount of credit attached to it. The credit attached to singletons (P_λ's) is zero. The singletons that are initially fed into the factory thus carry the required amount of credit. Whenever constructs from $\mathcal{P}$ are recycled, we make sure that they too carry the correct credit. Each amount of credit used in the construction of a partial order from $\tilde{\mathcal{S}}_k^k$ is therefore paid for in full during the generation of other partial orders from $\tilde{\mathcal{S}}_k^k$.

The local upper and lower element costs of a combination C_i used in one of the factories were defined as $u_0(C_i) = c_0(C_i)/n_0(C_i)$ and $u_1(C_i) = c_1(C_i)/n_1(C_i)$, where $c_0(C_i)$ and $c_1(C_i)$ are the amortized lower and upper costs of the combination C_i (the numerators of (6.1) and (6.2)), and $n_0(C_i)$ and $n_1(C_i)$ are the number of elements in C_i above and below the center (the denominators of (6.1) and (6.2)). Let $u_0 = \max_{j=1}^{r} u_0(g_j)$ and $u_1 = \max_{j=1}^{r} u_1(g_j)$. It follows that for every combination C_i used in one of the subfactories $g_1, \ldots, g_r$ we have $u_0(C_i) \le u_0$ and $u_1(C_i) \le u_1$.

Suppose that an output partial order is generated in subfactory g_j. Suppose that $C_{i_1}, C_{i_2}, \ldots, C_{i_a}$ are the output combinations that contribute to this output partial order. The total amortized cost of producing the partial order, eliminating its lower side and recycling its upper side, is $\sum_{l=1}^{a} c_0(C_{i_l})$. The total number of elements below the center of the produced partial order is $\sum_{l=1}^{a} n_0(C_{i_l})$. The amortized, per-element cost of the produced partial order is therefore $\sum_{l=1}^{a} c_0(C_{i_l}) \Big/ \sum_{l=1}^{a} n_0(C_{i_l})$. As for each $1 \le l \le a$ we have $c_0(C_{i_l})/n_0(C_{i_l}) \le u_0$, we get that $\sum_{l=1}^{a} c_0(C_{i_l}) \Big/ \sum_{l=1}^{a} n_0(C_{i_l}) \le u_0$.

Similarly, we also get that the amortized, per-element, upper cost of the produced partial order is at most u_1.

The above accounting did not take into account the cost of breaking the unused parts of the hyper-product. The cost of this operation is negligible, however, compared with the generation cost of the output partial order. It also did not take into account the generation and recycling costs of the outcomes of the grafting processes that were not used to construct the output partial order. However, there may be only a constant number of such outcomes and the costs associated with them are therefore also negligible.

As a result, we get that $u_0(\mathcal{G}) = u_0 + o(1)$ and $u_1(\mathcal{G}) = u_1 + o(1)$ and the theorem is proved. □

Our concrete factories are described in sections 8 and 9. For each one of our factories we specify the set $\mathcal{P}$ of basic and compound constructs used and the credit attached to each one of its elements. We then describe the subfactories employed by the factory. For each subfactory we specify the hyper-product generation process and grafting processes used and, finally, its list of output combinations. We verify that each construct from $\mathcal{P}$ can indeed be consumed by at least one subfactory and that the list of output combinations of each subfactory is adequate. For each output combination we then compute its local lower and upper element costs, using the definitions given in (6.1) and (6.2). As implied by Theorem 6.1, the maxima of these values are then the lower and upper element costs of the whole factory. We begin the description of our factories by describing, in the next section, the grafting processes used by them.

7. Advanced grafting processes. In this section we describe the grafting processes used by our improved factories.

7.1. Recursive pair grafting. The recursive pair grafting is, as its name suggests, a recursive version of the basic pair grafting procedure described in subsection 4.2. There are two variants of this process depending on whether the upper elements or lower elements of pairs are compared to the center of the hyper-product. We describe the variant in which lower elements are compared. The other variant is symmetric.

The recursive pair grafting recursively builds hyperpairs which are *dominated* by the center c of the hyper-product. A hyperpair $P = P_{1^i}$ with center c' is dominated by c if every element of P, except possibly c', is known to be smaller than c. A dominated hyperpair P_{1^i} is said to be of *level* i.

The process is composed of rounds. The ith round receives two dominated hyperpairs P^1 and P^2 (with centers c_1 and c_2, respectively) of level i and attempts to construct a dominated hyperpair of level $i+1$. This is done by first comparing the centers c_1 and c_2 of these two dominated hyperpairs, thus obtaining a hyperpair $P = P_{1^{i+1}}$. We may assume, without loss of generality, that $c_2 > c_1$ and c_2 is therefore the center of the new hyperpair formed. We then compare c_1 (i.e., *not* c_2, the center of the hyperpair P) with c. The two possible outcomes are as follows:

(1) $c_1 < c$ and P is a dominated hyperpair of level $i+1$;

(2) $c_1 > c$ and P is not dominated by c (as $c_2 > c_1 > c$).

If (2) occurs, the process is stopped. For the purposes of $\mathcal{G}_k$ and $\mathcal{G}'_k$ (the factories described in sections 8 and 9) we also stop the process when a dominated hyperpair P_{111} of level 3 is generated. We then separate the generated hyperpair into two hyperpairs: a dominated hyperpair P^1 of level 2 and a hyperpair P^2 of level 2 all whose elements are known to be below the center. The hyperpair P^2 is an output

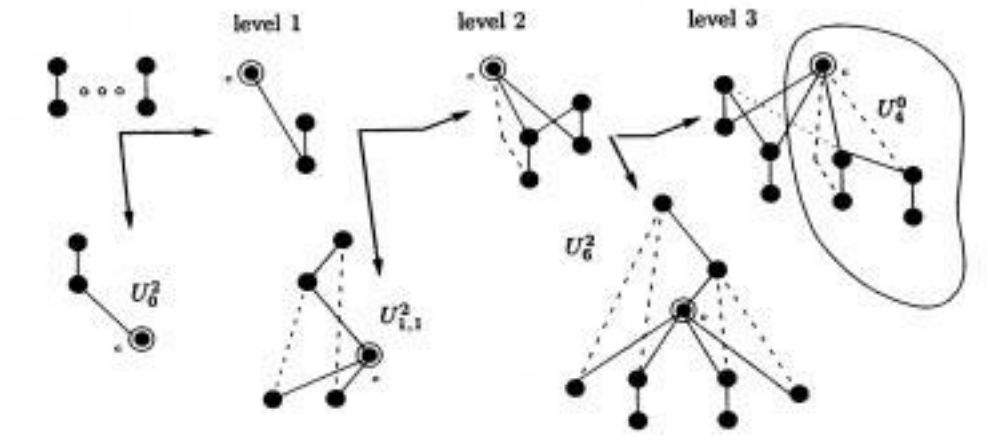


FIG. 7.1. *The recursive pair grafting process.*

of this grafting process. The dominated hyperpair P^1 is used to construct the next dominated hyperpair of level 3.

As described, the two hyperpairs P^1 and P^2 fed into the 0th round of the process are two singletons c_1 and c_2. The 0th round then starts by comparing these two singletons, thus forming a pair. Instead of feeding the 0th round of the process with singletons, we can therefore feed it with pairs and simply skip the first comparison of this round.

The recursive pair grafting process is described schematically in Figure 7.1. The dashed edges in the figure represent edges that became redundant. The four possible outcomes of this process are denoted by U_0^2, $U_{1,1}^2$, U_6^2, and U_4^0. These four outcomes, as well as the four outcomes L_2^0, $L_2^{1,1}$, L_2^6, and L_0^4 of the symmetric grafting process in which upper elements are compared to the center, are shown in Figure 7.2.

We now turn to the cost analysis of this grafting process. As explained, we do not count the number of comparisons made but rather the number of edges cut. Let us analyze the number of edges cut during the ith round of the process. Let P^1 and P^2 be the two dominated hyperpairs of level i fed to the ith round, let c_1 and c_2 be their centers, and assume, without loss of generality, that the first comparison of the round established that $c_1 < c_2$. It is easy to verify, using induction, that i edges connect c with elements of P^1 and that i edges connect c with elements of P^2. Similarly, i edges connect c_1 with other elements of P^1 and i edges connect c_2 with other elements of P^2. If the ith round results in a dominated hyperpair of level $i+1$, i.e., if $c_1 < c$ (case (1)), then the i edges connecting c with the elements of P^1 become redundant and therefore are cut. If on the other hand $c_1 > c$ (case (2)), then the $2i$ edges connecting c_1 and c_2 with elements of their hyperpairs become redundant and therefore are cut.

Finally, note that an additional edge is cut when a dominated hyperpair P_{111} of level 3 is separated into a dominated hyperpair of a level 2 and to a U_4^0. Using these observations, it is easy to verify that the generation costs of the different outcomes shown in Figure 7.2 are as shown in Table 7.1.

7.2. Balanced quartet grafting. The balanced quartet grafting process is a very straightforward process. We describe the process that grafts P_{00}'s. A symmetric

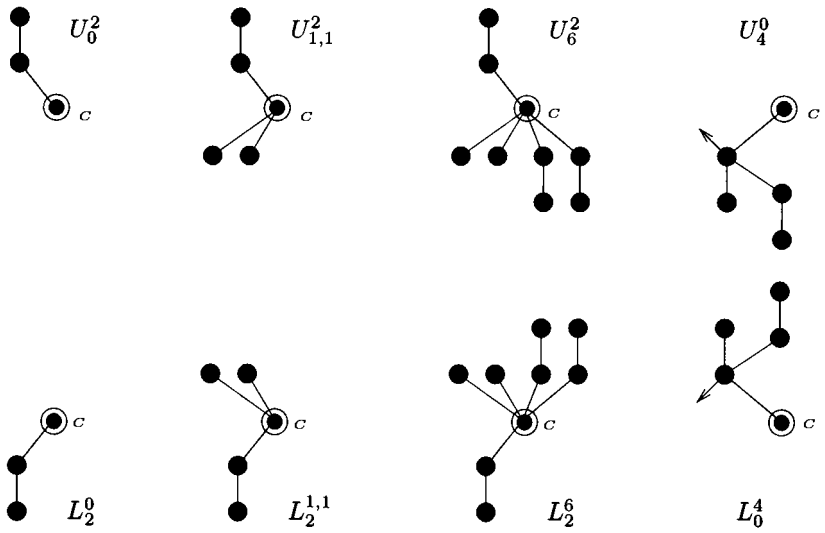


FIG. 7.2. *Possible outcomes of recursive pair grafting.*

TABLE 7.1
The costs of the different outcomes of the recursive pair grafting processes.

Outcome	Above	Below	*gen*	rec_0	rec_1
U_0^2	P_0	—	0	2	1
$U_{1,1}^2$	P_0	$2\times P_\lambda$	2	4	3
U_6^2	P_0	$2\times P_\lambda$, $2\times P_0$	6	6	7
U_4^0	—	P_{11}	4	1	4
Outcome	Above	Below	*gen*	rec_0	rec_1
L_2^0	—	P_0	0	1	2
$L_2^{1,1}$	$2\times P_\lambda$	P_0	2	3	4
L_2^6	$2\times P_\lambda$, $2\times P_0$	P_0	6	7	6
L_0^4	P_{00}	—	4	4	1

process can be used for grafting P_{11}'s. The grating process always consumes a single quartet and returns one of five possible outcomes.

Let u, v, w, z be the elements of a P_{00}, where $u < v$ and $u < w < z$. Start by comparing w with c. If $w > c$, cut the edge connecting w and u. The obtained outcome is denoted by Q_0^2. The remaining pair is recycled. If $w < c$, then compare v and z with c. The four different outcomes of these comparisons give rise to the outcomes $Q_2^{1,1}$, Q_3^1, and Q_4^0 depicted in Figure 7.3. Dashed edges in this figure again represent edges that became redundant during the grafting process.

The elements in a $\bar{Q}_3^1$ outcome satisfy all the relations satisfied by the elements of a Q_3^1 outcome. A $\bar{Q}_3^1$ outcome is therefore a special case of a Q_3^1 outcome and we shall not consider it separately. The generation and recycling costs of the remaining four possibilities are given in Table 7.2.

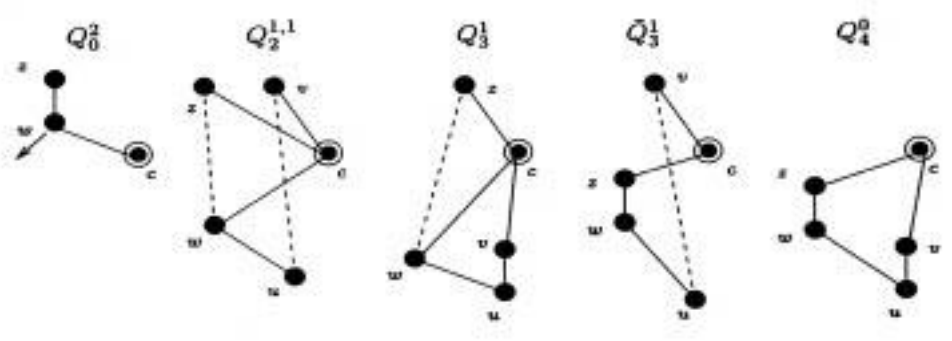


FIG. 7.3. *Possible outcomes of balanced quartet* (P_{00}) *grafting.*

TABLE 7.2
The costs of the different outcomes of the balanced P_{00} *grafting process.*

Outcome	Above	Below	Leftovers	*gen*	rec_0	rec_1
Q_0^2	P_0	—	P_0	1	2	1
$Q_2^{1,1}$	$2\times P_\lambda$	P_0	—	2	3	4
Q_3^1	P_λ	W_3	—	1	3	5
Q_4^0	—	P_{00}	—	1	2	5

7.3. Extended balanced quartet grafting. The grafting process described in this subsection is an extension of the simple grafting process described in the previous subsection. We again describe the process that grafts P_{00}'s. A symmetric process can be used for grafting P_{11}'s.

Let u, v, w, z be the elements of a P_{00}, where $u < v$ and $u < w < z$. The process starts again by comparing w and c. If $w < c$, continue as before, obtaining one of the outcomes $Q_2^{1,1}$, Q_3^1, or Q_4^0 shown in Figure 7.3. If, however, $w > c$, then instead of removing the redundant pair, activate the recursive pair grafting process, as described in subsection 7.1, on the redundant pair $u < v$. The four possible outcomes of this process are shown in Figure 7.4. The outcomes Q_0^4, Q_2^6, Q_6^{10}, and Q_4^4 are obtained, respectively, from the U_0^2, $U_{1,1}^2$, U_6^2, and U_4^0 outcomes of the recursive pair grafting process. The generation and recycling costs of all the outcomes of this grafting process are given in Table 7.3.

7.4. Unbalanced quartet grafting. The unbalanced quartet grafting process is another quartet grafting process used by our factories. Most of its outcomes are less balanced than the results obtained using the two previous quartet grafting processes. Again we describe the version of the process that grafts P_{00}'s. A symmetric version can be used to graft P_{11}'s.

Let u, v, w, z be the elements of a P_{00}, where $u < v$ and $u < w < z$. The two balanced quartet grafting processes started by comparing w with c, the center of the hyper-product. The unbalanced process, on the other hand, starts by comparing v and z to c. The possible outcomes are shown at the top of Figure 7.5.

In the first two cases, i.e., if $v, z < c$, or if $z < c < v$, the grafting process stops. The obtained outcomes are denoted, respectively, by R_4^0 and R_3^1. The other two cases, i.e., $v < c < z$ and $c < v, z$, are more complicated.

If $v < c < z$, use the pair $w < z$, and other pairs like it, as inputs to the

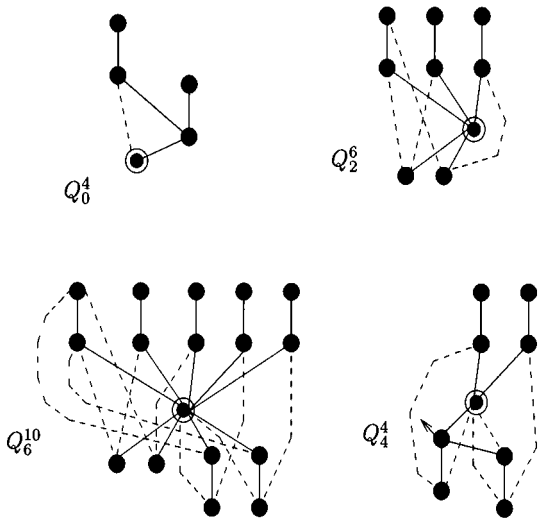


FIG. 7.4. *Possible outcomes of extended balanced grafting of P_{00}'s.*

TABLE 7.3
The costs of the different outcomes of the extended balanced P_{00} grafting process.

Outcome	Above	Below	*gen*	rec_0	rec_1
$Q_2^{1,1}$	—	P_0	2	3	4
Q_3^1	P_λ	W_3	1	3	5
Q_4^0	—	P_{00}	1	2	5
Q_0^4	P_{00}	—	1	4	1
Q_2^6	$3\times P_0$	$2\times P_\lambda$	4	8	5
Q_6^{10}	$5\times P_0$	$2\times P_\lambda$, $2\times P_0$	10	14	11
Q_4^4	$2\times P_0$	P_{11}	6	5	6

recursive P_0 grafting process, i.e., the mirror image of the grafting process described in subsection 7.1. The recursive pair grafting process results in the creation of $L_2^{1,1}$'s, L_2^6's, and L_0^4's. These $L_2^{1,1}$'s, L_2^6's, and L_0^4's are composed of w and z elements of P_{00}'s. By adding to the obtained constructs the u and the v elements of the quartets from which these w and z elements were taken, we obtain outcomes of the forms R_6^2, R_{10}^6, and R_4^4 shown in Figure 7.5.

Finally, if $c < v, z$, then the situation is identical to the situation after the first comparison of the second round of the recursive P_0 grafting. By continuing the recursive P_0 process from this point, possibly using another P_{00} for which $c < v, z$, we obtain an $L_2^{1,1}$, L_2^6, or L_0^4.

The unbalanced P_{00} grafting has eight outcomes in total: R_4^0, R_3^1, R_6^2, R_{10}^6, R_4^4, $L_2^{1,1}$, L_2^6, and L_0^4. The last three outcomes are excellent outcomes because we have been able to use quartets as if they were pairs. The costs associated with the outcomes R_4^0, R_3^1, R_6^2, R_{10}^6, and R_4^4 are given in Table 7.4. The costs of the outcomes $L_2^{1,1}$, L_2^6, and L_0^4 are as shown in Table 7.1.

8. A factory with $u_0, u_1 \sim 2.955$. The construction of the factory $\mathcal{G}_k$ satisfying the conditions of Theorem 2.3 is extremely involved. In this section we give a

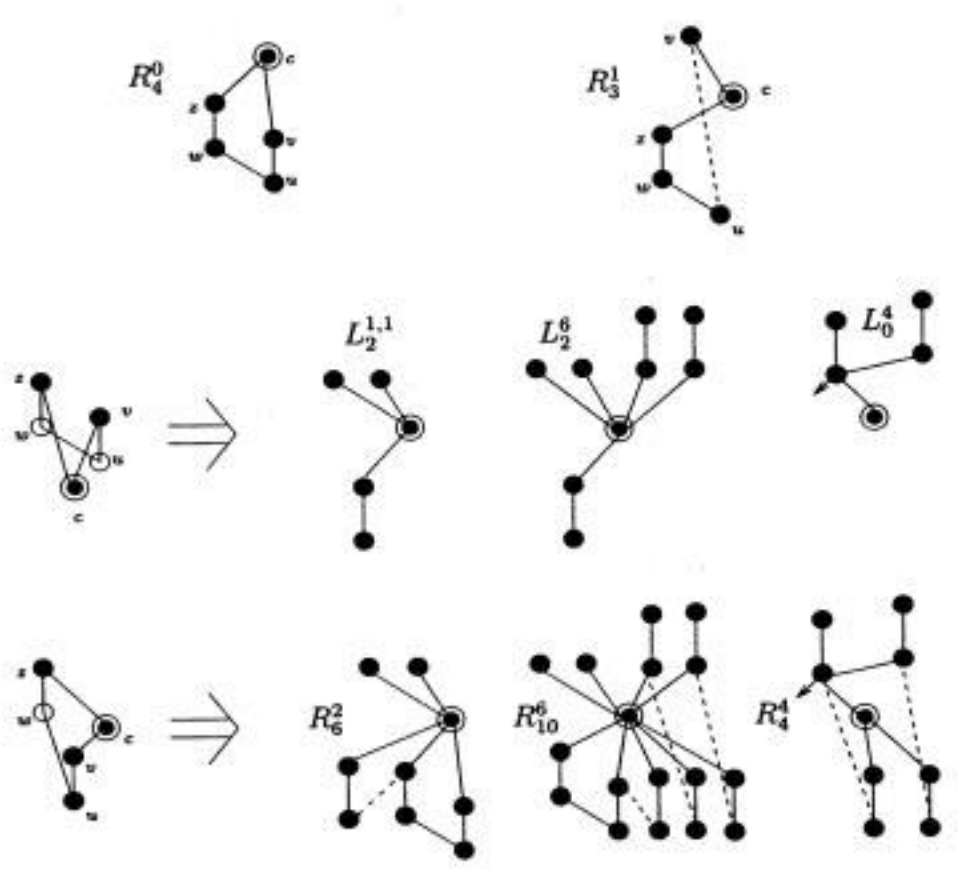


FIG. 7.5. *Possible outcomes of unbalanced P_{00} grafting.*

TABLE 7.4
The costs of the different outcomes of the unbalanced P_{00} grafting process.

Outcome	Above	Below	*gen*	rec_0	rec_1
R_4^0	—	P_{00}	0	2	5
R_3^1	P_λ	I_3	1	2	4
R_6^2	$2\times P_\lambda$	P_{00}, P_0	3	5	9
R_{10}^6	$2\times P_\lambda$, $2\times P_0$	$3\times P_0$, P_{00}	9	11	15
R_4^4	P_{00}	$2\times P_0$	6	6	5

Additional outcomes: $L_2^{1,1}$, L_2^6, L_0^4.

complete description of a simplified version $\mathcal{G}'_k$ of the factory $\mathcal{G}_k$, thereby proving the following theorem, which is only slightly weaker than Theorem 2.3.

THEOREM 8.1. *There is a green factory $\mathcal{G}'_k$ for $\tilde{\mathcal{S}}_k^k$ with $u_0, u_1 \sim 2.955$.*

As was the case with all the other factories we considered, the initial cost and production residues of $\mathcal{G}'_k$ are $O(k^2)$.

The constructs processed by the factory $\mathcal{G}'_k$ are shown in Figure 8.1. They are singletons P_λ's, pairs P_0/P_1's, quartets P_{00}/P_{11}'s, chains of length three I_3's, and triples W_3/M_3's. The factory $\mathcal{G}'_k$ also processes compound constructs of the forms

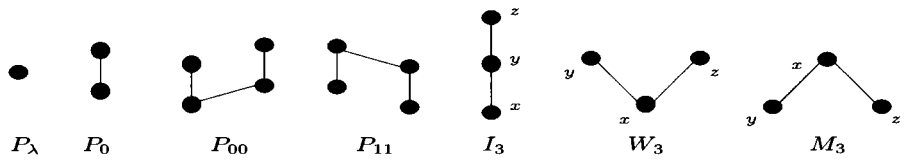


FIG. 8.1. *The basic input constructs of the factory* $\mathcal{G}'_k$.

TABLE 8.1
The credits attached to the constructs used by the factory $\mathcal{G}'_k$.

Construct	Credit
P_λ	0
P_0, P_1	0
P_{00}, P_{11}	0.1093
I_3, M_3, W_3	0.5000
$(P_0, \gamma \times P_{00}), (P_1, \gamma \times P_{11})$	0

$(P_0, \gamma \times P_{00})$ and $(P_1, \gamma \times P_{11})$, where $\gamma \simeq 2.6603$. The credits attached to these basic and compound constructs are given in Table 8.1. Note that singletons have zero credit attached to them. Note also that quartets have a positive credit attached to them while the constructs $(P_0, \gamma \times P_{00})$ and $(P_1, \gamma \times P_{11})$ have zero credit attached to them. This reflects the fact that quartets can be more efficiently utilized if pairs are also supplied with them.

The factories $\mathcal{G}'_k$ are composed of the following four subfactories:

1. $(P_0, \gamma \times P_{00})$ subfactory,
2. $(P_1, \gamma \times P_{11})$ subfactory,
3. P_{00} subfactory,
4. P_{11} subfactory.

The $(P_1, \gamma \times P_{11})$ subfactory is the mirror image of the $(P_0, \gamma \times P_{00})$ subfactory and the P_{11} subfactory is the mirror image of the P_{00} subfactory. The description of the $(P_0, \gamma \times P_{00})$ and the P_{00} subfactories are given in the next two subsections.

Because singletons (P_λ's) and pairs (P_0's) can be converted to quartets (P_{00}'s) at no cost, the $(P_0, \gamma \times P_{11})$ subfactory may also receive singletons and pairs.

The constructs W_3, M_3, and I_3 cannot be utilized directly by the four subfactories mentioned above. We therefore use the following simple process (carried out in a workshop?) to transform two W_3's into a P_0 and a P_{00}: Let $x_1, y_1 > z_1$ and $x_2, y_2 > z_2$ be two W_3's. Compare x_1 with x_2. Assume, without loss of generality, that $x_1 < x_2$. By removing the edge $x_2 > z_2$ we obtain the P_{00} x_1, y_1, z_1, x_2, where $z_2 < y_1$ and $z_2 < x_1 < x_2$, and the pair $z_2 < y_2$.

A single edge is cut in this process. As $\gamma > 1$, a combination of a P_0 and a P_{00} can be recycled without attaching any credit to it. Formally, this combination may be treated as $\frac{1}{\gamma} \times (P_0, \gamma \times P_{00})$ plus $(1 - \frac{1}{\gamma}) \times P_0$ and both these construct carry zero credit. The credits attached to two W_3's should therefore pay for cutting a single edge. This is exactly the case as $credit(W_3) = 0.5$.

Similarly, we can transform two M_3's into a P_1 and a P_{11}. An I_3 may be treated as either a W_3 or an M_3.

The factory $\mathcal{G}'_k$ described in this section can be improved if instead of using the above process for converting W_3's, M_3's, and I_3's into P_0's and P_{00}'s or P_{11}'s, we

used subfactories that could directly consume these constructs. The improved factory $\mathcal{G}_k$ (whose construction is sketched in the next section) includes, along with other additions and changes, a separate I_3 subfactory. The constructs I_3's turn out to be excellent raw materials for the construction of $\tilde{\mathcal{S}}_k^k$'s. Each I_3 carries, in $\mathcal{G}_k$, a credit of -0.4213 units and the presence of I_3's among the recycled elements becomes a boon, not a burden as in $\mathcal{G}'_k$.

We now describe the $(P_0, \gamma \times P_{00})$ subfactory and the P_{00} subfactory used by the $\mathcal{G}'_k$ factory.

8.1. The $(P_0, \gamma \times P_{00})$ subfactory. The input to this subfactory consists of constructs of the form $(P_0, \gamma \times P_{00})$, where $\gamma \simeq 2.6603$, with zero credit attached to each one of them. As singletons (P_λ's) and pairs (P_0's) can be converted to quartets (P_{00}'s) at no cost, the subfactory may also receive singletons and pairs with no credit attached to them. Equivalently, we can say that the input to this subfactory consists of singletons (P_λ's), pairs (P_0's), and quartets (P_{00}'s) with no credit attached to them, subject to the condition that each quartet is accompanied by at least $2/\gamma$ elements in singletons or pairs. The parameter γ is chosen to optimize the performance of the factory.

The $(P_0, \gamma \times P_{00})$ subfactory can also receive quartets with no accompanying singletons or pairs, provided that each such quartet carries a credit of $1/(1 + 2\gamma)$. Such a credit can pay for breaking a quartet P_{00} into $\frac{2}{2\gamma+1} \times (P_0, \gamma \times P_{00})$ which can be accepted with no credit. In the $\mathcal{G}'_k$ factory, $0.1093 \simeq credit(P_{00}) < 1/(1+2\gamma) \simeq 0.1582$. It is therefore cheaper to feed quartets without accompanying singletons or pairs into the P_{00} subfactory described in the next subsection.

Quartets fed into the subfactory are used for the construction of a hyper-product which, in this case, is simply a hyperpair. Quartets are also grafted using the balanced quartet grafting process described in subsection 7.2. Pairs are grafted using the recursive pair grafting process described in subsection 7.1.

8.1.1. The hyper-product generation process of the $(P_0, \gamma \times P_{00})$ subfactory. The hyper-products used in the $(P_0, \gamma \times P_{00})$ subfactory are simply the balanced hyperpairs constructed according to the string $\mathcal{W} = 01(10)^\omega$. Since P_{01}'s and P_{00}'s are in fact the same partial order (just the center is different), P_{00}'s can be used for the construction of such hyperpairs. The upper and lower element pruning costs of these hyperpairs are $pr_1(\mathcal{W}) = 1.5$ and $pr_0(\mathcal{W}) = 1.5$.

When r elements below the center of a hyperpair H_i are pruned, the hyperpair H_i is broken into a collection of smaller hyperpairs. This collection includes r singletons (P_λ's). All the other remaining hyperpairs are of size at least eight. A similar thing happens when r elements above the center of a hyperpair H_i are pruned. This time the collection of smaller hyperpairs obtained includes $\lfloor r/2 \rfloor$ pairs (P_0's). All the other hyperpairs are of size at least four. In both cases, when r elements are pruned, about r elements are obtained in singletons or pairs while all the other elements are contained in hyperpairs of size at least four. The hyperpairs of size at least four are immediately used for the construction of the next H_i hyperpairs. The singletons and pairs are too valuable to be used for this purpose. They are recycled. Their recycling enables the recycling of quartets, fed to the subfactory as parts of $(P_0, \gamma \times P_{00})$'s constructs, without the necessity of attaching credits to them.

If $r = \sum_{i=0}^{\ell} r_i$, where the r_i's are distinct powers of 2, then pruning r elements above the center c of a hyperpair H_i results in a collection of hyperpairs $P_{0^{r_1}}, P_{0^{r_2}}, \ldots, P_{0^{r_\ell}}$ all whose centers are above c. Similarly, pruning r elements below

TABLE 8.2
The costs of the pruning process of the $(P_0, \gamma \times P_{00})$ subfactory, viewed as a grafting process.

Outcome	Above	Below	Leftovers	*gen*	rec_0	rec_1
PR_0	—	$\frac{1}{4} \times P_{11}$	P_λ	1.5	0.25	1
PR_1	$\frac{1}{4} \times P_{00}$	—	$\frac{1}{2} \times P_0$	1.5	1	0.25

the center c of H_i results in a collection of hyperpairs $P_{1^{r_1}}, P_{1^{r_2}}, \ldots, P_{1^{r_\ell}}$, all whose centers are below c. All but at most three of such r pruned elements are contained in quartets (we can also assume, if convenient, that r is divisible by 4, in which case all the pruned elements are contained in quartets). All but at most seven of the pruned elements are contained in octets. The subfactories of the factory $\mathcal{G}'_k$ are not capable of utilizing octets. When pruned elements are recycled in the subfactories of the factory $\mathcal{G}'_k$, we have to break them, therefore, into a collection of quartets at a cost of 1/4 of a comparison per element. Some of the subfactories of the factory $\mathcal{G}_k$, described in the next section, are capable of utilizing 16-tuples (P_{0000}'s and P_{1111}'s). In the subfactories of $\mathcal{G}_k$, it is therefore enough to break recycled pruned elements into P_{0000}'s or P_{1111}'s at a cost of 1/16th of a comparison per element. In both cases, when these quartets or 16-tuples are recycled, an appropriate amount of credit should be attached to them.

As was already explained, it is convenient, for accounting purposes,to consider pruning as a special grafting process. It follows from the discussion above that we can consider the pruning process of the $(P_0, \gamma \times P_{00})$ subfactory as a grafting process with the characteristics described in Table 8.2. The pruning processes used in other subfactories will have other characteristics. We denote a downward pruned element by PR_0 and an upward pruned element by PR_1. This notation is used in all subfactories.

8.1.2. The output combinations of the $(P_0, \gamma \times P_{00})$ subfactory. The output combinations used by the $(P_0, \gamma \times P_{00})$ subfactory are given in Table 8.3. Each combination involves just two outcomes. The exact proportion of the two outcomes in each combination is chosen so that the lower and upper element costs of the combination become equal, thereby minimizing their maximum.

Before analyzing the costs of these combinations, we verify that if enough outcomes of each of the three grafting processes used by the subfactory are available, then at least one output combination can indeed be used. Suppose that at least one outcome of each of the two recursive pair grafting processes (i.e., at least one outcome of $U_0^2, U_{1,1}^2, U_6^2$, and U_4^0 and at least one outcome of $L_2^0, L_2^{1,1}, L_2^6$, and L_0^4) is available and that at least nine outcomes of the P_{00} grafting process (i.e., at least nine outcomes out of $Q_2^{1,1}, Q_3^1, Q_4^0$, and Q_0^2) are available. Each of the outcomes U_0^2, L_2^0, and $Q_2^{1,1}$ can immediately be used in conjunction with pruned elements (combinations 1, 2, or 3), since pruned elements are always available. If none of these outcomes are available, then we have at least three outcomes of either Q_3^1, Q_4^0, or Q_0^2, at least one of $U_{1,1}^2, U_6^2$, and U_4^0, and at least one of $L_2^{1,1}, L_2^6$, and L_0^4. If three Q_3^1's are available, we can therefore activate one of combinations 4, 5, or 6. If three Q_4^0's are available (actually two are enough here), we can activate one of combinations 7, 8, or 9. Finally, if three Q_4^0's are available, we can activate one of combinations 10, 11, or 12.

We now turn to the cost analysis of these combinations. We perform explicitly the computations for combinations 2 and 7. The other computations are similar.

- $(1. \times U_0^2, 2.1995 \times PR_0)$.

  The number $r \simeq 2.1995$ of pruned elements used in conjunction with U_0^2

TABLE 8.3
The output combinations of the $(P_0, \gamma \times P_{00})$ *subfactory.*

	Combination	Lower and upper element cost
1	$(\ 1.\times Q_2^{1,1}\ ,\ 0.4639\times PR_0\ )$	2.9059
2	$(\ 1.\times U_0^2\ ,\ 2.1995\times PR_0\ )$	2.9546
3	$(\ 1.\times L_2^0\ ,\ 2.1995\times PR_1\ )$	2.9546
4	$(\ 1.\times L_2^{1,1}\ ,\ 0.2269\times Q_3^1\ )$	2.9538
5	$(\ 1.\times L_2^6\ ,\ 2.4546\times Q_3^1\ )$	2.9499
6	$(\ 1.\times L_0^4\ ,\ 2.0254\times Q_3^1\ )$	2.9001
7	$(\ 1.\times L_2^{1,1}\ ,\ 0.1134\times Q_4^0\ )$	2.9546
8	$(\ 1.\times L_2^6\ ,\ 1.2271\times Q_4^0\ )$	2.9517
9	$(\ 1.\times L_0^4\ ,\ 1.0000\times Q_4^0\ )$	2.8954
10	$(\ 1.\times U_{1,1}^2\ ,\ 0.2038\times Q_0^2\ )$	2.9538
11	$(\ 1.\times U_6^2\ ,\ 2.2042\times Q_0^2\ )$	2.9483
12	$(\ 1.\times U_4^0\ ,\ 1.8150\times Q_0^2\ )$	2.9075

is chosen so that the lower and upper element costs of the combination would become equal. We demonstrate the computation that leads to this optimal choice of r.

Remembering that the input to the $(P_0, \gamma \times P_{00})$ subfactory consists of $(P_0, \gamma \times P_{00})$'s, we get that the generation of a U_0^2 consumes one pair P_0 and leaves $\gamma \times P_{00}$ as leftovers. Similarly, the generation of $r \times PR_0$ consumes $\frac{r}{2} \times P_{00}$ and leaves $r \times P_\lambda$ (consult Table 8.2) and $\frac{r}{2\gamma} \times P_0$ as leftovers. The leftover singletons are immediately joined into pairs. The leftovers from the two processes are therefore $\frac{r}{2}(1+\frac{1}{\gamma}) \times P_0$ and $\gamma \times P_{00}$. Provided that $\frac{r}{2}(1+\frac{1}{\gamma}) \geq 1$, these leftovers can be recycled without having to attach any credit to them.

The combination $(1.\times U_0^2, r \times PR_0)$ is composed of a P_0 above the center and $\frac{r}{4} \times P_{11}$ below the center. As no credit is attached to the inputs, nor paid for the leftovers, the local lower and upper element costs of this combination are

$$u_0(C_2) = \frac{1.\times(gen(U_0^2)+rec_1(U_0^2)) + r\times(gen(PR_0)+rec_1(PR_0)) + credit(P_0)}{1.\times n_0(U_0^2)+r\times n_0(PR_0)},$$

$$u_1(C_2) = \frac{1.\times(gen(U_0^2)+rec_0(U_0^2)) + r\times(gen(PR_0)+rec_0(PR_0)) + \frac{r}{4}\times credit(P_{11})}{1.\times n_1(U_0^2)+r\times n_1(PR_0)}.$$

Consulting Table 7.1 we get that $gen(U_0^2) = 0, rec_0(U_0^2) = 2, rec_1(U_0^2) = 1$, and of course that $n_0(U_0^2) = 0$ and $n_1(U_0^2) = 2$. Consulting Table 8.2 we get that $gen(PR_0) = 1.5, rec_0(PR_0) = 0.25, rec_1(PR_0) = 1$, and of course that $n_0(PR_0) = 1$ and $n_1(PR_0) = 0$. We also have $credit(P_{11}) = 0.1093$ and $credit(P_0) = 0$. Substituting these values into the above expressions and

equating the two costs we obtain the equation

$$\frac{1.\times(0+1)+r\times(1.5+1)}{1.\times 0+r\times 1}=\frac{1.\times(0+2)+r\times(1.5+0.25)+\frac{r}{4}\times 0.1093}{1.\times 2+r\times 0}\,.$$

It is easy to check that the solution of this equation is $r\simeq 2.1995$ and the values of both the lower and upper element costs in this case are $u_0(C_2)=u_1(C_2)\simeq 2.9546$.

- $(1.\times L_2^{1,1}, 0.1134\times Q_4^0)$.

  We again compute the lower and upper element costs of a combination $(1.\times L_2^{1,1}, r\times Q_4^0)$ and choose r so that both these costs would become equal. The generation of an $L_2^{1,1}$ consumes $2\times P_0$ and leaves $2\gamma\times P_{00}$ as leftovers. The generation of $r\times Q_4^0$ consumes $r\times P_{00}$ and leaves $\frac{r}{\gamma}\times P_0$ as leftovers. The proportion of quartets among the leftovers is much higher than in the previous case and they cannot all be recycled by pairing them with pairs. The $\frac{r}{\gamma}\times P_0$ can be paired with $r\times P_{00}$ of the leftovers to form $\frac{r}{\gamma}\times(P_0,\gamma\times P_{00})$ which can be recycled without credit. Credit, however, should be attached to the remaining $(2\gamma-r)\times P_{00}$ (assuming, as will be the case, that $2\gamma-r>0$).

  The combination $(1.\times L_2^{1,1}, r\times Q_4^0)$ is composed of $2\times P_\lambda$ above the center and a P_0 and a $r\times P_{00}$ below the center. The lower part is recycled to the $(P_0,\gamma\times P_{00})$ subfactory with no attached credit, as $r<\gamma$. The local lower and upper element costs of this combination are therefore

$$u_0(C_7)=\frac{1.\times(gen(L_2^{1,1})+rec_1(L_2^{1,1}))+r\times(gen(Q_4^0)+rec_1(Q_4^0))+(2\gamma-r)\times credit(P_{00})}{1.\times n_0(L_2^{1,1})+r\times n_0(Q_4^0)},$$
$$u_1(C_7)=\frac{1.\times(gen(L_2^{1,1})+rec_0(L_2^{1,1}))+r\times(gen(Q_4^0)+rec_0(Q_4^0))+(2\gamma-r)\times credit(P_{00})}{1.\times n_1(L_2^{1,1})+r\times n_1(Q_4^0)}.$$

Substituting the values $gen(L_2^{1,1})=2, rec_0(L_2^{1,1})=3, rec_1(L_2^{1,1})=4$, and $n_0(L_2^{1,1})=n_1(L_2^{1,1})=2$, found in Table 7.1, and the values $gen(Q_4^0)=1, rec_0(Q_4^0)=2, rec_1(Q_4^0)=5$, and $n_0(Q_4^0)=4, n_1(Q_4^0)=0$, found in Table 7.2, together with the value $credit(P_{00})=0.1093$ and equating these costs, we obtain the following equation:

$$\frac{1.\times(2+4)+r\times(1+5)+(2\cdot 2.6603-r)\cdot 0.1093}{1.\times 2+r\times 4}$$
$$=\frac{1.\times(2+3)+r\times(1+2)+(2\cdot 2.6603-r)\cdot 0.1093}{r\times 0+1.\times 2}\,.$$

It is easy to check that the solution of this equation is $r\simeq 0.1134$ and the values of both the lower and upper element costs in this case are $u_0(C_7)=u_1(C_7)\simeq 2.9546$.

The analysis of the other 10 cases is similar. Combinations 2, 3, and 7 turn out to be the worst combinations of this subfactory.

8.2. The P_{00} subfactory. The input to the P_{00} subfactory consists of P_{00}'s. Each P_{00} fed to the subfactory carries a credit of $credit(P_{00})\simeq 0.1093$. The subfactory constructs hyper-products using the process described below. The subfactory uses the extended balanced and the unbalanced P_{00} grafting processes described in subsections 7.3 and 7.4.

TABLE 8.4
The costs of the pruning process of the P_{00} subfactory, viewed as a grafting process.

Outcome	Above	Below	*gen*	rec_0	rec_1
PR_0	—	$\frac{1}{4}\times P_{11}$	1.4366	0.25	1
PR_1	$\frac{1}{4}\times P_{00}$	—	1.6268	1	0.25

TABLE 8.5
The output combinations of the P_{00} subfactory.

	Combination	Lower and upper element cost
1	$(\ 1.\times R_4^0 \quad , \quad 4.0000\times PR_1\ )$	2.9267
2	$(\ 1.\times R_3^1 \quad , \quad 2.0000\times PR_1\ )$	2.7481
3	$(\ 1.\times R_6^2 \quad , \quad 3.5464\times PR_1\ )$	2.9546
4	$(\ 1.\times R_{10}^6 \quad , \quad 3.5456\times PR_1\ )$	2.9508
5	$(\ 1.\times R_4^4 \quad , \quad 0.4096\times PR_1\ )$	2.8972
6	$(\ 1.\times Q_0^4 \quad , \quad 4.0000\times PR_0\ )$	2.7366
7	$(\ 1.\times Q_2^6 \quad , \quad 4.0000\times PR_0\ )$	2.9546
8	$(\ 1.\times Q_6^{10} \quad , \quad 4.0000\times PR_0\ )$	2.9509
9	$(\ 1.\times Q_4^4 \quad , \quad 0.4131\times PR_0\ )$	2.8790

Additional combinations involve $Q_2^{1,1}$, Q_3^1, Q_4^0, $L_2^{1,1}$, L_2^6, L_0^4 and pruning.

8.2.1. The hyper-product generation process of the P_{00} subfactory. The hyper-products used by the P_{00} subfactory are generated according to an infinite string $\mathcal{W}' = 011\mathcal{W}''$, where $\mathcal{W}''$ is an infinite string with $pr_0(\mathcal{W}'') \simeq 1.7465$ and $pr_1(\mathcal{W}'') \simeq 1.2535$. Such a string exists according to Theorem 4.6. According to Lemma 4.4 we get that $pr_0(\mathcal{W}') = \frac{1}{4}pr_0(\mathcal{W}'') + 1 \simeq 1.4366$ and that $pr_1(\mathcal{W}') = \frac{1}{2}pr_1(\mathcal{W}'') + 1 \simeq 1.6268$. As the string $\mathcal{W}'$ begins with 01, the hyperpairs constructed according to it are indeed hyper-products of P_{00}'s. We let H'_i denote the hyper-product generated according to the prefix of length i of $\mathcal{W}'$.

The generation of the hyper-products H'_i consumes P_{00}'s. Each such P_{00} carries a credit of $credit(P_{00}) = 0.1093$. As we have seen in the $(P_0, \gamma\times P_{00})$ subfactory, the pruning process may be viewed as a pruning process that receives $\frac{1}{2}\times P_{00}$ and returns either $\frac{1}{4}\times P_{11}$ below the center, or $\frac{1}{4}\times P_{00}$ above the center, as well as a leftover of $\frac{1}{2}\times P_0$ (or a P_λ which can be converted into $\frac{1}{2}\times P_0$ at no cost). The leftover $\frac{1}{2}\times P_0$ can be joined with $\frac{\gamma}{2}\times P_{00}$ from the input stream to form $\frac{1}{2}\times(P_0, \gamma\times P_{00})$. Such constructs can be recycled without any credit attached to them. The cost of pruning an element can therefore be "subsidized" by the credit attached to $\frac{1}{2}(1+\gamma)\times P_{00}$. The costs of the pruning process of the P_{00} subfactory are given in Table 8.4.

8.2.2. The output combinations of the P_{00} subfactory. The output combinations used by the P_{00} subfactory are given in Table 8.5. This list includes all the combinations that involve the outcomes Q_0^4, Q_2^6, and Q_6^{10} of the extended balanced P_{00} grafting and the outcomes R_4^0, R_3^1, R_6^2, R_{10}^6, and R_4^4 of the unbalanced P_{00} grafting. As can be seen, each of these outcomes can be used in conjunction with pruned elements to obtain low local lower and upper element costs. The balanced grafting process may also produce outcomes $Q_2^{1,1}$, Q_3^1, and Q_4^0, and the unbalanced grafting process may also produce outcomes $L_2^{1,1}$, L_2^6, and L_0^4. These outcomes are combined using the

combinations of the $(P_0, \gamma \times P_{00})$ subfactory given in Table 8.3. It is easy to verify that when each of the two grafting processes is applied a sufficient number of times, at least one output combination is applicable.

The costs of all the combinations involving $Q_2^{1,1}$, Q_3^1, and Q_4^0 and $L_2^{1,1}$, L_2^6, and L_0^4 are strictly smaller than their costs in the $(P_0, \gamma \times P_{00})$ subfactory. This is due to the fact that the input constructs to the P_{00} subfactory carry (positive) credits while the inputs to the $(P_0, \gamma \times P_{00})$ subfactory do not, and it is because the effective pruning costs $pr'_0(\mathcal{W}') \simeq 1.4366 - \frac{1+\gamma}{2}\, credit(P_{00}) \simeq 1.2366$ and $pr'_1(\mathcal{W}') \simeq 1.6268 - \frac{1+\gamma}{2}\, credit(P_{00}) \simeq 1.4267$ in the P_{00} subfactory are lower than the pruning costs $pr_0(\mathcal{W}) = pr_1(\mathcal{W}) = 1.5$ of the $(P_0, \gamma \times P_{00})$ subfactory.

It is therefore enough to analyze the costs of the combinations listed in Table 8.5. We present, as examples, the analyses of combinations 3 and 7 which, together with combinations 2, 3, and 7 of the $(P_0, \gamma \times P_{00})$ subfactory, determine the worst-case behavior of the whole factory. The analysis of all the other cases is similar.

- $(1. \times R_6^2, 3.5464 \times PR_1)$.

  The local lower and upper element costs of the combination $(1. \times R_6^2, r \times PR_1)$ are

$$u_0(C_3) = \frac{1. \times (gen(R_6^2) + rec_1(R_6^2)) + r \times (gen(PR_1) + rec_1(PR_1)) - (2 + \frac{r}{2}(1+\gamma))\, credit(P_{00})}{1. \times n_0(R_6^2) + r \times n_0(PR_1)},$$

$$u_1(C_3) = \frac{1. \times (gen(R_6^2) + rec_0(R_6^2)) + r \times (gen(PR_1) + rec_0(PR_1)) - (2 + \frac{r}{2}(1+\gamma))\, credit(P_{00})}{1. \times n_1(R_6^2) + r \times n_0(PR_1)},$$

  No credits should be attached to recycled elements, assuming that $\frac{r}{4} \leq \gamma$, since the proportion of quartets among the recycled elements is low enough. Solving the equation

$$\frac{1. \times (3+9) + r \times (1.6268 + 0.25) - (2 + \frac{r}{2}(1 + 2.6603)) \cdot 0.1093}{1. \times 6 + r \times 0} = \frac{1. \times (3+5) + r \times (1.6268 + 1) - (2 + \frac{r}{2}(1 + 2.6603)) \cdot 0.1093}{1. \times 2 + r \times 1}$$

  we get that $r \simeq 3.5464$ and $u_0(C_3) = u_1(C_3) \simeq 2.9546$.

- $(1. \times Q_2^6, 4.0000 \times PR_0)$.

  The local lower and upper element costs of the combination $(1. \times Q_2^6, r \times PR_0)$ are

$$u_0(C_7) = \frac{1. \times (gen(Q_2^6) + rec_1(Q_2^6)) + r \times (gen(PR_0) + rec_1(PR_0)) - (2 + \frac{r}{2}(1+\gamma))\, credit(P_{00})}{1. \times n_0(Q_2^6) + r \times n_0(PR_0)},$$

$$u_1(C_7) = \frac{1. \times (gen(Q_2^6) + rec_0(Q_2^6)) + r \times (gen(PR_0) + rec_0(PR_0)) - (2 + \frac{r}{2}(1+\gamma))\, credit(P_{00})}{1. \times n_1(Q_2^6) + r \times n_0(PR_0)}.$$

  Again, no credits should be attached to recycled elements, assuming that $\frac{r}{4} \leq \gamma$. Solving the equation

$$\frac{1. \times (4+5) + r \times (1.4366 + 1) - (2 + \frac{r}{2}(1 + 2.6603)) \cdot 0.1093}{1. \times 2 + r \times 1} = \frac{1. \times (4+8) + r \times (1.4366 + 0.25) - (2 + \frac{r}{2}(1 + 2.6603)) \cdot 0.1093}{1. \times 6 + r \times 0}$$

  we get that $r \simeq 4.0000$ and $u_0(C_7) = u_1(C_7) \simeq 2.9546$.

This completes the description and the analysis of the factory $\mathcal{G}'_k$.

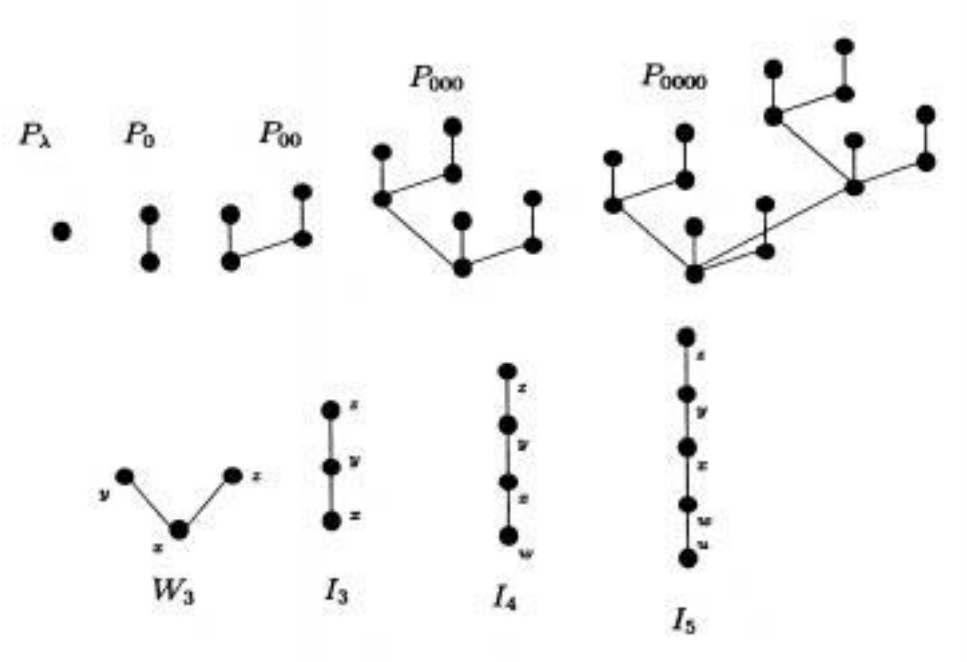


FIG. 9.1. *The basic input constructs of the factory* $\mathcal{G}_k$.

9. A factory with $u_0, u_1 \sim 2.942$. In this section we sketch the construction of factories $\mathcal{G}_k$ with lower and upper element costs $u_0, u_1 \sim 2.942$ whose existence was claimed in Theorem 2.3. A complete description of the factories $\mathcal{G}_k$ can be found in [Dor95].

The general structure of the factories $\mathcal{G}_k$ is similar to the structure of the factories $\mathcal{G}'_k$ described in the previous section. The $\mathcal{G}_k$ factories, however, recycle many more constructs. The structures recycled by the factories G_k are the structures shown in Figure 9.1 and their mirror images (compare this with the much smaller set of constructs recycled by the factories $\mathcal{G}'_k$, shown in Figure 8.1).

The factory $\mathcal{G}_k$ is composed of 13 subfactories: a $(P_0, \gamma \times P_{00})$ subfactory, a $(P_1, \gamma \times P_{11})$ subfactory, a P_{00} subfactory, a P_{11} subfactory, a P_{000} subfactory, a P_{111} subfactory, a P_{0000} subfactory, a P_{1111} subfactory, a $(P_\lambda, P_0, \beta \times P_{000})$ subfactory, a $(P_\lambda, P_1, \beta \times P_{111})$ subfactory, an I_3 subfactory, an I_4 subfactory, and an I_5 subfactory, where $\beta \simeq 1.6500$ and $\gamma \simeq 2.0500$.

The credits attached to the basic and compound constructs are $credit(P_\lambda) = 0$, $credit(P_0) = 0$, $credit(P_0, \gamma \times P_{00}) = 0$, $credit(P_{00}) = 0.1330$, $credit(P_{000}) = 0.8630$, $credit(P_{0000}) = 2.4847$, $credit(I_3) = -0.4213$, $credit(I_4) = -0.9230$, $credit(I_5) = -1.2000$, $credit(W_3) = 0.5000$, and $credit(P_\lambda, P_0, \beta \times P_{000}) = 1.2259$. The credit attached to a construct is equal to the credit attached to its mirror image.

The first four subfactories employed by $\mathcal{G}_k$ are essentially identical to the corresponding subfactories used by $\mathcal{G}'_k$. The main difference is that different parameters are used and that pruned elements are recycled as 16-tuples (i.e., P_{0000}'s or P_{1111}'s) and not as quartets. The value of the parameter γ is decreased to $\gamma \simeq 2.0500$. The credit attached to a quartet is increased to $credit(P_{00}) \simeq 0.1330$. These changes change the costs of these subfactories but the analysis is very similar to the one carried out in

the previous section. The other nine subfactories are new.

As can be seen, there is no W_3 subfactory. As before, a workshop is used to convert two W_3's into a P_0 and a P_{00} at the price of cutting a single edge.

The second output combination of the $(P_0, \gamma \times P_{00})$ subfactory is again one of the worst cases of the factories $\mathcal{G}_k$. We describe the analysis of this case and compare it with the analysis of the same case in the $\mathcal{G}'_k$ factories.

- $(1. \times U_0^2, 2.2613 \times PR_0)$.

  The local lower and upper element costs of the combination $(1. \times U_0^2, r \times PR_0)$ are

$$u_0(C_2) = \frac{1. \times (gen(U_0^2) + rec_1(U_0^2)) + r \times (gen(PR_0) + rec_1(PR_0)) + credit(P_0)}{1. \times n_0(U_0^2) + r \times n_0(PR_0)},$$

$$u_1(C_2) = \frac{1. \times (gen(U_0^2) + rec_0(U_0^2)) + r \times (gen(PR_0) + rec_0(PR_0)) + \frac{r}{16} \times credit(P_{0000})}{1. \times n_1(U_0^2) + r \times n_1(PR_0)}.$$

  The expression for $u_0(C_2)$ is identical to the expression for $u_0(C_2)$ in $\mathcal{G}'_k$. The expression for $u_1(C_2)$ differs from the corresponding expression in $\mathcal{G}'_k$ as pruned elements are now recycled as P_{1111}'s. The recycling costs of pruned elements are now $rec_0(PR_0) = 0.0625$ and $rec_1(PR_0) = 1$. Each recycled P_{1111} should carry a credit of $credit(P_{0000}) \simeq 2.4847$. Substituting these updated values into the above expressions and equating the two costs we obtain the equation

$$\frac{1. \times (0 + 1) + r \times (1.5 + 1)}{1. \times 0 + r \times 1} = \frac{1. \times (0 + 2) + r \times (1.5 + 0.0625) + \frac{r}{16} \times 2.4847}{1. \times 2 + r \times 0}.$$

  It is easy to check that the solution of this equation is $r \simeq 2.2613$ and the values of both the lower and upper element costs in this case are $u_0(C_2) = u_1(C_2) \simeq 2.9422$.

We believe that it is possible to obtain further *small* improvements by recycling more and yet larger constructs and by designing a new subfactory for each such construct or combination of constructs. The $(P_0, \gamma \times P_{00})$ subfactory serves as a keystone in all our factories and in all such possible extensions. Recall that one of the worst cases of the $(P_0, \gamma \times P_{00})$ subfactory is the combination of U_0^2 and pruning. The lower part of an output partial order generated using this combination is essentially a hyperpair of the form $P_{1^{k'}}$, where $k \leq k' \leq 2k$. It is not hard to verify that even if we could recycle such a large hyperpair without attaching any credit to it, the lower and upper element costs of this combination would still be about $u_0(C_2) = u_1(C_2) \simeq 2.895$. It seems, therefore, that some new ideas are required to obtain a major improvement to our median selection algorithm.

10. Concluding remarks and open problems. We have improved the result of Schönhage, Paterson, and Pippenger [SPP76] and obtained an algorithm for the selection of the median that uses slightly less than $3n$ comparisons. Our algorithm is much more complicated than the algorithm of Schönhage, Paterson, and Pippenger, and it is perhaps a bit disappointing that the improvement obtained is so small. As mentioned at the end of the previous section, further small improvements are possible but it seems that new ideas are required to obtain a more substantial improvement.

Further narrowing the gap between the known upper and lower bounds on the number of comparisons needed to select the median remains a challenging open problem. Paterson has conjectured, for a long time, that the number of comparisons required for selecting the median, in the worst case, is about $\log_{4/3} 2 \cdot n \approx 2.41n$. This conjecture has now finally appeared in print; see Paterson [Pat96].

Schönhage, Paterson, and Pippenger [SPP76] show that a conjecture of Yao [Yao74] implies the existence of a median-finding algorithm that uses at most $2.5n + o(n)$ comparisons. Proving or disproving Yao's conjecture is also a challenging open problem.

The factories constructed in this paper are $\tilde{\mathcal{S}}_k^k$ factories and not S_k^k factories, as were the factories of Schönhage, Paterson, and Pippenger. Is it possible to improve the S_k^k factories of Schönhage, Paterson, and Pippenger and obtain S_k^k factories whose lower and upper element costs are below 3?

Schönhage, Paterson, and Pippenger [SPP76] describe nongreen S_k^k factories with unit cost $U_k \sim 3.5k$. Is it possible to improve this result?

Acknowledgment. The authors would like to thank Mike Paterson for some helpful discussions and for his comments on an earlier version of this paper.

REFERENCES

[Aig82] M. Aigner, *Selecting the top three elements*, Discrete Appl. Math., 4 (1982), pp. 247–267.

[BFP+73] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan, *Time bounds for selection*, J. Comput. System Sci., 7 (1973), pp. 448–461.

[BJ85] S. W. Bent and J. W. John, *Finding the median requires 2n comparisons*, in Proceedings of the 17th Annual ACM Symposium on Theory of Computing, Providence, RI, 1985, pp. 213–216.

[CM89] W. Cunto and J. I. Munro, *Average case selection*, J. ACM, 36 (1989), pp. 270–279.

[DHUZ96] D. Dor, J. Håstad, S. Ulfberg, and U. Zwick, *On lower bounds for selecting the median*, submitted.

[Dor95] D. Dor, *Selection Algorithms*, Ph.D. thesis, Department of Computer Science, Tel Aviv University, Tel Aviv, Israel, 1995.

[DZ95] D. Dor and U. Zwick, *Selecting the median*, in Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, January 22–24, 1995, SIAM, Philadelphia, 1995, pp. 28–37.

[DZ96a] D. Dor and U. Zwick, *Finding the αn-th largest element*, Combinatorica, 16 (1996), pp. 41–58.

[DZ96b] D. Dor and U. Zwick, *Median selection requires* $(2+\epsilon)n$ *comparisons*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Burlington, VT, 1996, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 125–134.

[Eus93] J. Eusterbrock, *Errata to selecting the top three elements by M. Aigner: A result of a computer-assisted proof search*, Discrete Appl. Math., 41 (1993), pp. 131–137.

[FG78] F. Fussenegger and H. N. Gabow, *A counting approach to lower bounds for selection problems*, J. ACM, 26 (1978), pp. 227–238.

[FJ59] L. R. Ford and S. M. Johnson, *A tournament problem*, Amer. Math. Monthly, 66 (1959), pp. 387–389.

[FR75] R. W. Floyd and R. L. Rivest, *Expected time bounds for selection*, Comm. ACM, 18 (1975), pp. 165–173.

[HS69] A. Hadian and M. Sobel, *Selecting the t-th largest using binary errorless comparisons*, in Combinatorial Theory and Its Applications, II, Proc. Colloq., Balatonfüred, 1969, North-Holland, Amsterdam, 1970, pp. 585–599.

[Hya76] L. Hyafil, *Bounds for selection*, SIAM J. Comput., 5 (1976), pp. 109–114.

[Joh88] J. W. John, *A new lower bound for the set-partitioning problem*, SIAM J. Comput., 17 (1988), pp. 640–647.

[Kir81] D. G. Kirkpatrick, *A unified lower bound for selection and set partitioning problems*, J. ACM, 28 (1981), pp. 150–165.

[Kis64] S. S. Kislitsyn, *On the selection of the k-th element of an ordered set by pairwise comparisons*, Sibirsk. Mat. Zh., 5 (1964), pp. 557–564.

[MP82] I. Munro and P. V. Poblete, *A Lower Bound for Determining the Median*, Technical report CS-82-21, University of Waterloo, Waterloo, Ontario, 1982.

[Pat96] M. S. Paterson, *Progress in selection*, in Proceedings of the 5th Scandinavian Workshop on Algorithm Theory, Reykjavík, Iceland, 1996, Lecture Notes in Comput. Sci. 1097, Springer, Berlin, 1996, pp. 368–379.

[Poh72] I. Pohl, *A sorting problem and its complexity*, Comm. ACM, 15 (1972), pp. 462–464.

[RH84] P. V. Ramanan and L. Hyafil, *New algorithms for selection*, J. Algorithms, 5 (1984), pp. 557–578.

[Sch32] J. Schreier, *On tournament elimination systems*, Math. Polska, 7 (1932), pp. 154–160 (in Polish).

[SPP76] A. Schönhage, M. Paterson, and N. Pippenger, *Finding the median*, J. Comput. System Sci., 13 (1976), pp. 184–199.

[SY80] P. K. R. Stockmeyer and F. F. Yao, *On the optimality of linear merge*, SIAM J. Comput., 9 (1980), pp. 85–90.

[Yao74] F. Yao, *On Lower Bounds for Selection Problems*, Technical report MAC TR-121, Massachusetts Institute of Technology, Cambridge, MA, 1974.

[Yap76] C. K. Yap, *New upper bounds for selection*, Comm. ACM, 19 (1976), pp. 501–508.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1759–1782

STRUCTURE IN APPROXIMATION CLASSES*

PIERLUIGI CRESCENZI†, VIGGO KANN‡, RICCARDO SILVESTRI§, AND LUCA TREVISAN¶

Abstract. The study of the approximability properties of NP-hard optimization problems has recently made great advances mainly due to the results obtained in the field of proof checking. The last important breakthrough proves the APX-completeness of several important optimization problems and thus reconciles "two distinct views of approximation classes: *syntactic* and *computational*" [S. Khanna et al., in *Proc. 35th IEEE Symp. on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 819–830]. In this paper we obtain new results on the structure of several computationally-defined approximation classes. In particular, after defining a new approximation preserving reducibility to be used for as many approximation classes as possible, we give the first examples of natural NPO-complete problems and the first examples of natural APX-intermediate problems. Moreover, we state new connections between the approximability properties and the query complexity of NPO problems.

Key words. complexity classes, reducibilities, approximation algorithms

AMS subject classifications. 03D30, 68Q15, 68Q20

PII. S0097539796304220

1. Introduction.

In his pioneering paper on the approximation of combinatorial optimization problems [22], David Johnson formally introduced the notion of an approximable problem, proposed approximation algorithms for several problems, and suggested a possible classification of optimization problems on the grounds of their approximability properties. Since then it has been clear that, even though the decision versions of most NP-hard optimization problems are many-one polynomial-time reducible to each other, they do not share the same approximability properties. The main reason is that many-one reductions usually do not preserve the objective function and, even when they do, they rarely preserve the quality of the solutions. It is then clear that a stronger kind of reducibility has to be used. Indeed, an approximation preserving reduction not only has to map instances of a problem A to instances of a problem B, but it also has to be able to come back from "good" solutions for B to "good" solutions for A. Surprisingly, the first definition of this kind of reducibility [35] was given a full 13 years after Johnson's paper; after that, at least seven different approximation preserving reducibilities appeared in the literature (see Fig. 1.1). These reducibilities are identical with respect to the overall scheme but differ essentially in the way they preserve approximability: they range from the Strict reducibility in which the error cannot increase to the PTAS-reducibility in which there are basically no restrictions (see also Chapter 3 of [25] and [11]).

*Received by the editors May 24, 1996; accepted for publication (in revised form) January 7, 1998; published electronically May 13, 1999. An extended abstract of this paper was presented at the 1st Annual International Computing and Combinatorics Conference.

http://www.siam.org/journals/sicomp/28-5/30422.html

†Dipartimento di Sistemi ed Informatica, Università degli Studi di Firenze, 50134 Firenze, Italy (piluc@dsi.unifi.it).

‡Department of Numerical Analysis and Computing Science, Royal Institute of Technology, S-100 44 Stockholm, Sweden (viggo@nada.kth.se).

§Dipartimento di Scienze dell'Informazione, Università degli Studi di Roma "La Sapienza," 00198 Rome, Italy (silver@dsi.uniroma1.it).

¶Laboratory for Computer Science, MIT, Cambridge, MA 02139 (luca@theory.lcs.mit.edu).

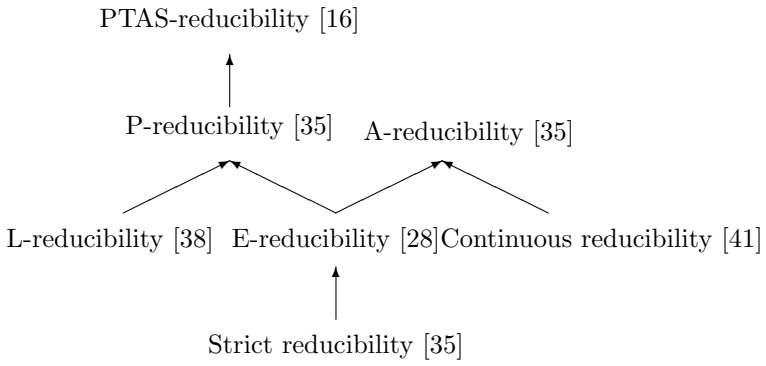


FIG. 1.1. *The taxonomy of approximation preserving reducibilities.*

By means of these reducibilities, several notions of completeness in approximation classes have been introduced and, basically, two different approaches have been followed. On one hand, the attention has been focused on computationally defined classes of problems, such as NPO (the class of optimization problems whose underlying decision problem is in NP) and APX (the class of constant-factor approximable NPO problems): along this line of research, however, almost all completeness results dealt either with artificial optimization problems or with problems for which lower bounds on the quality of the approximation were easily obtainable [14, 35]. On the other hand, researchers focused on the logical definability of optimization problems and introduced several syntactically defined classes for which natural completeness results were obtained [29, 36, 38]: unfortunately, the approximability properties of the problems in these latter classes were not related to standard complexity-theoretic conjectures. A first step towards the reconciling of these two approaches consisted of proving lower bounds (modulo $P \neq NP$ or some other likely condition) on the approximability of complete problems for syntactically defined classes [1, 33]. More recently, another step has been performed since the closure of syntactically defined classes with respect to an approximation preserving reducibility which has been proved to be equal to the more familiar computationally-defined classes [28].

In spite of this important achievement, beyond APX we are still forced to distinguish between maximization and minimization problems as long as we are interested in completeness proofs. Indeed, a result of [29] states that it is not possible to rewrite every NP maximization problem as an NP minimization problem unless NP = *co*-NP. A natural question is thus whether this duality extends to approximation preserving reductions.

Finally, even though the existence of "intermediate" artificial problems—that is, problems for which lower bounds on their approximation are not obtainable by completeness results—was proved in [14], a natural question arises: do natural intermediate problems exist? Observe that this question is also open in the field of decision problems: for example, it is known that the graph isomorphism problem cannot be NP-complete unless the polynomial-time hierarchy collapses [40], but no result has ever been obtained giving evidence that the problem does not belong to P.

The first goal of this paper is to define an approximation preserving reducibility that can be used for as many approximation classes as possible and such that all

reductions that have appeared in the literature still hold. In spite of the fact that the L-reducibility has been the most widely used so far, we will give strong evidence that it cannot be used to obtain completeness results in "computationally defined" classes such as APX, log-APX (that is, the class of problems approximable within a logarithmic factor), and poly-APX (that is, the class of problems approximable within a polynomial factor). Indeed, in [16] it has been shown that the L-reducibility is too strict and does not allow reduction of some problems that are known to be easy to approximate to problems that are known to be hard to approximate. In this paper we show, somewhat surprisingly, that the L-reducibility is too weak and is not approximation preserving (unless $\mathrm{P} = \mathrm{NP} \cap \textit{co}\text{-NP}$). The weakness of the L-reducibility is, essentially, shared by all reducibilities of Fig. 1.1 except the Strict reducibility and the E-reducibility; the strictness of the L-reducibility is shared by all of them (unless $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{NP}[O(\log n)]}$) except the PTAS-reducibility. The reducibility we propose is *a combination of the E-reducibility and of the PTAS-reducibility*. As far as we know, it is the strictest reducibility that allows us to obtain all approximation completeness results that have appeared in the literature, such as the APX-completeness of Maximum Satisfiability [16, 28] and the poly-APX-completeness of Maximum Clique [28].

The second group of results refers to the existence of natural complete problems for NPO. Indeed, both [35] and [14] provide examples of natural complete problems for the class of minimization and maximization NP problems, respectively. In section 3 we will show *the existence of both maximization and minimization* NPO*-complete natural problems*. In particular, we prove that Maximum $0-1$ Programming and Minimum $0-1$ Programming are NPO-complete. This result shows that making use of a natural approximation preserving reducibility is powerful enough to encompass the "duality" problem raised in [29]. (Indeed, in [28] it was shown that this duality does not arise in APX, log-APX, poly-APX, and other subclasses of NPO.) Moreover, the same result can also be obtained when we restrict ourselves to the class NPO PB (i.e., the class of polynomially bounded NPO problems). In particular, we prove that Maximum PB $0-1$ Programming and Minimum PB $0-1$ Programming are NPO PB-complete.

The third group of results refers to the existence of natural APX-intermediate problems. In section 4, we will prove that Minimum Bin Packing (and other natural NPO problems) cannot be APX-complete unless the polynomial-time hierarchy collapses. Since it is well known [34] that this problem belongs to APX and that it does not belong to PTAS (that is, the class of NPO problems with polynomial-time approximation schemes) unless $\mathrm{P} = \mathrm{NP}$, our result yields *the first example of a natural* APX*-intermediate problem* (under a natural complexity-theoretic conjecture). Roughly speaking, the proof of our result is structured into two main steps. In the first step, we show that if Minimum Bin Packing were APX-complete then the problem of answering any set of k nonadaptive queries to an NP-complete problem could be reduced to the problem of approximating an instance of Minimum Bin Packing within a ratio depending on k. In the second step, we show that the problem of approximating an instance of Minimum Bin Packing within a given performance ratio can be solved in polynomial time by means of a constant number of nonadaptive queries to an NP-complete problem. These two steps will imply the collapse of the query hierarchy which in turn implies the collapse of the polynomial-time hierarchy. As a side effect of our proof, we will show that *if a problem is* APX*-complete, then it does not admit an asymptotic approximation scheme*.

The previous results are consequences of new connections between the approximability properties and the query complexity of NP-hard optimization problems. In several recent papers the notion of query complexity (that is, the number of queries to an NP oracle needed to solve a given problem) has been shown to be a very useful tool for understanding the complexity of approximation problems. In [7, 9] upper and lower bounds have been proved on the number of queries needed to approximate certain optimization problems (such as MAXIMUM SATISFIABILITY and MAXIMUM CLIQUE): these results deal with the complexity of approximating the value of the optimum solution and not with the complexity of computing approximate solutions. In this paper, instead, the complexity of "constructive" approximation will be addressed by considering the languages that can be recognized by polynomial-time machines which have a function oracle that solves the approximation problem. In particular, after proving the existence of natural APX-intermediate problems, in section 4.1 we will be able to solve an open question of [7], proving that *finding the vertices of the largest clique is more difficult than merely finding the vertices of a* 2*-approximate clique* unless the polynomial-time hierarchy collapses.

The results of [7, 9] show that the query complexity is a good measure to study approximability properties of optimization problems. The last group of our results show that completeness in approximation classes implies lower bounds on the query complexity. Indeed, in section 5 we show that the two approaches are basically equivalent by giving *sufficient and necessary conditions for approximation completeness in terms of query-complexity hardness and combinatorial properties.* These results have a twofold importance: they give new insights into the structure of complete problems for approximation classes and they reconcile the approach based on standard computation models with the approach based on the computation model for approximation proposed in [8]. As a final observation, our results can be seen as extensions of a result of [28] in which general sufficient (but not necessary) conditions for APX-completeness are proved.

1.1. Preliminaries. We assume the reader is familiar with the basic concepts of computational complexity theory. For the definitions of most of the complexity classes used in the paper we refer the reader to one of the books on the subject (see, for example, [2, 5, 18, 37]). We now give some standard definitions in the field of optimization and approximation theory.

DEFINITION 1.1. *An* NP optimization problem A *is a* 4*-tuple* $(I, sol, m, type)$ *such that the following hold:*

1. I *is the set of the* instances *of* A *and it is recognizable in polynomial time.*
2. *Given an instance* x *of* I, $sol(x)$ *denotes the set of* feasible solutions *of* x. *These solutions are short, that is, there exists a polynomial* p *such that, for any* $y \in sol(x)$, $|y| \leq p(|x|)$. *Moreover, for any* x *and for any* y *with* $|y| \leq p(|x|)$, *it is decidable in polynomial time whether* $y \in sol(x)$.
3. *Given an instance* x *and a feasible solution* y *of* x, $m(x, y)$ *denotes the positive integer* measure *of* y *(often also called the value of* y*). The function* m *is computable in polynomial time and is also called the* objective *function.*
4. $type \in \{\max, \min\}$.

The goal of an NP optimization problem with respect to an instance x is to find an *optimum solution*, that is, a feasible solution y such that $m(x, y) = type\{m(x, y') : y' \in sol(x)\}$. In the following opt will denote the function mapping an instance x to the measure of an optimum solution.

The *class* NPO is the set of all NP optimization problems. Max NPO is the set of

maximization NPO problems and Min NPO is the set of minimization NPO problems.

An NPO problem is said to be *polynomially bounded* if there exists a polynomial q such that, for any instance x and for any solution y of x, $m(x,y) \leq q(|x|)$. The *class* NPO PB is the set of all polynomially bounded NPO problems. Max PB is the set of all maximization problems in NPO PB and Min PB is the set of all minimization problems in NPO PB.

DEFINITION 1.2. *Let A be an* NPO *problem. Given an instance x and a feasible solution y of x, we define the* performance ratio of y with respect to x *as*

$$R(x,y) = \max\left\{\frac{m(x,y)}{\mathsf{opt}(x)}, \frac{\mathsf{opt}(x)}{m(x,y)}\right\}$$

and the relative error of y with respect to x *as*

$$E(x,y) = \frac{|\mathsf{opt}(x) - m(x,y)|}{\mathsf{opt}(x)}.$$

It is easy to see that, for any instance x and for any feasible solution y of x,

$$R(x,y) - 1 \geq E(x,y) \geq 0.$$

Also, the performance ratio $R(x,y)$ (respectively, relative error $E(x,y)$) is as close to 1 (respectively, 0) as the value of the feasible solution y is close to the value of an optimum solution for x.

DEFINITION 1.3. *Given an* NPO *problem A and an arbitrary function $r : N \to [1,\infty)$, we say that an algorithm T is an* $r(n)$-approximate algorithm for A *if T runs in polynomial time and, for any instance x of A with $sol(x) \neq \emptyset$, T returns a feasible solution $T(x)$ such that*

$$R(x,T(x)) \leq r(|x|).$$

DEFINITION 1.4. *Given a class of functions F, an* NPO *problem A belongs to the* class F-APX *if there exists an $r(n)$-approximate algorithm T for A, for some function $r \in F$. In particular,* APX, *log-*APX, *poly-*APX, *and exp-*APX *will denote the classes F-*APX *with F equal to the set $O(1)$, to the set $O(\log n)$, to the set $O(n^{O(1)})$, and to the set $O(2^{n^{O(1)}})$, respectively.*

One could object that there is no difference between NPO and exp-APX since the polynomial bound on the computation time of the objective function implies that any NPO problem is $h2^{n^k}$-approximable for some h and k. This is not true, since NPO problems exist for which it is even hard to find a feasible solution. We will see examples of such problems in section 3.

DEFINITION 1.5. *An* NPO *problem A belongs to the* class PTAS *if there exists an algorithm T such that, for any fixed rational $r > 1$, $T(\cdot, r)$ is an r-approximate algorithm for A.*

Clearly, the following inclusions hold:

$$\text{PTAS} \subseteq \text{APX} \subseteq \text{log-APX} \subseteq \text{poly-APX} \subseteq \text{exp-APX} \subseteq \text{NPO}.$$

It is also easy to see that these inclusions are strict if and only if P $\neq$ NP.

1.2. A list of NPO problems. We here define the NP optimization problems that will be used in the paper (for a much larger list of NPO problems we refer to [12, 13]). Observe that in the following definitions we will not mention the type of the problem since it will be specified by the name of the problem itself.

MAXIMUM CLIQUE
INSTANCE: Graph $G = (V, E)$.
SOLUTION: A clique in G, i.e., a subset $V' \subseteq V$ such that every two vertices in V' are joined by an edge in E.
MEASURE: Cardinality of the clique, i.e., $|V'|$.

MAXIMUM WEIGHTED ONES and MINIMUM WEIGHTED ONES
INSTANCE: Set of variables X, Boolean quantifier-free first-order formula ϕ over the variables in X, and a weight function $w : X \to N$.
SOLUTION: Truth assignment that satisfies ϕ.
MEASURE: The sum of the weights of the true variables.

MAXIMUM PB $0-1$ PROGRAMMING and MINIMUM PB $0-1$ PROGRAMMING
INSTANCE: Integer $m \times n$-matrix A, integer m-vector b, binary n-vector c.
SOLUTION: A binary n-vector x such that $Ax \geq b$.
MEASURE: $1 + \sum_{i=1}^{n} c_i x_i$.

MAXIMUM SATISFIABILITY
INSTANCE: Set of variables X and Boolean CNF formula ϕ over the variables in X.
SOLUTION: Truth assignment to the variables in X.
MEASURE: The number of satisfied clauses.

MINIMUM BIN PACKING
INSTANCE: Finite set U of items, and a size $s(u) \in Q \cap (0, 1]$ for each $u \in U$.
SOLUTION: A partition of U into disjoint sets $U_1, U_2, \ldots, U_m$ such that the sum of the sizes of the items in each U_i is at most 1.
MEASURE: The number of used bins, i.e., the number m of disjoint sets.

MINIMUM ORDERED BIN PACKING
INSTANCE: Finite set U of items, a size $s(u) \in Q \cap (0, 1]$ for each $u \in U$, and a partial order $\preceq$ on U.
SOLUTION: A partition of U into disjoint sets $U_1, U_2, \ldots, U_m$ such that the sum of the sizes of the items in each U_i is at most 1 and if $u \in U_i$ and $u' \in U_j$ with $u \preceq u'$, then $i \leq j$.
MEASURE: The number of used bins, i.e., the number m of disjoint sets.

MINIMUM DEGREE SPANNING TREE
INSTANCE: Graph $G = (V, E)$.
SOLUTION: A spanning tree for G.
MEASURE: The maximum degree of the spanning tree.

MINIMUM EDGE COLORING
INSTANCE: Graph $G = (V, E)$.
SOLUTION: A coloring of E, i.e., a partition of E into disjoint sets $E_1, E_2, \ldots, E_k$ such that, for $1 \leq i \leq k$, no two edges in E_i share a common endpoint in G.
MEASURE: Cardinality of the coloring, i.e., the number k of disjoint sets.

2. A new approximation preserving reducibility. The goal of this section is to define a new approximation preserving reducibility that can be used for as many approximation classes as possible and such that all reductions that have appeared in the literature still hold. We will justify the definition of this new reducibility by emphasizing the disadvantages of previously known ones. In the following, we will assume that, for any reducibility, an instance x such that $sol(x) \neq \emptyset$ is mapped into an instance x' such that $sol(x') \neq \emptyset$.

2.1. The L-reducibility. The first reducibility we shall consider is the L-reducibility (for *linear* reducibility) [38] which is often most practical to use in order to show that a problem is at least as hard to approximate as another.

DEFINITION 2.1. *Let A and B be two* NPO *problems. A is said to be* L-reducible *to B, in symbols $A \leq_{\mathrm{L}} B$, if there exist two functions f and g and two positive constants α and β such that:*

1. *For any $x \in I_A$, $f(x) \in I_B$ is computable in polynomial time.*
2. *For any $x \in I_A$ and for any $y \in sol_B(f(x))$, $g(x,y) \in sol_A(x)$ is computable in polynomial time.*
3. *For any $x \in I_A$, $\mathsf{opt}_B(f(x)) \leq \alpha \mathsf{opt}_A(x)$.*
4. *For any $x \in I_A$ and for any $y \in sol_B(f(x))$,*

$$|\mathsf{opt}_A(x) - m_A(x, g(x,y))| \leq \beta |\mathsf{opt}_B(f(x)) - m_B(f(x), y)|.$$

The 4-tuple (f, g, α, β) is said to be an L-reduction from A to B.

Clearly, the L-reducibility preserves membership in PTAS. Indeed, if (f, g, α, β) is an L-reduction from A to B then, for any $x \in I_A$ and for any $y \in sol_B(f(x))$, we have that

$$E_A(x, g(x,y)) \leq \alpha\beta E_B(f(x), y),$$

so that if $B \in$ PTAS then $A \in$ PTAS [38]. The above inequality also implies that if A is a minimization problem and there exists an r-approximate algorithm for B, then there exists a $(1 + \alpha\beta(r-1))$-approximate algorithm for A. In other words, L-reductions from minimization problems to optimization problems preserve membership in APX. The next result gives strong evidence that, in general, this is not true whenever the starting problem is a maximization one.

THEOREM 2.2. *The following statements are equivalent:*

1. *There exist two problems $A \in$* Max NPO *and $B \in$* Min NPO *such that $A \notin$* APX*, $B \in$* APX*, and $A \leq_{\mathrm{L}} B$.*
2. *There exist two* Max NPO *problems A and B such that $A \notin$* APX*, $B \in$* APX*, and $A \leq_{\mathrm{L}} B$.*
3. *There exists a polynomial-time recognizable set of satisfiable Boolean formulas for which no polynomial-time algorithm can compute a satisfying assignment for each of them.*

Proof. (1) $\Rightarrow$ (2). Assume that there exist two problems $A \in$ Max NPO and $B \in$ Min NPO such that $A \notin$ APX, $B \in$ APX, and $A \leq_{\mathrm{L}} B$. In section 3.1 of [28], it is shown that, for any minimization problem C in APX, there exists a maximization problem $C_{\max}$ in APX such that $C \leq_{\mathrm{L}} C_{\max}$. Since the L-reducibility clearly satisfies the transitive property, we thus have that there exists a maximization problem $B_{\max} \in$ APX such that $A \leq_{\mathrm{L}} B_{\max}$.

(2) $\Rightarrow$ (3). Assume that for any polynomial-time recognizable set of satisfiable Boolean formulas there is a polynomial-time algorithm computing a satisfying assignment for each formula in the set. Suppose that (f, g, α, β) is an L-reduction

from a maximization problem A to a maximization problem B and that B is r-approximable for some $r > 1$. Let x be an instance of A and let y be a solution of $f(x)$ such that $\mathsf{opt}_B(f(x))/m_B(f(x), y) \leq r$. For the sake of convenience, let $\mathsf{opt}_A = \mathsf{opt}_A(x)$, $m_A = m_A(x, g(x, y))$, $\mathsf{opt}_B = \mathsf{opt}_B(f(x))$, and $m_B = m_B(f(x), y)$. Let also $m_x = \max\{m_A, m_B/\alpha\}$. Since $m_A \leq \mathsf{opt}_A$ and $m_B/\alpha \leq \mathsf{opt}_B/\alpha \leq \mathsf{opt}_A$, we have that $m_x \leq \mathsf{opt}_A$. We now show that $\mathsf{opt}_A/m_x \leq 1 + \alpha\beta(r-1)$, that is, m_x is a nonconstructive approximation of opt_A. Let $\gamma = \frac{\alpha r}{1+\alpha\beta(r-1)}$. There are two cases.

1. $\mathsf{opt}_B \leq \gamma \mathsf{opt}_A$. By the definition of the L-reducibility, $\mathsf{opt}_A - m_A \leq \beta(\mathsf{opt}_B - m_B)$. Since $\mathsf{opt}_B \leq \gamma\mathsf{opt}_A$ and $\mathsf{opt}_B/m_B \leq r$, we have that

$$\frac{\mathsf{opt}_A - m_A}{\mathsf{opt}_A} \leq \gamma\beta\frac{\mathsf{opt}_B - m_B}{\mathsf{opt}_B} \leq \gamma\beta(1 - 1/r).$$

Hence,

$$\frac{\mathsf{opt}_A}{m_x} \leq \frac{\mathsf{opt}_A}{m_A} \leq \frac{1}{1 - \gamma\beta\frac{r-1}{r}} = 1 + \alpha\beta(r-1),$$

where the last equality is due to the definition of γ.

2. $\mathsf{opt}_B > \gamma\mathsf{opt}_A$. It holds that

$$\begin{aligned}
\frac{\mathsf{opt}_A}{m_x} &\leq \frac{\mathsf{opt}_A}{m_B/\alpha} \\
&< \frac{\alpha(\mathsf{opt}_B/\gamma)}{m_B} \text{ (since } \mathsf{opt}_A < \mathsf{opt}_B/\gamma) \\
&\leq \frac{\alpha(\mathsf{opt}_B/\gamma)}{(\mathsf{opt}_B/r)} \text{ (since } m_B \geq \mathsf{opt}_B/r) \\
&= \frac{\alpha r}{\gamma} \\
&= 1 + \alpha\beta(r-1).
\end{aligned}$$

Let us now consider the following nondeterministic polynomial-time algorithm.

begin {input: $x \in I_A$}
  x_B := $f(x)$;
  y_B := r-approximate solution of x_B;
  y_A := $g(x, y_B)$;
  m_x := $\max\{m_A(x, y_A), m_B(x_B, y_B)/\alpha\}$;
  guess $y \in sol_A(x)$;
  if $m_A(x, y) \geq m_x$ **then** accept **else** reject;
end;

By applying Cook's reduction [10] to the above algorithm, it easily follows that, for any $x \in I_A$, a satisfiable Boolean formula ϕ_x can be constructed in polynomial time in the length of x so that any satisfying assignment for ϕ_x encodes a solution of x whose measure is at least m_x. Moreover, the set $\{\phi_x : x \in I_A\}$ is recognizable in polynomial time. By assumption, it is then possible to compute in polynomial time a satisfying assignment for ϕ_x and thus an approximate solution for x. This contradicts the assumption that $A \notin$ APX.

(3) $\Rightarrow$ (1). Assume that there exists a polynomial-time recognizable set S of satisfiable Boolean formulas for which no polynomial-time algorithm can compute a satisfying assignment for each of them. Consider the following two NPO problems

$A = (I_A, sol_A, m_A, \max)$ and $B = (I_B, sol_B, m_B, \min)$, where $I_A = I_B = S$, $sol_A(x) = sol_B(x) = \{y : y \text{ is a truth assignment to the variables of } x\}$:

$$m_A(x,y) = \begin{cases} |x| & \text{if } y \text{ is a satisfying assignment for } x, \\ 1 & \text{otherwise} \end{cases}$$

and

$$m_B(x,y) = \begin{cases} |x| & \text{if } y \text{ is a satisfying assignment for } x, \\ 2|x| & \text{otherwise.} \end{cases}$$

Clearly, problem B is in APX; if A is in APX, then there is a polynomial-time algorithm that computes a satisfying assignment for each formula in S, contradicting the assumption. Moreover, it is easy to see that A L-reduces to B via $f \equiv \lambda x.x$, $g \equiv \lambda x \lambda y.y$, $\alpha = 1$, and $\beta = 1$. □

Observe that in [32] it is shown that the third statement of the above theorem holds if and only if the γ-reducibility is different from the many-one reducibility. Moreover, in [21] it is shown that the latter hypothesis is somewhat intermediate between P $\neq$ NP $\cap$ *co*-NP and P $\neq$ NP. In other words, there is strong evidence that, even though the L-reducibility is suitable for proving completeness results within classes contained in APX (such as Max SNP [38]), this reducibility cannot be used to define the notion of completeness for classes beyond APX. Moreover, it cannot be blindly used to obtain positive results, that is, to prove the existence of approximation algorithms via reductions. Finally, it is possible to L-reduce the maximization problem B defined in the last part of the proof of the previous theorem to Maximum 3-Satisfiability: this implies that the closure of Max SNP with respect to the L-reducibility is not included in APX, contrary to what is commonly believed (see, for example, [37, p. 314]).

2.2. The E-reducibility. The drawbacks of the L-reducibility are mainly due to the fact that the relation between the performance ratios is set by two separate linear constraints on both the optimum values and the absolute errors. The E-reducibility (for *error* reducibility) [28] instead imposes a linear relation directly between the performance ratios.

Definition 2.3. *Let A and B be two* NPO *problems. A is said to be* E-reducible *to B, in symbols $A \leq_{\mathrm{E}} B$, if there exist two functions f and g and a positive constant α such that the following hold:*

1. *For any $x \in I_A$, $f(x) \in I_B$ is computable in polynomial time.*
2. *For any $x \in I_A$ and for any $y \in sol_B(f(x))$, $g(x,y) \in sol_A(x)$ is computable in polynomial time.*
3. *For any $x \in I_A$ and for any $y \in sol_B(f(x))$,*

$$R_A(x, g(x,y)) \leq 1 + \alpha(R_B(f(x), y) - 1).$$

The 3-tuple (f, g, α) is said to be an E-reduction from A to B.

Observe that, for any function r, an E-reduction maps $r(n)$-approximate solutions into $(1+\alpha(r(n^h)-1))$-approximate solutions, where h is a constant depending only on the reduction. Hence, the E-reducibility not only preserves membership in PTAS but also membership in exp-APX, poly-APX, log-APX, and APX. As a consequence of this observation and of the results of the previous section, we have that NPO problems exist which are L-reducible to each other but not E-reducible. However, the following

result shows that within the class APX the E-reducibility is just a generalization of the L-reducibility.

PROPOSITION 2.4. *For any two* NPO *problems* A *and* B, *if* $A \leq_{\mathrm{L}} B$ *and* $A \in$ APX, *then* $A \leq_{\mathrm{E}} B$.

Proof. Let T be an r-approximate algorithm for A with r constant and let $(f_L, g_L, \alpha_L, \beta_L)$ be an L-reduction from A to B. Then, for any $x \in I_A$ and for any $y \in sol_B(f_L(x))$, $E_A(x, g_L(x,y)) \leq \alpha_L \beta_L E_B(f_L(x), y)$. If A is a minimization problem then, for any $x \in I_A$ and for any $y \in sol_B(f_L(x))$,

$$\begin{aligned} R_A(x, g_L(x,y)) &= 1 + E_A(x, g_L(x,y)) \\ &\leq 1 + \alpha_L \beta_L E_B(f_L(x), y) \\ &\leq 1 + \alpha_L \beta_L (R_B(f_L(x), y) - 1), \end{aligned}$$

and thus $(f_L, g_L, \alpha_L \beta_L)$ is an E-reduction from A to B. Otherwise (that is, A is a maximization problem) we distinguish the following two cases:

1. $E_B(f_L(x), y) \leq \frac{1}{2\alpha_L \beta_L}$: in this case we have that

$$\begin{aligned} R_A(x, g_L(x,y)) - 1 &= \frac{E_A(x, g_L(x,y))}{1 - E_A(x, g_L(x,y))} \\ &\leq \frac{\alpha_L \beta_L E_B(f_L(x), y)}{1 - \alpha_L \beta_L E_B(f_L(x), y)} \\ &\leq 2\alpha_L \beta_L (R_B(f_L(x), y) - 1). \end{aligned}$$

2. $E_B(f_L(x), y) > \frac{1}{2\alpha_L \beta_L}$: in this case we have that $R_B(f_L(x), y) - 1 \geq \frac{1}{2\alpha_L \beta_L}$ so that

$$R_A(x, T(x)) - 1 \leq r - 1 \leq 2\alpha_L \beta_L (r-1)(R_B(f_L(x), y) - 1),$$

where the first inequality is due to the fact that T is an r-approximation algorithm for A.

We can thus define a 3-tuple (f_E, g_E, α_E) as follows:

1. For any $x \in I_A$, $f_E(x) = f_L(x)$.
2. For any $x \in I_A$ and for any $y \in sol_B(f_E(x))$,

$$g_E(x,y) = \begin{cases} g_L(x,y) & \text{if } m_A(x, g_L(x,y)) \geq m_A(x, T(x)), \\ T(x) & \text{otherwise.} \end{cases}$$

3. $\alpha_E = \max\{2\alpha_L \beta_L, 2\alpha_L \beta_L (r-1)\}$.

From the above discussion, it follows that the 3-tuple (f_E, g_E, α_E) is an E-reduction from A to B. □

Clearly, the converse of the above result does not hold since no problem in NPO − NPO PB can be L-reduced to a problem in NPO PB while any problem in PO can be E-reduced to any NPO problem. Moreover, in [28] it is shown that MAXIMUM 3-SATISFIABILITY is (NPO PB∩APX)-complete with respect to the E-reducibility. This result is not obtainable by means of the L-reducibility: indeed, it is easy to prove that MINIMUM BIN PACKING is not L-reducible to MAXIMUM 3-SATISFIABILITY unless P = NP (see, for example, [6]).

The E-reducibility is still somewhat too strict. Indeed, in [16] it has been shown that there exist natural PTAS problems, such as MAXIMUM KNAPSACK, which are not E-reducible to polynomially bounded APX problems, such as MAXIMUM 3-SATISFIABILITY (unless a logarithmic number of queries to an NP oracle is as powerful as a polynomial number of queries).

2.3. The AP-reducibility. The above mentioned drawback of the E-reducibility is mainly due to the fact that an E-reduction preserves optimum values (see [16]). Indeed, the linear relation between the performance ratios seems to be too restrictive. According to the definition of approximation preserving reducibilities given in [14], we could overcome this problem by expressing this relation by means of an implication. However, this is not sufficient: intuitively, since the function g does not know which approximation is required, it must still map optimum solutions into optimum solutions. The final step thus consists of letting the functions f and g depend on the performance ratio.[1] This implies that different constraints have to be put on the computation time of f and g: on one hand, we still want to preserve membership in PTAS; on the other, we want the reduction to be efficient even when poor performance ratios are required. These constraints are formally imposed in the following definition of *approximation preserving* reducibility (which is a restriction of the PTAS-reducibility introduced in [16]).

DEFINITION 2.5. *Let A and B be two* NPO *problems. A is said to be* AP-reducible *to B, in symbols $A \leq_{\mathrm{AP}} B$, if there exist two functions f and g and a positive constant α such that the following hold:*

1. *For any $x \in I_A$ and for any $r > 1$, $f(x,r) \in I_B$ is computable in time $t_f(|x|,r)$.*
2. *For any $x \in I_A$, for any $r > 1$, and for any $y \in sol_B(f(x,r))$, $g(x,y,r) \in sol_A(x)$ is computable in time $t_g(|x|,|y|,r)$.*
3. *For any fixed r, both $t_f(\cdot,r)$ and $t_g(\cdot,\cdot,r)$ are bounded by a polynomial.*
4. *For any fixed n, both $t_f(n,\cdot)$ and $t_g(n,n,\cdot)$ are nonincreasing functions.*
5. *For any $x \in I_A$, for any $r > 1$, and for any $y \in sol_B(f(x,r))$,*

$$R_B(f(x,r),y) \leq r \qquad \Rightarrow \qquad R_A(x,g(x,y,r)) \leq 1 + \alpha(r-1).$$

The 3-tuple (f,g,α) is said to be an AP-reduction from A to B.

According to the above definition, functions like $2^{1/(r-1)}n^h$ or $n^{1/(r-1)}$ are admissible bounds on the computation time of f and g, while this is not true for functions like n^r or 2^n.

Observe that, clearly, the AP-reducibility is a generalization of the E-reducibility. Moreover, it is easy to see that, contrary to the E-reducibility, any PTAS problem is AP-reducible to any NPO problem.

As far as we know, this reducibility is the strictest one appearing in the literature that allows us to obtain natural APX-completeness results (for instance, the APX-completeness of MAXIMUM SATISFIABILITY [16, 28]).

3. NPO-complete problems. In this section we will prove that there are natural problems that are complete for the classes NPO and NPO PB (from now on, unless otherwise specified, we will always refer to the AP-reducibility). Previously, completeness results have been obtained just for Max NPO, Min NPO, Max PB, and Min PB [14, 35, 4, 26]. One example of such a result is the following theorem.

THEOREM 3.1. MINIMUM WEIGHTED ONES *is* Min NPO*-complete and* MAXIMUM WEIGHTED ONES *is* Max NPO*-complete (with respect to the E-reducibility), even if only a subset $\{v_1,\ldots,v_s\}$ of the variables has nonzero weight $w(v_i) = 2^{s-i}$ and any truth assignment satisfying the instance gives the value true to at least one v_i.*

[1] We also let the function f depend on the performance ratio because this feature will turn out to be useful in order to prove interesting characterizations of complete problems for approximation classes.

We will construct AP-reductions from maximization problems to minimization problems and vice versa. Using these reductions we will show that a problem that is Max NPO-complete or Min NPO-complete in fact is complete for the whole of NPO, and that a problem that is Max PB-complete or Min PB-complete is complete for the whole of NPO PB.

THEOREM 3.2. MINIMUM WEIGHTED ONES *and* MAXIMUM WEIGHTED ONES *are* NPO*-complete.*

Proof. In order to establish the NPO-completeness of MINIMUM WEIGHTED ONES we just have to show that there is an AP-reduction from a Max NPO-complete problem to MINIMUM WEIGHTED ONES. As the Max NPO-complete problem we will use the restricted version of MAXIMUM WEIGHTED ONES from Theorem 3.1.

Let x be an instance of MAXIMUM WEIGHTED ONES, i.e., a formula ϕ over variables $v_1, \ldots, v_s$ with weights $w(v_i) = 2^{s-i}$ and some variables with weight zero. We will first give a simple reduction that preserves the approximability within the factor 2, and then adjust it to obtain an AP-reduction.

Let $f(x)$ be the formula $\phi \wedge \alpha_1 \wedge \cdots \wedge \alpha_s$, where α_i is defined as

$$(z_i \equiv (\overline{v}_1 \wedge \cdots \wedge \overline{v}_{i-1} \wedge v_i)),$$

where $z_1, \ldots, z_s$ are new variables with weights $w(z_i) = 2^i$ and where all other variables (even the v-variables) have zero weight. If y is a satisfying assignment of $f(x)$, let $g(x, y)$ be the restriction of the assignment to the variables that occur in ϕ. This assignment clearly satisfies ϕ.

Note that exactly one of the z-variables is true in any satisfying assignment of $f(x)$. Indeed, if all z-variables were false, then all v-variables would be false and ϕ would not be satisfied. On the other hand, if both z_i and z_j were true with $j > i$, then v_i would be both true and false, which is a contradiction. Hence,

$$\begin{aligned} m(f(x), y) = 2^i &\Leftrightarrow z_i = 1 \\ &\Leftrightarrow v_1 = v_2 = \cdots = v_{i-1} = 0, v_i = 1 \\ &\Leftrightarrow 2^{s-i} \leq m(x, g(x, y)) < 2 \cdot 2^{s-i} \\ &\Leftrightarrow \frac{2^s}{m(f(x), y)} \leq m(x, g(x, y)) < 2\frac{2^s}{m(f(x), y)}. \end{aligned}$$

In particular this holds for the optimum solution. Thus the performance ratio for MAXIMUM WEIGHTED ONES is

$$R(x, g(x, y)) = \frac{\mathsf{opt}(x)}{m(x, g(x, y))} < \frac{2\dfrac{2^s}{\mathsf{opt}(f(x))}}{\dfrac{2^s}{m(f(x), y)}} = 2\frac{m(f(x), y)}{\mathsf{opt}(f(x))} = 2R(f(x), y),$$

which means that the reduction preserves the approximability within 2.

Let us now extend the construction in order to obtain $R(x, g(x, y)) \leq (1 + 2^{-k})R(f_k(x), y)$ for every nonnegative integer k. The reduction described above corresponds to $k = 0$.

For any $i \in \{1, \ldots, s\}$ and for any $(b_1, \ldots, b_{k(i)}) \in \{0, 1\}^{k(i)}$, where $k(i) = \min\{s - i, k\}$, we have a variable $z_{i,b_1,\ldots,b_{k(i)}}$. Let

$$f_k(x) = \phi \wedge \bigwedge_{\substack{i \in \{1,\ldots,s\} \\ (b_1,\ldots,b_{k(i)}) \in \{0,1\}^{k(i)}}} \alpha_{i,b_1,\ldots,b_{k(i)}},$$

where $\alpha_{i,b_1,\ldots,b_{k(i)}}$ is defined as

$$\left(z_{i,b_1,\ldots,b_{k(i)}} \equiv \left(\overline{v}_1 \wedge \cdots \wedge \overline{v}_{i-1} \wedge v_i \wedge (v_{i+1} = b_1) \wedge \cdots \wedge (v_{i+k(i)} = b_{k(i)})\right)\right).$$

Define $g(x, y)$ as above. Finally, define

$$w(z_{i,b_1,\ldots,b_{k(i)}}) = \left\lceil \frac{K \cdot 2^s}{w(v_i) + \sum_{j=1}^{k(i)} b_j w(v_{i+j})} \right\rceil = \left\lceil \frac{K \cdot 2^i}{1 + \sum_{j=1}^{k(i)} b_j 2^{-j}} \right\rceil$$

(by choosing K greater than 2^k we can disregard the effect of the ceiling operation in the following computations).

As in the previous reduction exactly one of the z-variables is true in any satisfying assignment of $f_k(x)$. If, in a solution y of $f_k(x)$, $z_{i,b_1,\ldots,b_{k(i)}} = 1$, then we have $m(f_k(x), y) = w(z_{i,b_1,\ldots,b_{k(i)}})$ and we know that

$$m(x, g(x, y)) \geq w(v_i) + \sum_{j=1}^{k(i)} b_j w(v_{i+j}) = 2^{s-i} \left(1 + \sum_{j=1}^{k(i)} b_j 2^{-j}\right).$$

On the other hand, if $k(i) = s - i$, then $m(x, g(x, y)) = 2^{s-i}(1 + \sum_{j=1}^{k(i)} b_j 2^{-j})$; otherwise

$$m(x, g(x, y)) \leq w(v_i) + \sum_{j=1}^{k} b_j w(v_{i+j}) + \sum_{j=k+i+1}^{s} w(v_j) < 2^{s-i} \left(1 + \sum_{j=1}^{k} b_j 2^{-j}\right)(1 + 2^{-k}).$$

In both cases, we thus get

$$\frac{K \cdot 2^s}{m(f_k(x), y)} \leq m(x, g(x, y)) < \frac{K \cdot 2^s}{m(f_k(x), y)}(1 + 2^{-k})$$

and therefore $R(x, g(x, y)) < (1 + 2^{-k})R(f_k(x), y)$. Given any $r > 1$, if we choose k such that $2^{-k} \leq (r - 1)/r$, e.g., $k = \lceil \log r - \log(r - 1) \rceil$, then $R(f_k(x), y) \leq r$ implies $R(x, g(x, y)) < (1 + 2^{-k})R(f_k(x), y) \leq r + r2^{-k} \leq r + r - 1 = 1 + 2(r - 1)$. This is obviously an AP-reduction with $\alpha = 2$.

A very similar proof can be used to show that Maximum Weighted Ones is NPO-complete. □

Corollary 3.3. *Any* Min NPO*-complete problem is* NPO*-complete and any* Max NPO*-complete problem is* NPO*-complete.*

As an application of the above corollary, we have that the Minimum $0-1$ Programming problem is NPO-complete.

We can also show that there are natural complete problems for the class of polynomially bounded NPO problems.

Theorem 3.4. Maximum PB $0-1$ Programming *and* Minimum PB $0-1$ Programming *are* NPO PB*-complete.*

Proof. Maximum PB $0-1$ Programming is known to be Max PB-complete [4] and Minimum PB $0-1$ Programming is known to be Min PB-complete [26]. Thus we just have to show that there are AP-reductions from Minimum PB $0-1$ Programming to Maximum PB $0-1$ Programming and from Maximum PB $0-1$ Programming to Minimum PB $0-1$ Programming.

Both reductions use exactly the same construction. Given a satisfying variable assignment, we define the *one-variables* to be the variables occurring in the objective

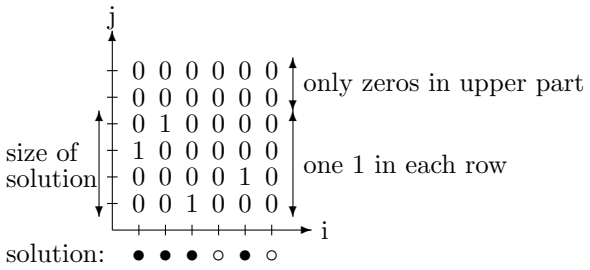


FIG. 3.1. *The idea of the reduction from* MINIMUM/MAXIMUM PB $0-1$ PROGRAMMING *to* MAXIMUM/MINIMUM PB $0-1$ PROGRAMMING. *The variable* $x_i^j = 1$ *if and only if* v_i *is the* j*th one-variable in the solution. There is at most one* 1 *in each column and in each row.*

function that have the value one. The objective value is the number of one-variables plus 1.

The objective value of a solution is encoded by introducing an order of the one-variables. The order is encoded by a squared number of $0-1$ variables (see Fig. 3.1). The idea is to invert the objective values, so that a solution without one-variables corresponds to an objective value of n of the constructed problem, and, in general, a solution with p one-variables corresponds to an objective value of $\left\lfloor \frac{n}{p+1} \right\rfloor$.

The reductions are constructed as follows. Given an instance of MINIMUM PB $0-1$ PROGRAMMING or MAXIMUM PB $0-1$ PROGRAMMING, i.e., an objective function $1 + \sum_{k=1}^{m} v_k$ and some inequalities over variables $V = \{v_1, \ldots, v_m\} \cup U$, construct m^2 variables x_i^j, $1 \le i, j \le m$, and the following inequalities:

$$\forall i \in [1 \ldots m] \qquad \sum_{j=1}^{m} x_i^j \le 1 \qquad \text{(at most one 1 in each column)}, \tag{3.1}$$

$$\forall j \in [1 \ldots m] \qquad \sum_{i=1}^{m} x_i^j \le 1 \qquad \text{(at most one 1 in each row)}, \tag{3.2}$$

$$\forall j \in [1 \ldots m-1] \ \sum_{i=1}^{m} x_i^j - \sum_{i=1}^{m} x_i^{j+1} \ge 0 \ \text{(only zeros in upper part)}. \tag{3.3}$$

Besides these inequalities we include all inequalities from the original problem, but we substitute each variable v_i with the sum $\sum_{k=1}^{m} x_i^k$. The variables in U (that do not occur in the objective function) are left intact.

The objective function is defined as

$$n - \sum_{p=1}^{m} \left(\left\lfloor \frac{n}{p} \right\rfloor - \left\lfloor \frac{n}{p+1} \right\rfloor \right) \sum_{i=1}^{m} x_i^p. \tag{3.4}$$

In order to express the objective function with only binary coefficients we have to introduce n new variables $y_1, \ldots, y_n$, where $y_j = 1 - \sum_{i=1}^{m} x_i^p$ for $\lfloor n/(p+1) \rfloor < j \le \lfloor n/p \rfloor$ and $y_j = 1$ for $j \le \lfloor n/(m+1) \rfloor$. The objective function then is $\sum_{j=1}^{n} y_j$. One can now verify that a solution of the original problem instance with s one-variables (i.e., with an objective value of $s+1$) will exactly correspond to a solution of the constructed problem instance with objective value $\lfloor n/(s+1) \rfloor$ and vice versa.

Suppose that the optimum solution to the original problem instance has M one-variables; then the performance ratio $(s+1)/(M+1)$ will correspond to the performance ratio

$$\frac{\left\lfloor \dfrac{n}{M+1} \right\rfloor}{\left\lfloor \dfrac{n}{s+1} \right\rfloor} = \frac{s+1}{M+1}\left(1 \pm \frac{m}{n}\right)$$

for the constructed problem, where $\frac{m}{n}$ is the relative error due to the floor operation. By choosing n large enough, the relative error can be made arbitrarily small. Thus it is easy to see that the reduction is an AP-reduction. □

COROLLARY 3.5. *Any* Min PB*-complete problem is* NPO PB*-complete and any* Max PB*-complete problem is* NPO PB*-complete.*

4. Query complexity and APX-intermediate problems. The existence of APX-intermediate problems (that is, problems in APX which are not APX-complete) has already been shown in [14] (assuming P $\neq$ NP) where an artificial such problem is obtained by diagonalization techniques similar to those developed to prove the existence of NP-intermediate problems [31]. In this section, we prove that "natural" APX-intermediate problems exist: for instance, we will show that MINIMUM BIN PACKING is APX-intermediate. In order to prove this result, we will establish new connections between the approximability properties and the query complexity of NP-hard optimization problems. To this aim, let us first recall the following definition.

DEFINITION 4.1. *A language L belongs to the* class $\mathrm{P}^{\mathrm{NP}[f(n)]}$ *if it is decidable by a polynomial-time oracle Turing machine which asks at most $f(n)$ queries to an* NP*-complete oracle, where n is the input size. The* class QH *is defined to be the union* $\bigcup_{k>1} \mathrm{P}^{\mathrm{NP}[k]}$.

Similarly, we can define the class of functions $\mathrm{FP}^{\mathrm{NP}[f(n)]}$ [30]. The following result has been proved in [23, 24].

THEOREM 4.2. *If there exists a constant k such that*

$$\mathrm{QH} = \mathrm{P}^{\mathrm{NP}[k]},$$

then the polynomial-time hierarchy collapses.

The query-complexity of the "nonconstructive" approximation of several NP-hard optimization problems has been studied by using hardness results with respect to classes of functions $\mathrm{FP}^{\mathrm{NP}[\cdot]}$ [7, 9]. However, this approach cannot be applied to analyze the complexity of "constructing" approximate solutions. To overcome this limitation, we use a novel approach that basically consists of considering how helpful is an approximation algorithm for a given optimization problem to solve decision problems.

DEFINITION 4.3. *Given an* NPO *problem A and a rational $r \geq 1$, A_r is a multivalued partial function that, given an instance x of A, returns the set of feasible solutions y of x such that $R(x,y) \leq r$.*

DEFINITION 4.4. *Given an* NPO *problem A and a rational $r \geq 1$, a language L belongs to P^{A_r} if there exist two polynomial-time computable functions f and g such that, for any x, $f(x)$ is an instance of A with $sol(f(x)) \neq \emptyset$, and, for any $y \in A_r(f(x))$, $g(x,y) = 1$ if and only if $x \in L$. The* class $\mathrm{AQH}(A)$ *is equal to the union* $\bigcup_{r>1} \mathrm{P}^{A_r}$.

The following result states that an approximation problem does not help more than a constant number of queries to an NP-complete problem. It is worth observing that, in general, an approximate solution, even though not very helpful, requires more than a logarithmic number of queries to be computed [8].

PROPOSITION 4.5. *For any problem A in* APX, $\mathrm{AQH}(A) \subseteq \mathrm{QH}$.

Proof. Assume that A is a maximization problem (the proof for minimization problems is similar). Let T be an r-approximate algorithm for A, for some $r > 1$, and let $L \in \mathrm{P}^{A_\rho}$ for some $\rho > 1$. Two polynomial-time computable functions f and g then exist witnessing this latter fact. For any x, let $m = m(f(x), T(f(x)))$, so that $m \leq \mathsf{opt}(f(x)) \leq rm$. We can then partition the interval $[m, rm]$ into $\lfloor \log_\rho r \rfloor + 1$ subintervals

$$[m, \rho m), [\rho m, \rho^2 m), \ldots, [\rho^{\lfloor \log_\rho r \rfloor - 1} m, \rho^{\lfloor \log_\rho r \rfloor} m], [\rho^{\lfloor \log_\rho r \rfloor} m, rm],$$

and start looking for the subinterval containing the optimum value. (A similar technique has been used in [7, 9].) This can clearly be done using $\lfloor \log_\rho r \rfloor + 1$ queries to an NP-complete oracle. One more query is sufficient to know whether there exists a feasible solution y whose value lies in that interval and such that $g(x, y) = 1$. Since y is ρ-approximate, it follows that L can be decided using $\lfloor \log_\rho r \rfloor + 2$ queries, that is, $L \in \mathrm{QH}$. □

Recall that an NPO problem admits an *asymptotic polynomial-time approximation scheme* if there exists an algorithm T such that, for any x and for any $r > 1$, $R(x, T(x, r)) \leq r + k/\mathsf{opt}(x)$ with k constant and the time complexity of $T(x, r)$ is polynomial with respect to $|x|$. The class of problems that admit an asymptotic polynomial-time approximation scheme is usually denoted by PTAS^∞. The following result shows that, for this class, the previous fact can be strengthened.

PROPOSITION 4.6. *Let $A \in \mathrm{PTAS}^\infty$. Then there exists a constant h such that* $\mathrm{AQH}(A) \subseteq \mathrm{P}^{\mathrm{NP}[h]}$.

Proof. Let A be a minimization problem in PTAS^∞. (The proof for maximization problem is very similar.) By definition, there exist a constant k and an algorithm T such that, for any instance x and for any rational $r > 1$,

$$m(x, T(x, r)) \leq r \cdot \mathsf{opt}(x) + k.$$

We will now prove that there exists a constant h such that, for any $r > 1$, a function $l_r \in \mathrm{FP}^{\mathrm{NP}[h-1]}$ exists such that, for any instance x of the problem A,

$$\mathsf{opt}(x) \leq l_r(x) \leq r \cdot \mathsf{opt}(x).$$

Intuitively, functions l_r form a nonconstructive approximation scheme that is computable by a constant number of queries to an NP-complete oracle. Given an instance x, we can check whether $sol(x) = \emptyset$ by means of a single query to an NP oracle, so that we can restrict ourselves to instances such that $sol(x) \neq \emptyset$ (and thus $\mathsf{opt}(x) \geq 1$). Note that for these instances, $T(\cdot, 2)$ is a $(k + 2)$-approximate algorithm for A. Let us fix an $r > 1$, let $\varepsilon = r - 1$, $y = T(x, 1 + \varepsilon/2)$, and $a = m(x, T(x, 2))$. We have to distinguish two cases.

1. $a \geq 2k(k + 2)/\varepsilon$: in this case, $\mathsf{opt}(x) \geq 2k/\varepsilon$, that is, $\mathsf{opt}(x)\varepsilon/2 \geq k$. Then

$$\begin{aligned} m(x, y) &\leq \mathsf{opt}(x)(1 + \varepsilon/2) + k \\ &\leq \mathsf{opt}(x)(1 + \varepsilon/2) + \mathsf{opt}(x)\varepsilon/2 \\ &= \mathsf{opt}(x)(1 + \varepsilon) = r\mathsf{opt}(x); \end{aligned}$$

that is, y is an r-approximate solution for x, and we can set $l_r(x) = m(x, y)$. (In this case l_r has been computed by only one query.)

2. $a < 2k(k+2)/\varepsilon$: in this case, $\mathsf{opt}(x) < 2k(k+2)/\varepsilon$. Then,

$$\mathsf{opt}(x) \leq m(x, y) \leq \mathsf{opt}(x) + \mathsf{opt}(x)\varepsilon/2 + k < \mathsf{opt}(x) + k(k+2) + k.$$

Clearly, $\lceil \log k(k+3) \rceil$ queries to NP are sufficient to find the optimum value $\mathsf{opt}(x)$ by means of a binary search technique: in this case $l_r(x) = \mathsf{opt}(x)$ has been computed by $\lceil \log k(k+3) \rceil + 1$ queries.

Let L now be a language in AQH(A); then $L \in \mathrm{P}^{A_r}$ for some $r > 1$. Let f and g be the functions witnessing that $L \in \mathrm{P}^{A_r}$. Observe that, for any x, $x \in L$ if and only if there exists a solution y for $f(x)$ such that $m(f(x), y) \geq l_r(f(x))$ and $g(f(x), y) = 1$; that is, given $l_r(f(x))$, deciding whether $x \in L$ is an NP problem. Since $l_r(f(x))$ is computable by means of at most $\lceil \log k(k+3) \rceil + 1$ queries to NP, we have that $L \in \mathrm{P}^{\mathrm{NP}[h]}$, where $h = \lceil \log k(k+3) \rceil + 2$. □

The next proposition, instead, states that any language L in the query hierarchy can be decided using just one query to A_r, where A is APX-complete and r depends on the level of the query hierarchy L belongs to. In order to prove this proposition, we need the following technical result to be realized.[2]

LEMMA 4.7. *For any* APX*-complete problem A and for any k, there exist two polynomial-time computable functions f and g and a constant r such that the following hold:*

1. *For any k-tuple $(x_1, \ldots, x_k)$ of instances of* PARTITION*, $x = f(x_1, \ldots, x_k)$ is an instance of A.*

2. *If y is a solution of x whose performance ratio is smaller than r, then $g(x, y) = (b_1, \ldots, b_k)$, where $b_i \in \{0, 1\}$ and $b_i = 1$ if and only if x_i is a yes-instance.*

Proof. Let $x_i = (U_i, s_i)$ be an instance of PARTITION for $i = 1, \ldots, k$. Without loss of generality, we can assume that the U_is are pairwise disjoint and that, for any i, $\sum_{u \in U_i} s_i(u) = 2$. Let $w = (U, s, \preceq)$ be an instance of MINIMUM ORDERED BIN PACKING defined as follows. (A similar construction has been used in [39].)

1. $U = \bigcup_{i=1}^{k} U_i \cup \{v_1, \ldots, v_{k-1}\}$, where the v_is are new items.
2. For any $u \in U_i$, $s(u) = s_i(u)$ and $s(v_i) = 1$ for $i = 1, \ldots, k-1$.
3. For any $i < j \leq k$, for any $u \in U_i$, and for any $u' \in U_j$, $u \preceq v_i \preceq u'$.

Any solution of w must be formed by a sequence of packings of $U_1, \ldots, U_k$ such that, for any i, the bins used for U_i are separated by the bins used for U_{i+1} by means of one bin which is completely filled by v_i. In particular, the packings of the U_is in any optimum solution must use either two or three bins: two bins are used if and only if x_i is a yes-instance. The optimum measure thus is at most $4k - 1$ so that any $(1 + 1/(4k))$-approximate solution is an optimum solution.

Since MINIMUM ORDERED BIN PACKING belongs to APX [43] and A is APX-complete, there then exists an AP-reduction (f_1, g_1, α) from MINIMUM ORDERED BIN PACKING to A. We can then define $x = f(x_1, \ldots, x_k) = f_1(w, 1 + 1/(4\alpha k))$ and $r = 1 + 1/(4\alpha k)$. For any r-approximate solution y of x, the fourth property of the AP-reducibility implies that $z = g_1(x, y, 1 + 1/(4\alpha k))$ is a $(1 + 1/(4k))$-approximate

[2] Recall that the NP-complete problem PARTITION is defined as follows: given a set U of items and a size function $s : U \to Q \cap (0, 1]$, does there exist a subset $U' \subseteq U$ such that

$$\sum_{u \in U'} s(u) = \sum_{u \notin U'} s(u) \ ?$$

solution of w and thus an optimum solution of w. From z, we can easily derive the right answers to the k queries $x_1, \ldots, x_k$. □

We are now able to prove the following result.

PROPOSITION 4.8. *For any* APX*-complete problem* A, $\mathrm{QH} \subseteq \mathrm{AQH}(A)$.

Proof. Let $L \in \mathrm{QH}$, then $L \in \mathrm{P}^{\mathrm{NP}[h]}$ for some h. It is well known (see, for instance, [3]) that L can be reduced to the problem of answering $k = 2^{h-1}$ nonadaptive queries to NP. More formally, there exist two polynomial-time computable functions t_1 and t_2 such that:

1. For any x, $t_1(x) = (x_1, \ldots, x_k)$, where $x_1, \ldots, x_k$ are k instances of the PARTITION problem.
2. For any $(b_1, \ldots, b_k) \in \{0,1\}^k$, $t_2(x, b_1, \ldots, b_k) \in \{0,1\}$.
3. If, for any j, $b_j = 1$ if and only if x_j is a yes-instance, then $t_2(x, b_1, \ldots, b_k) = 1$ if and only if $x \in L$.

Let now f, g, and r be the two functions and the constant of Lemma 4.7 applied to problem A and constant k. For any x, $x' = f(t_1(x))$ is an instance of A such that if y is an r-approximate solution for x', then $t_2(g(x', y)) = 1$ if and only if $x \in L$. Thus, $L \in \mathrm{P}^{A_r}$. □

By combining Propositions 4.5 and 4.8, we thus have the following theorem that characterizes the approximation query hierarchy of the hardest problems in APX.

THEOREM 4.9. *For any* APX*-complete problem* A, $\mathrm{AQH}(A) = \mathrm{QH}$.

Finally, we have the following result that states the existence of natural intermediate problems in APX.

THEOREM 4.10. *If the polynomial-time hierarchy does not collapse, then* MINIMUM BIN PACKING*,* MINIMUM DEGREE SPANNING TREE*, and* MINIMUM EDGE COLORING *are* APX*-intermediate.*

Proof. From Proposition 4.6 and from the fact that MINIMUM BIN PACKING is in PTAS^{∞} [27], it follows that AQH(MINIMUM BIN PACKING) $\subseteq \mathrm{P}^{\mathrm{NP}[h]}$ for a given h. If MINIMUM BIN PACKING is APX-complete, then from Proposition 4.8 it follows that $\mathrm{QH} \subseteq \mathrm{P}^{\mathrm{NP}[h]}$. From Theorem 4.2 we thus have the collapse of the polynomial-time hierarchy. The proofs for MINIMUM DEGREE SPANNING TREE and MINIMUM EDGE COLORING are identical and use the results of [20, 17]. □

Observe that the previous result does not seem to be obtainable by using the hypothesis $\mathrm{P} \neq \mathrm{NP}$, as shown by the following theorem.

THEOREM 4.11. *If* NP = *co*-NP*, then* MINIMUM BIN PACKING *is* APX*-complete.*

Proof. Assume NP = *co*-NP; we will present an AP reduction from MAXIMUM SATISFIABILITY to MINIMUM BIN PACKING. Since NP = *co*-NP, there exists a nondeterministic polynomial-time Turing machine M that, given in input an instance ϕ of MAXIMUM SATISFIABILITY, has an accepting computation and all accepting computations halt with an optimum solution for ϕ written on the tape. Indeed, M guesses an integer k, an assignment τ such that $m(\phi, \tau) = k$, and a proof of the fact that $\mathsf{opt}(\phi) \leq k$. From the proof of Cook's theorem it follows that, given ϕ, we can find in polynomial time a formula ϕ' such that ϕ' is satisfiable and that given any satisfying assignment for ϕ', we can find in polynomial time an optimum solution for ϕ. By combining this construction with the NP-completeness proof of the MINIMUM BIN PACKING problem, we obtain two polynomial-time computable functions t_1 and t_2 such that, for any instance ϕ of MAXIMUM SATISFIABILITY, $t_1(\phi) = x_\phi$ is an instance of MINIMUM BIN PACKING such that $\mathsf{opt}(x_\phi) = 2$ and, for any optimum solution y of x_ϕ, $t_2(x_\phi, y)$ is an optimum solution of ϕ. Observe that, by construction,

an r-approximate solution for x_ϕ is indeed an optimum solution provided that $r < 3/2$. Let T be a 4/3-approximate algorithm for MAXIMUM SATISFIABILITY [44, 19]. The reduction from MAXIMUM SATISFIABILITY to MINIMUM BIN PACKING is defined as follows: $f(\phi, r) = t_1(\phi)$;

$$g(\phi, y, r) = \begin{cases} T(\phi) & \text{if } r \geq 4/3, \\ t_2(t_1(\phi), y) & \text{otherwise.} \end{cases}$$

It is easy to verify that the above is an AP-reduction with $\alpha = 1$. □

Finally, note that the above result can be extended to any APX problem which is NP-hard to approximate within a given performance ratio.

4.1. A remark on MAXIMUM CLIQUE. The following lemma is the analogue of Proposition 4.5 within NPO PB and can be proved similarly by binary search techniques.

LEMMA 4.12. *For any* NPO PB *problem A and for any $r > 1$,*

$$\mathrm{P}^{A_r} \subseteq \mathrm{P}^{\mathrm{NP}[\log\log n + O(1)]}.$$

From this lemma, from the fact that $\mathrm{P}^{\mathrm{NP}[\log n]}$ is contained in $\mathrm{P}^{\mathrm{MC}_1}$, where MC stands for MAXIMUM CLIQUE [30], and from the fact that if there exists a constant k such that

$$\mathrm{P}^{\mathrm{NP}[\log\log n + k]} = \mathrm{P}^{\mathrm{NP}[\log n]},$$

then the polynomial-time hierarchy collapses [42], it follows that the next result solves an open question posed in [7]. Informally, this result states that it is not possible to reduce the problem of finding a maximum clique to the problem of finding a 2-approximate clique (unless the polynomial-time hierarchy collapses).

THEOREM 4.13. *If $\mathrm{P}^{\mathrm{MC}_1} \subseteq \mathrm{P}^{\mathrm{MC}_2}$ then the polynomial-time hierarchy collapses.*

5. Query complexity and completeness in approximation classes. In this final section, we shall give a full characterization of problems complete for poly-APX and APX, respectively—in terms of hardness of the corresponding approximation problems with respect to classes of partial multivalued functions and in terms of suitably defined combinatorial properties.

The classes of functions we will refer to have been introduced in [8] as follows.

DEFINITION 5.1. $\mathrm{FNP}^{\mathrm{NP}[q(n)]}$ *is the class of partial multivalued functions computable by nondeterministic polynomial-time Turing machines which ask at most $q(n)$ queries to an* NP *oracle in the* entire *computation tree.*[3]

In order to talk about hardness with respect to these classes we will use the following reducibility, which is an extension of both metric reducibility [30] and one-query reducibility [15] and has been introduced in [8].

DEFINITION 5.2. *Let F and G be two partial multivalued functions. We say that F many-one reduces to G (in symbols, $F \leq_{\mathrm{mv}} G$) if there exist two polynomial-time algorithms t_1 and t_2 such that, for any x in the domain of F, $t_1(x)$ is in the domain of G and, for any $y \in G(t_1(x))$, $t_2(x, y) \in F(x)$.*

The combinatorial property used to characterize poly-APX-complete problems is the well-known self-improvability (see, for instance, [36]).

[3] We say that a multivalued partial function F is computable by a nondeterministic Turing machine N if, for every x in the domain of F, there exists a halting computation path of $N(x)$ and any halting computation path of $N(x)$ outputs a value in $F(x)$.

DEFINITION 5.3. *A problem A is* self-improvable *if there exist two algorithms t_1 and t_2 such that, for any instance x of A and for any two rationals $r_1, r_2 > 1$, $x' = t_1(x, r_1, r_2)$ is an instance of A and, for any $y' \in A_{r_2}(x')$, $y = t_2(x, y', r_1, r_2) \in A_{r_1}(x)$. Moreover, for any fixed r_1 and r_2, the running time of t_1 and t_2 is polynomial.*

We are now ready to state the first result of this section.

THEOREM 5.4. *A poly-*APX *problem A is poly-*APX*-complete if and only if it is self-improvable and A_{r_0} is $\mathrm{FNP}^{\mathrm{NP}[\log\log n + O(1)]}$-hard for some $r_0 > 1$.*

Proof. Let A be a poly-APX-complete problem. Since MAXIMUM CLIQUE is self-improvable [18] and poly-APX-complete [28] and since the equivalence with respect to the AP-reducibility preserves the self-improvability property (see [36]), we have that A is self-improvable. It is then sufficient to prove that A_2 is hard for $\mathrm{FNP}^{\mathrm{NP}[\log\log n + O(1)]}$.

From the poly-APX-completeness of A we have that MAXIMUM CLIQUE $\leq_{\mathrm{AP}} A$: let α be the constant of this reduction. From Theorem 12 of [8] we have that any function F in $\mathrm{FNP}^{\mathrm{NP}[\log\log n + O(1)]}$ many-one reduces to MAXIMUM CLIQUE$_{1+\alpha}$. From the definition of AP-reducibility, we also have that MAXIMUM CLIQUE$_{1+\alpha} \leq_{\mathrm{mv}} A_2$ so that F many-one reduces to A_2.

Conversely, let A be a poly-APX self-improvable problem such that, for some r_0, A_{r_0} is $\mathrm{FNP}^{\mathrm{NP}[\log\log n + O(1)]}$-hard. We will show that, for any problem B in poly-APX, B is AP-reducible to A. To this aim, we introduce the following partial multivalued function `multisat`: given an input sequence $(\phi_1, \ldots, \phi_m)$ of instances of the satisfiability problem with $m \leq \log|(\phi_1, \ldots, \phi_m)|$ and such that, for any i, if ϕ_{i+1} is satisfiable then ϕ_i is satisfiable, a possible output is a satisfying truth assignment for ϕ_{i^*}, where $i^* = \max\{i : \phi_i \text{ is satisfiable}\}$. From the proof of Theorem 12 of [8] it follows that this function is $\mathrm{FNP}^{\mathrm{NP}[\log\log n + O(1)]}$-complete.

By making use of techniques similar to those of the proof of Proposition 4.5, it is easy to see that, since B is in poly-APX, there exist two algorithms t_1^B and t_2^B such that, for any fixed $r > 1$, $t_1^B(\cdot, r)$ and $t_2^B(\cdot, \cdot, r)$ form a many-one reduction from B_r to `multisat`. Moreover, since A_{r_0} is $\mathrm{FNP}^{\mathrm{NP}[\log\log n + O(1)]}$-hard, there then exists a many-one reduction (t_1^M, t_2^M) from `multisat` to A_{r_0}. Finally, let t_1^A and t_2^A be the functions witnessing the self-improvability of A.

The AP-reduction from B to A can then be derived as follows:

$$\begin{array}{ccccccc} x, r & \xrightarrow{t_1^B(x,r)} & x' & \xrightarrow{t_1^M(x')} & x'' & \xrightarrow{t_1^A(x'', r_0, r)} & x''' \\ & & & & & & \downarrow \\ y & \xleftarrow{t_2^B(x, y', r)} & y' & \xleftarrow{t_2^M(x', y'')} & y'' & \xleftarrow{t_2^A(x'', y''', r_0, r)} & y''' \end{array}$$

It is easy to see that if y''' is an r-approximate solution for the instance x''' of A, then y is an r-approximate solution of the instance x of B. That is, B is AP-reducible to A with $\alpha = 1$. □

The above theorem cannot be proved without the dependency of both f and g on r in the definition of AP-reducibility. Indeed, it is possible to prove that if only g has this property then, unless the polynomial-time hierarchy collapses, there exists a self-improvable problem A such that A_2 is $\mathrm{FNP}^{\mathrm{NP}[\log\log n + O(1)]}$-hard and A is not poly-APX-complete.

In order to characterize APX-complete problems, we have to define a different combinatorial property. Intuitively, this property states that it is possible to merge several instances into one instance in an approximation preserving fashion.

DEFINITION 5.5. *An* NPO *problem A is* linearly additive *if there exist a constant β and two algorithms t_1 and t_2 such that, for any rational $r > 1$ and for any sequence $x_1, \ldots, x_k$ of instances of A, $x' = t_1(x_1, \ldots, x_k, r)$ is an instance of A and, for any $y' \in A_{1+(r-1)\beta/k}(x')$, $t_2(x_1, \ldots, x_k, y', r) = y_1, \ldots, y_k$, where each y_i is an r-approximate solution of x_i. Moreover, the running time of t_1 and t_2 is polynomial for every fixed r.*

THEOREM 5.6. *An* APX *problem A is* APX*-complete if and only if it is linearly additive and there exists a constant r_0 such that A_{r_0} is* $\mathrm{FNP}^{\mathrm{NP}[1]}$*-hard.*

Proof. Let A be an r_A-approximable APX-complete problem. From the proof of Proposition 4.8 there exists a constant r_0 such that A_{r_0} is hard for $\mathrm{FNP}^{\mathrm{NP}[1]}$. In order to prove the linear additivity, fix any $r > 1$ and let $x_1, \ldots, x_k$ be instances of A. Without loss of generality, we can assume $r < r_A$ (otherwise the k instances can be r-approximated by using the r_A-approximate algorithm). For any $i = 1, \ldots, k$ the problem of finding an r-approximate solution y_i for x_i is reducible to the problem of constructively solving a set of $\lceil \log_r r_A \rceil$ instances of PARTITION. Observe that $\lceil \log_r r_A \rceil \leq c/(r-1)$ for a certain constant c depending on r_A. Moreover, we claim that there exists a constant γ such that constructively solving $kc/(r-1)$ instances of PARTITION is reducible to $(1 + \gamma(r-1)/kc)$-approximating a single instance of A (indeed, this can be shown along the lines of the proof of Proposition 4.8). That is, A is linearly additive with $\beta = \gamma/c$.

Conversely, let A be a linearly additive APX problem such that A_{r_0} is $\mathrm{FNP}^{\mathrm{NP}[1]}$-hard for some r_0 and let B be an r_B-approximable problem. Given an instance x of B, for any $r > 1$ we can reduce the problem of finding an r-approximate solution for x to the problem of constructively solving $c/(r-1)$ instances of PARTITION, for a proper constant c not depending on r. Each of these questions is reducible to A_{r_0}, since any NP problem can be constructively solved by an $\mathrm{FNP}^{\mathrm{NP}[1]}$ function. From linear additivity, it follows that r_0-approximating $c/(r-1)$ instances of A is reducible to $(1+\beta(r_0-1)(r-1)/c)$-approximating a single instance of A. This is an AP-reduction from B to A with $\alpha = c/(\beta(r_0-1))$. □

Note that linear additivity plays for APX more or less the same role of self-improvability for poly-APX. These two properties are, in a certain sense, opposites: while the ability of APX-complete approximation problems to solve decision problems depends on the performance ratio and does not depend on the size of the instance, the usefulness of poly-APX-complete approximation problems depends on the size of the instance and does not depend on the performance ratio. Indeed, it is possible to prove that no APX-complete problem can be self-improvable (unless P = NP) and that no poly-APX-complete problem can be linearly additive (unless the polynomial-time hierarchy collapses).

It is now an interesting question to find a characterizing combinatorial property of log-APX-complete problems. Indeed, we have not been able to establish this characterization; at present, we can state only that it cannot be based on the self-improvability property as shown by the following result.

THEOREM 5.7. *No log-*APX*-complete problem can be self-improvable unless the polynomial-time hierarchy collapses.*

Proof. Let us consider the following optimization problem.

MAX NUMBER OF SATISFIABLE FORMULAS (MNSF).

INSTANCE: Set of m Boolean formulas $\phi_1, \ldots, \phi_m$ in 3CNF, such that ϕ_1 is a tautology and $m \leq \log n$, where n is the size of the input instance.

SOLUTION: Truth assignment τ to the variables of $\phi_1, \ldots, \phi_m$.

MEASURE: The number of satisfied formulas, i.e., $|\{i : \phi_i \text{ is satisfied by } \tau\}|$.

Clearly, MNSF is in log-APX, since the measure of any assignment τ is at least 1, and the optimum value is always smaller than $\log n$, where n is the size of the input. We will show that, for any $r < 2$, MNSF_r is hard for $\text{FNP}^{\text{NP}[\log\log\log n-1]}$.

Given $\log\log n$ queries to an NP-complete language (of size polynomial in n) $x_1, \ldots, x_{\log\log n}$, we can construct an instance $\Phi = \phi_1, \ldots, \phi_m$ of MNSF, where ϕ_1 is a tautology and, for $i \geq 1$, the formulas $\phi_{2^i} = \cdots = \phi_{2^{i+1}-1}$ are satisfiable if and only if at least i instances among $x_1, \ldots, x_{\log\log n}$ are yes-instances (these formulas can be easily constructed using the standard proof of Cook's theorem). Note that $m = 2^{\log\log n+1} - 1$ and, by adding dummy clauses to some formulas, we can achieve the bound $m \leq \log|\phi_1, \ldots, \phi_m|$. Moreover, from an r-approximate solution for Φ we can decide how many instances in $x_1, \ldots, x_{\log\log n}$ are yes-instances, and we can also recover solutions for such instances. That is, any function in $\text{FNP}^{\text{NP}[\log\log\log n-1]}$ is many-one reducible to MNSF_r.

Let A be a self-improvable log-APX-complete problem. Then, for any function $F \in \text{FNP}^{\text{NP}[\log\log\log n-1]}$, $F \leq_{\text{mv}} \text{MNSF}_{1.5} \leq_{\text{mv}} A_{1+\alpha/2} \leq_{\text{mv}} A_{2^{16}}$, where α is the constant in the AP-reduction from MNSF to A and where the last reduction is due to the self-improvability of A. Thus, for any x, computing $F(x)$ is reducible to finding a 2^{16}-approximate solution for an instance x' with $|x'| \leq |x|^c$ for a certain constant c. Since $A \in$ log-APX, it is possible to find in polynomial time a $(k\log|x'|)$-approximate solution y for x' where k is a constant. From y, by means of binary search techniques, we can find a 2^{16}-approximate solution for x' using $\lceil\log\lceil\log_{2^{16}}(k\log|x'|)\rceil\rceil \leq \log\lceil\log\log|x|^{kc}\rceil - 3 \leq \log\log\log|x| - 2$ adaptive queries to NP where the last inequality surely holds for sufficiently large $|x|$. Thus,

$$\text{FNP}^{\text{NP}[\log\log\log n-1]} \subseteq \text{FNP}^{\text{NP}[\log\log\log n-2]}$$

which implies the collapse of the polynomial-time hierarchy [42]. □

As a consequence of the above theorem and of the results of [28], we conjecture that the minimum set cover problem is not self-improvable.

REFERENCES

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, in Proc. 33rd IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 14–23.

[2] J. L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity* I, Springer-Verlag, Berlin, New York, 1988.

[3] R. Beigel, *Bounded queries to SAT and the Boolean hierarchy*, Theoret. Comp. Sci., 84 (1991), pp. 199–223.

[4] P. Berman, and G. Schnitger, *On the complexity of approximating the independent set problem*, Inform. and Comput., 96 (1992), pp. 77–94.

[5] D. P. Bovet and P. Crescenzi, *Introduction to the Theory of Complexity*, Prentice Hall, London, 1994.

[6] L. Cai, *Nondeterminism and Optimization*, Ph.D. thesis, Department of Computer Science, Texas A&M University, College Station, TX, 1994.

[7] R. Chang, *On the query complexity of clique size and maximum satisfiability*, in Proc. 9th IEEE Structure in Complexity Theory Conf., IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 31–42.

[8] R. Chang, *A machine model for NP-approximation problems and the revenge of the Boolean hierarchy*, EATCS Bulletin, 54 (1994), pp. 166–182.

[9] R. Chang, W. I. Gasarch, and C. Lund, *On Bounded Queries and Approximation*, Technical Report TR CS-94-05, Department of Computer Science, University of Maryland Baltimore County, Baltimore, MD, 1994.

[10] S. A. Cook, *The complexity of theorem proving procedures*, in Proc. 3rd ACM Symp. on Theory of Computing, ACM, New York, 1971, pp. 151–158.

[11] P. Crescenzi, *A short guide to approximation preserving reductions*, in Proc. 12th IEEE Conference on Computational Complexity, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 262–273.

[12] P. Crescenzi and V. Kann, *A Compendium of NP Optimization Problems*, Technical Report SI/RR-95/02, Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza," http://www.nada.kth.se/theory/compendium/ (1995).

[13] P. Crescenzi and V. Kann, *Approximation on the web: a compendium of NP optimization problems*, in Proc. Int. Workshop on Randomization and Approximation Techniques in Computer Science, Lecture Notes in Comput. Sci. 1269, Springer-Verlag, New York, 1997, pp. 111–118.

[14] P. Crescenzi and A. Panconesi, *Completeness in approximation classes*, Inform. and Comput., 93 (1991), pp. 241–262.

[15] P. Crescenzi and R. Silvestri, *Relative complexity of evaluating the optimum cost and constructing the optimum for maximization problems*, Inform. Process. Lett., 33 (1990), pp. 221–226

[16] P. Crescenzi and L. Trevisan, *On approximation scheme preserving reducibility and its applications*, in Proc. 14th Conf. on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 880, Springer-Verlag, New York, 1994, pp. 330–341.

[17] M. Fürer and B. Raghavachari, *Approximating the minimum degree spanning tree to within one from the optimal degree*, in Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms, SIAM, Philadelphia, 1992, pp. 317–324.

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

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

[20] I. Holyer, *The NP-completeness of edge-coloring*, SIAM J. Comput., 10 (1981), pp. 718–720.

[21] R. Impagliazzo and M. Naor, *Decision trees and downward closures*, in Proc. 3rd IEEE Structure in Complexity Theory Conf., IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 29–38.

[22] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.

[23] J. Kadin, *The polynomial time hierarchy collapses if the Boolean hierarchy collapses*, SIAM J. Comput., 17 (1988), pp. 1263–1282.

[24] J. Kadin, *Erratum: The polynomial time hierarchy collapses if the Boolean hierarchy collapses*, SIAM J. Comput., 20 (1991), p. 404.

[25] V. Kann, *On the Approximability of NP-Complete Optimization Problems*, Ph.D. thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1992.

[26] V. Kann, *Polynomially bounded minimization problems that are hard to approximate*, Nordic J. Comput., 1 (1994), pp. 317–331.

[27] N. Karmarkar and R. M. Karp, *An efficient approximation scheme for the one-dimensional bin packing problem*, in Proc. 23rd IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1982, pp. 312–320.

[28] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani, *On syntactic versus computational views of approximability*, in Proc. 35th IEEE Symp. on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 819–830.

[29] P. G. Kolaitis and M. N. Thakur, *Approximation properties of NP minimization classes*, in Proc. 6th IEEE Structure in Complexity Theory Conf., IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 353–366.

[30] M. W. Krentel, *The complexity of optimization problems*, J. Comput. System Sci., 36 (1988), pp. 490–509.

[31] R. E. Ladner, *On the structure of polynomial-time reducibility*, J. ACM, 22 (1975), pp. 155–171.

[32] T. J. Long, *On* γ*-reducibility versus polynomial time many-one reducibility*, Theoret. Comput. Sci., 14 (1981), pp. 91–101.

[33] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41 (1993), pp. 960–981.

[34] R. Motwani, *Lecture Notes on Approximation Algorithms*, Technical Report STAN-CS-92-1435, Department of Computer Science, Stanford University, Stanford, CA, 1992.

[35] P. Orponen and H. Mannila, *On Approximation Preserving Reductions: Complete Problems and Robust Measures*, Technical Report C-1987-28, Department of Computer Science, University of Helsinki, Helsinki, Finland, 1987.

[36] A. Panconesi and D. Ranjan, *Quantifiers and approximation*, Theoret. Comput. Sci., 107 (1993), pp. 145–163.
[37] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.
[38] C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.
[39] M. Queyranne, *Bounds for assembly line balancing heuristics*, Oper. Res., 33 (1985), pp. 1353–1359.
[40] U. Schöning, *Graph isomorphism is in the low hierarchy*, in Proc. 4th Symp. on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 247, Springer-Verlag, New York, 1986, pp. 114–124.
[41] H. U. Simon, *Continuous reductions among combinatorial optimization problems*, Acta Inform., 26 (1989), pp. 771–785.
[42] K. Wagner, *Bounded query computations*, in Proc. 3rd IEEE Structure in Complexity Theory Conf., IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 260–277.
[43] T. S. Wee and M. J. Magazine, *Assembly line balancing as generalized bin packing*, Oper. Res. Lett., 1 (1982), pp. 56–58.
[44] M. Yannakakis, *On the approximation of maximum satisfiability*, J. Algorithms, 17 (1994), pp. 475–502.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1783–1805

AN EFFICIENT DATA STRUCTURE FOR LATTICE OPERATIONS*

MAURIZIO TALAMO† AND PAOLA VOCCA‡

Abstract. In this paper, we consider the representation and management of an element set on which a lattice partial order relation is defined.

In particular, let n be the element set size. We present an $O(n\sqrt{n})$-space *implicit* data structure for performing the following set of basic operations:

1. Test the presence of an order relation between two given elements, in constant time.
2. Find a path between two elements whenever one exists, in $O(l)$ steps, where l is the path length.
3. Compute the successors and/or predecessors set of a given element, in $O(h)$ steps, where h is the size of the returned set.
4. Given two elements, find all elements between them, in time $O(k \log d)$, where k is the size of the returned set and d is the maximum in-degree or out-degree in the transitive reduction of the order relation.
5. Given two elements, find the least common ancestor and/or the greatest common successor in $O(\sqrt{n})$-time.
6. Given k elements, find the least common ancestor and/or the greatest common successor in $O(\sqrt{n} + k \log n)$time. (Unless stated otherwise, all logarithms are to the base 2.)

The preprocessing time is $O(n^2)$. Focusing on the first operation, representing the building-box for all the others, we derive an overall $O(n\sqrt{n})$-space $\times$ time bound which beats the order n^2 bottleneck representing the present complexity for this problem. Moreover, we will show that the complexity bounds for the first three operations are optimal with respect to the worst case. Additionally, a stronger result can be derived. In particular, it is possible to represent a lattice in space $O(n\sqrt{t})$, where t is the minimum number of disjoint chains which partition the element set.

Key words. data structure, lattices, reachability, least common ancestors, graph decomposition

AMS subject classifications. 06B99, 68P05, 68P25, 68R10

PII. S0097539794274404

1. Introduction.

The study of partial orders (*posets*) efficient representation, from either space or query time point of view, has been extensively tackled in the last few years, as it is a basic problem in many fields of applications. For instance, posets representation is needed when the elements are points in the d-dimensional space over which we wish to perform a dominance test or range query (computational geometry [16]); when the elements are sets and we want to perform a containment or intersection query (knowledge bases [1, 2]; object-oriented and semantic data models [19]); when the elements are vertices of a directed acyclic graph over which binary relations are defined (taxonomy, graph traversal, distributed computing [25, 26], etc.), and when we wish to perform the reachability test.

In general, posets are involved in all applications dealing with traversing sets of items over which an order relation is defined [28, 18, 33, 9].

When the order relation is complex, for example when the order dimension is proportional to the size of the element set, then these problems cannot be efficiently

*Received by the editors September 19, 1994; accepted for publication (in revised form) May 22, 1998; published electronically May 13, 1999. This work was supported by the Italian Authority for Public Administration and the ESPRIT Basic "Research Action on Algorithms and Data Structure" 20244 (ALCOM-IT).

http://www.siam.org/journals/sicomp/28-5/27440.html

†Italien Authority for Public Administration and Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza," Via Salaria 113, I-00198 Rome, Italy (talamo@dis.uniroma1.it).

‡Dipartimento di Matematica, Università di Roma "Tor Vergata," Via della Ricerca Scientifica, I-00133 Rome, Italy (vocca@axp.mat.uniroma2.it).

solved using the well-known techniques.

In this paper we show that for partial lattices it is possible to efficiently compute a set of basic operations. In particular, adopting a graph theoretic notation, given a directed acyclic graph (*dag*) $G = (V, E)$, we present an *implicit* data structure for efficiently performing the following operations:

1. REACHABILITY(x, y). Given $x, y \in V$, tests the presence of a directed path from x to y; returns true if such a path exists, false otherwise.
2. PATH(x, y). Given $x, y \in V$, returns a path from x to y, if at least one such path exists.
3. SUCC(x). Given $x \in V$, returns the set of all successors of x.
4. PRED(x). Given $x \in V$, returns the set of all predecessors x.
5. RANGE(x, y). Given $x, y \in V$, returns all vertices in all the directed paths connecting x to y.
6. LUB(x, y). Given $x, y \in V$, returns the least upper bound between x and y.
7. GLB(x, y). Given $x, y \in V$, returns the greatest lower bound between x and y.
8. LUB$(x_1, \ldots, x_k)$. Given $x_1, \ldots, x_k \in V$, returns the least upper bound of $x_1, \ldots, x_k$.
9. GLB$(x_1, \ldots, x_k)$. Given $x_1, \ldots, x_k \in V$, returns the greatest lower bound of $x_1, \ldots, x_k$.

The idea is to minimize the storage complexity while efficiently performing the above operations. In particular, the aim is to be able to test REACHABILITY in $O(1)$. REACHABILITY captures connectivity information and hence is a basic operation for digraphs management.

Without loss of generality, we prove all theorems under the assumption that $G = (V, E)$ is connected; hence if $|V| = n$ and $|E^T| = m$, where E^T is the edge set of the transitive reduction graph, then $m \geq n - 1$. In the case of nonconnected *dags*, all our results can be applied to each connected component without any change in complexity bounds.

Although a lot of work has been done in this field, it is still an open problem of how to represent partial order relations with a worst case time $\times$ space complexity less than $o(n^2)$. This problem is difficult because of "naive" representations, such as explicitly representing the whole transitive closure G^*, at one extreme, or testing reachability on G by means of DFS traversal, for example, at the other. Both require $\Omega(n^2)$ time $\times$ space in the worst case.

There is an additional reason for the importance of this open problem: understanding the complexity of directed graph versus undirected graph traversal. In fact, there was considerable empirical evidence (in terms of the efficiency of algorithms that have been discovered) that reachability in directed graphs is "harder" than reachability in undirected graphs. This has been proven to be substantially true in [4]. For instance, although directed graphs can be traversed nondeterministically in polynomial time and logarithmic space simultaneously, there is evidence that they cannot be traversed deterministically in polynomial time and small space simultaneously [31]. In contrast, the undirected s-t connectivity can be performed deterministically in polynomial time and sublinear space [6], and undirected graphs can be generally traversed in polynomial time and logarithmic space probabilistically by using a random walk [5, 10]; this implies similar resource bounds on (nonuniform) deterministic algorithms [5]. Moreover, using simple techniques, efficient data structures for undirected graphs have been developed for the dynamic maintenance of several graph properties [14].

The gap in efficiency between directed graphs and nondirected graphs for reachability problem management is mainly due to the nonsymmetry of the reachability relation. Moreover, the harder case is when the graph is acyclic, in the sense that the problem for general graphs can be reduced in linear time to the acyclic case. In fact, we can partition the vertex set of the graph into strongly connected components (e.g., using depth-first search) and then shrink the components to form an acyclic graph (see [3]).

Therefore, efficient solutions to the reachability problem have been found for special classes of *dags* [17, 26, 30] exploiting the order dimension property of the associated posets [13]. In this context, the partial orders considered are those having a constant order dimension [21]. Informally speaking, a partial order having order dimension k, with $k \geq 1$ a constant value, can be represented using $\Theta(kn)$ space, where n is the element set size, while testing partial order relationship in $O(k)$ time.

Unfortunately, since posets can have nonconstant order dimension, this technique cannot be extended to general digraphs [24]. In particular, this is true for partial lattices. Another approach that was studied, allowing derivation of an efficient representation for interval orders and a distributive lattice, is based on the representation of a poset by subsets of an n set [23, 8].

Hence, although there are strategies for dealing with restricted classes of *dags* optimal for time $\times$ space complexity, there is no uniform approach for general directed graphs. This is due mainly to intrinsic deficiencies of the general schema adopted by most of the widely accepted strategies for reachability problem resolution. In fact, the key idea is to decompose a *dag* G into a collection $\mathcal{F}$ of sparse *dags* G_i representing a covering of the graph G^*, the transitive closure of G.

In particular, for any *dag* $G = (V, E)$, a collection $\mathcal{F} = \{G_1, \ldots, G_k\}$ must satisfy the following property:

(i) $\langle x, y\rangle \in G^* \Leftrightarrow \exists G_i \in \mathcal{F}$ such that $\langle x, y\rangle \in G_i^*$.

This property is nice for the design of efficient algorithms as it allows us to search locally only and not in the whole graph. On the other hand, using this approach presents two main problems.

The first problem is how to bound the overall space complexity. In fact, a trivial solution to the above approach is to represent a *dag* G by means of an n-element collection $\mathcal{F}$, with each element representing the subgraph induced by all vertices connected to a given vertex. However, it can be easily seen that this approach requires $O(n^2)$ space.

The second problem is how to derive connectivity information in an efficient way with respect to the time complexity. In fact, a "naive" application of the decomposition technique, optimal from the space complexity point of view, is based on a one-element set $\mathcal{F}$, containing the transitive reduction of G. Unfortunately, in this case, the time complexity is $O(m)$.

In this paper, we present a two-level decomposition strategy, previously introduced in [28, 29] but now extended to deal with a more general set of operations, which balances time and space complexities.

In particular, we will prove the following.

THEOREM 1.1. *Let $G = (V, E)$ be a dag satisfying lattice property, with $n = |V|$. An $O(n\sqrt{n})$-space implicit data structure exists allowing us to perform the following:*

1. REACHABILITY(x, y) *in time $O(1)$;*
2. PATH(x, y) *in time $O(l)$, where l is the path length;*
3. SUCC(x) *in time $O(h)$, where h is the size of the returned set;*

4. PRED(x) *in time* $O(h)$*, where* h *is the size of the returned set;*

5. RANGE(x, y) *in time* $O(k \log d)$*, where* k *is the size of the returned set and* d *is the maximum in-degree or out-degree in the transitive reduction graph;*

6. LUB(x, y) *in time* $O(\sqrt{n})$*;*

7. GLB(x, y) *in time* $O(\sqrt{n})$*;*

8. LUB$(x_1, \ldots, x_k)$ *in time* $O(\sqrt{n} + k \log n)$*;*

9. GLB$(x_1, \ldots, x_k)$ *in time* $O(\sqrt{n} + k \log n)$*;*

Moreover, the preprocessing time is $O(n^2)$ *and all bounds are worst case.*

Results of Theorem 1.1 are a bit surprising. In fact, in some cases, e.g., when $|E| = \Omega(n\sqrt{n})$, our decomposition technique makes a compression of $G = (V, E)$. It is worth noting that the complexity bounds obtained do not break any information theoretic lower bound and are optimal with respect to the worst case, as shown in the following proposition.

PROPOSITION 1.1 (see [22]). *Let* $\mathcal{L}(n)$ *be the number of labeled lattices with* n *elements. Then*

$$\mathcal{L}(n) < \alpha^{(n\sqrt{n}+o(n\sqrt{n}))},$$

where α *is a constant (about* 6.11343*).*

Furthermore, starting from Theorem 1.1, a stronger result can be derived. In particular, it is possible to represent a partial lattice in space $O(n\sqrt{t})$ with the same time bound for all the operations of Theorem 1.1, where t is the minimum number of disjoint chains which partition the element set. This is an important result, since it provides a tight characterization of the complexity of partial lattices. In fact, based on Dilworth's theorem [12], t is equal to the width of the lattice.

As will be evident in the following, the proposed data structure cannot be straightforwardly applied to general digraphs. Therefore, in this case, it could represent a good heuristic method for testing reachability which takes into account the sparseness of the graph.

The paper is organized as follows. In section 2, some basic definitions are given; in section 3, both the basic decomposition strategy and the data structure are presented; in section 4, using a classification of the vertex set, the final version of the decomposition strategy is given; in section 5, an extended version of the data structure is given and the overall space complexity is analyzed; in section 6, time bounds for all operations of the main theorem are stated; finally, in section 7, future research is described.

2. Preliminaries. In this section, we briefly describe the notation used and give some basic definitions useful for the following. More definitions on graphs and posets can be found in textbooks such as [7, 11, 15].

Directed graphs are denoted $G = (V, E)$, where V is the set of *vertices* or *elements* and E is the set of *edges* or *arcs*. Whenever the vertex set is not explicitly defined, it is denoted $Vert(G)$.

In a directed graph, the *in-degree* of a vertex x is the number of edges directed towards x, denoted $Indeg(x)$. Analogously, the *out-degree* of x ($Outdeg(x)$) is the number of edges directed away from v.

A partially ordered set (poset) $\mathcal{P} = (\prec, \mathcal{N})$ is an irreflexive, asymmetric, and transitive relation on the element set $\mathcal{N}$. We denote $\preceq$ the reflexive relation associated with $\mathcal{P}$. Two elements x, y are *incomparable* (denoted $x \sim y$), if neither $x \preceq y$ nor $y \preceq x$.

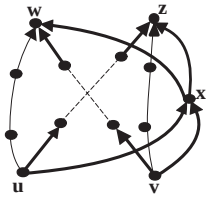


FIG. 1. *Lattice property.*

An element $z \in \mathcal{N}$ is an *upper bound* of $x, y \in \mathcal{N}$ if $x \preceq z$ and $y \preceq z$. The element z is called the *least upper bound* of x and y, denoted $\text{LUB}(x, y)$, if $z \preceq w$ for all upper bounds w of x and y. The *greatest lower bound*, denoted $\text{GLB}(x, y)$, is defined dually.

A *lattice* is a partial order $\mathcal{L} = (\prec, \mathcal{N})$ such that every two $x, y \in \mathcal{N}$ have both a least upper bound and a greatest lower bound.

For a given partial order $\mathcal{P} = (\prec, \mathcal{N})$, it is always possible to find a *linear extension* of $\mathcal{P}$, i.e., a total order which is consistent with $\mathcal{P}$. There always exists such a linear extension, and the intersection of all linear extensions of $\mathcal{P} = (\prec, \mathcal{N})$ is $\mathcal{P} = (\prec, \mathcal{N})$. A minimal set $\mathcal{R}$ of linear extensions of $\mathcal{P} = (\prec, \mathcal{N})$ whose intersection is $\mathcal{P} = (\prec, \mathcal{N})$ is called a *realizer* of $\mathcal{P} = (\prec, \mathcal{N})$, and the *order dimension* of $\mathcal{P} = (\prec, \mathcal{N})$ is the minimum cardinality of any realizer of $\mathcal{P} = (\prec, \mathcal{N})$.

Given a poset $\mathcal{P} = (\prec, \mathcal{N})$, its *st-completion* is the poset obtained by adding to $\mathcal{N}$ two elements labeled s and t and extending $\prec$ with the following order relations: $s \prec x$ and $x \prec t \ \forall x \in \mathcal{N}$.

Posets whose st-completion is a lattice are *partial lattices*. Other authors refer to this class of poset as *truncated lattices* [27].

Given a directed acyclic graph (*dag*) $G = (V, E)$, the *associated partial order* is the poset $\mathcal{P}_G = (\prec, V)$ such that, for all $u, v \in V$, $u \prec v$ if and only if $\langle u, v \rangle \in E^*$, where E^* is the edge set of the transitive closure graph $G^* = (V, E^*)$.

DEFINITION 2.1. *A dag $G = (V, E)$ satisfies the* lattice property *if and only if the associated partial order is a partial lattice.*

From the definition, the following property trivially follows and will be useful for the following (see Figure 1).

PROPOSITION 2.1. *A dag $G = (V, E)$ satisfies the lattice property if and only if, for every four vertices $(u, v, z, w) \in V$, if four directed paths exist, pairwise disjoint except for at most the extremal vertices, having as endpoints pairs of vertices $\langle u, w \rangle$, $\langle u, z \rangle$, $\langle v, w \rangle$, and $\langle v, z \rangle$, then there exist a vertex x and four paths having as endpoints pairs of nodes $\langle u, x \rangle$, $\langle v, x \rangle$, $\langle x, z \rangle$, and $\langle x, z \rangle$.*

Obviously, partial lattices always satisfy the above property.

3. Decomposition strategy: Basic version. Regarding what was said in the introduction, a general problem of a decomposition technique is the choice of the decomposition criteria. Our key idea is to use a two-stage decomposition, with each stage having its own decomposition criteria.

In particular, given a *dag* $G = (V, E)$ satisfying the lattice property, at the first stage we partition V in a sequence of sets, or *clusters*, denoted $Clus(c)$, where $c \in V$ is a representative vertex.

At the second stage, for each cluster $Clus(c)$, we choose a suitable collection of digraphs (*double tree* or $DT(u, c)$), representing all the connectivity information between vertices in $Clus(c)$ and vertices in $V - Clus(c)$.

Each double tree collection associated with a cluster represents the basic element of the proposed decomposition strategy.

For every two vertices x, y, with $x \in Clus(c)$, the corresponding collection $\{DT(u,c)|u \in Clus(c)\}$ satisfies the following property: x is connected to y in G if and only if at least one $DT(u_i,c) \in \{DT(u,c)\}$ exists such that x is connected to y in $DT(u_i,c)$.

As we will show in section 4, operating on the cluster size it is possible to bound the overall space complexity, while realizing, with the second level decomposition, a constant reachability test.

For the sake of simplicity, we first illustrate the decomposition strategy and the corresponding data structure with no constraint on cluster size.

3.1. Clusters. We now give a formal definition of a cluster and state its main properties.

DEFINITION 3.1. *Given a vertex $c \in V$, let $Clus^+(c) = \{Pred(c) \cup c\}$ and $Clus^-(c) = \{Succ(c) \cup c\}$, where $Pred(c)$ and $Succ(c)$ are the sets of predecessors and successors of c. A* cluster *is either $Clus^+(c)$ or $Clus^-(c)$ for some $c \in V$.*

In the following, when no confusion is possible, we will use $Clus(c)$ to denote either $Clus^+(c)$ or $Clus^-(c)$.

Let $v \in V - Clus(c)$. The following lemma shows an important relationship between two different clusters, $Clus(c)$ and $Clus(v)$.

LEMMA 3.1. *If $Clus^+(c) \cap Clus^+(v) = \mathcal{I}$, then $\text{LUB}(\mathcal{I}) \in \mathcal{I}$. Dually, if $Clus^-(c) \cap Clus^-(v) = \mathcal{I}$, then $\text{GLB}(\mathcal{I}) \in \mathcal{I}$.*

Proof. By the cluster definition, for any two elements $x, y \in Clus^+(c)$, $\text{LUB}(x,y) \in Clus^+(c)$. In fact, assume for contradiction that $\text{LUB}(x,y) \notin Clus^+(c)$. Then the following three conditions simultaneously hold:

(i) $x \prec c$,
(ii) $y \prec c$,
(iii) $\text{LUB}(x,y) \sim c$.

This contradicts Proposition 2.1.

In order to prove the lemma we have to show that $Clus^+(c) \cap Clus^+(v)$ does not have two distinct maximal elements. Let us now assume that two maximal elements $x, y \in Clus^+(c) \cap Clus^+(v)$ exist, with $x \neq y$. Then we have

(i) $\text{LUB}(x,y) \in Clus^+(c)$,
(ii) $\text{LUB}(x,y) \notin Clus^+(v) \Rightarrow \text{LUB}(x,y) \sim v$.

However, this once again contradicts Proposition 2.1. A similar reasoning holds for $Clus^-(c) \cap Clus^-(v)$. The lemma is so proved by contradiction. ∎

LEMMA 3.2. *Let $G = (V,E)$ be a dag satisfying the lattice property and $Clus(c)$ a cluster. Then $G' = (V - Clus(c), E')$ is a dag satisfying the lattice property, where E' is the set of edges of the subgraph induced by $V - Clus(c)$.*

Proof. The proof is an obvious consequence of the cluster definition. ∎

3.2. Double trees. Given a cluster $Clus(c)$, let us refer to vertices in $Clus(c)$ and in $V - Clus(c)$ as *internal* and *external vertices*, respectively. In particular, we denote by $Ext(Clus(c))$ the set of all external vertices *connected* to at least one internal vertex.

The problem of maintaining all the connectivity information related to a given cluster can be split into the following two subproblems: (i) the representation of the connectivity relationships between internal vertices and (ii) the representation of the connectivity relationships between internal and external vertices.

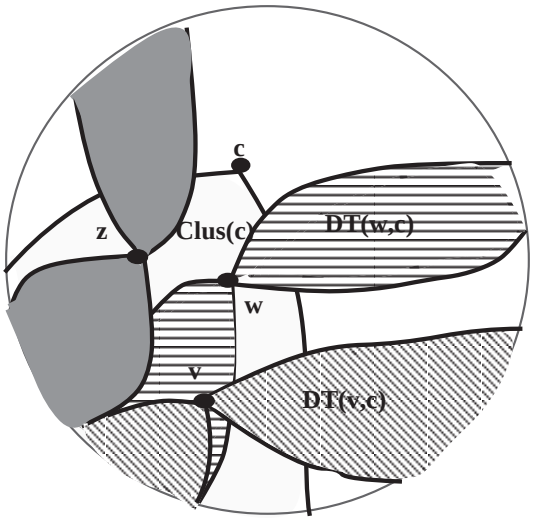


FIG. 2. *Double tree decomposition.*

We solve the first problem by computing for each internal vertex u a spanning tree of the graph induced by the set $Pred(u) \cap Clus(c)$ or by the set $Succ(u) \cap Clus(c)$, depending on whether $Clus(c)$ is a $Clus^+(c)$ or a $Clus^-(c)$. Hence, to each internal vertex $u \in Clus(c)$ there is associated a tree rooted at u, the *internal tree induced by* u, denoted $IntTree(u, c)$.

For the second problem, from Lemma 3.1, given an external vertex v, the pair $(v, Clus(c))$ univocally identifies a vertex $u \in Clus(c)$ representing either the LUB$(Clus^+(c) \cap Clus^+(v))$ or the GLB$(Clus^-(c) \cap Clus^-(v))$. This implies that, given a cluster $Clus(c)$ for each external vertex v connected to at least one vertex in $Clus(c)$, an internal vertex u, the *internal representative of the external vertex* v exists, univocally identifying the internal tree $IntTree(u, c)$ made up of all internal vertices connected to v.

Let $Ext(u)$ be the set of external vertices having u as an internal representative vertex. Obviously, we have

$$Ext(Clus(c)) = \bigcup_{u \in Clus(c)} Ext(u).$$

For each set $Ext(u)$, we compute a spanning tree, rooted at u, of the subgraph induced by $Ext(u)$. We refer to this tree as the *external tree induced by* u, denoted $ExtTree(u, c)$ (see Figure 2). The external trees in $\{ExtTree(u, c) | u \in Clus(c)\}$ have the nice property of being pairwise disjoint, as shown in the following lemma.

LEMMA 3.3. *Let* $v, w \in Clus(c)$*, with* $v \neq w$*. Then* $ExtTree(v, c) \cap ExtTree(w, c) = \emptyset$.

Proof. Let us assume for contradiction that $y \in ExtTree(v, c) \cap ExtTree(w, c)$. By the external tree definition, y is associated with both v and w. This contradicts the uniqueness of the representative vertex stated in Lemma 3.1. The proof follows by contradiction. □

DEFINITION 3.2. *Given a cluster* $Clus(c)$ *and the two collections* $\{IntTree(u, c)\}$ *and* $\{ExtTree(u, c)\}$ *of internal and external trees, for each* $u \in Clus(c)$ *a* double tree

is defined as

$$DT(u,c) = IntTree(u,c) \cup ExtTree(u,c).$$

A double tree represents the basic decomposition subgraph and, informally speaking, is the union of an internal tree and the external tree rooted at the same internal vertex, whenever the internal and external trees exist.

Each double tree $DT(u,c)$ is associated with a partial order having order dimension 2, because its *st*-completion is a planar poset with one greatest element and one least element [32]. The first consequence of the above property is that $DT(u,c)$ is representable with two linear extensions L_1 and L_2; that is, given two vertices $x, y \in DT(u,c)$, y is reachable from x if and only if $x < y$ in both linear extensions. In particular, two labels (*coordinates*) (x_1, x_2) are associated with any vertex $x \in DT(u,c)$, each one representing an x position within the first and second linear extensions, respectively.

The proof of the following proposition can be found in [20].

PROPOSITION 3.1. *Given* $x, y \in DT(u,c)$, REACHABILITY$(x,y) =$ **true** *in* $DT(u,c)$ *if and only if* $(x_1, x_2) < (y_1, y_2)$.

From the above proposition Corollary 3.4 easily follows.

COROLLARY 3.4. *Given a double tree* $DT(u,c)$, *such that* $|Vert(DT(u,c))| = K$, *an* $O(K)$*-space* implicit *data structure exists for performing a* REACHABILITY *test in* $O(1)$*-time.*

3.3. Basic decomposition strategy. The decomposition strategy we propose first generates a collection of clusters and then, for each cluster, builds the corresponding decomposition elements (double tree collection).

```
    procedure BasicBuildClusters;
1.  begin;
2.  C := ∅;        {Cluster Collection}
3.  while V ≠ ∅ do
4.      begin;
5.      Choose a cluster Clus(c);  {either Clus+(c) or Clus−(c)}
6.      C := C ∪ Clus(c);
7.      V := V − Clus(c);
8.      end;
9.  return C;
10. end.
```

The cluster collection returned is the input to the following procedure which builds, for each cluster, the associated double trees collection representing the decomposition elements on which we base our data structure.

```
    procedure BasicDecomposition (C : Cluster Collection);
1.  for each Clus(c) ∈ C;
2.  begin
3.      for each u ∈ Clus(c) Build {DT(u,c)};
4.      return Clus(c) and {DT(u,c)};
        {Returns the current cluster and double tree collection}
5.  end;
```

Hence, a *dag* $G = (V, E)$ induces two collections of subgraphs,

$\mathcal{C} = \langle Clus(c_1), \ldots, Clus(c_k)\rangle$,

$\mathcal{T} = \langle \{DT(u_{1,j}, c_1)\}, \ldots, \{DT(u_{k,j}, c_k)\}\rangle$ for all $u_{i,j} \in Clus(c_i)$,

satisfying the following invariants. Let $x \in V$.

(i) x belongs to one and only one cluster $Clus(c_i)$ (for construction, see line 7 of the `BasicBuildClusters` procedure);

(ii) given a cluster $Clus(c_j)$ different from the one to which x belongs according to (i), x belongs to at most one $DT(u_{j,i}, c_j)$ as an external vertex (Lemma 3.3);

(iii) given a cluster $Clus(c_i)$, each element of the collection $\{DT(u_{i,j}, c_i)\}$ is, by definition, univocally identified by the element $u_{i,j} \in Clus(c_i)$.

To prove the correctness of the proposed strategy we show that the double tree collection is a covering of the given graph.

THEOREM 3.5. *Given a dag* $G = (V, E)$ *satisfying the lattice property and* $x, y \in V$, REACHABILITY$(x, y) =$ **true** *in* $G = (V, E)$ *if and only* REACHABILITY$(x, y) =$ **true** *in* $DT(u, c)$, *for at least one* $DT(u, c) \in \mathcal{T}$.

Proof. ($\Rightarrow$) First, note that by Lemma 3.2 the graph $G' = (V - Clus(c), E')$ still satisfies the lattice property. Let REACHABILITY$(x, y) =$ **true** in $G = (V, E)$. We show that the decomposition strategy returns a double tree $DT(u, c) \in \mathcal{T}$ such that REACHABILITY$(x, y) =$ **true** in $DT(u, c)$. Note that, by construction, x and y must belong to one and only one cluster. Two different cases are possible according to which cluster they belong.

1. $x, y \in Clus(c)$.

If $Clus(c)$ is a $Clus^+(c)$, then line 3 of the `BasicDecomposition` procedure assures that x belongs to $IntTree(y, c)$, and hence $x \in DT(y, c)$. On the other hand, if $Clus(c)$ is a $Clus^-(c)$, then $y \in DT(x, c)$.

2. $x \in Clus(c_1)$ and $y \in Clus(c_2)$.

Without loss of generality, let us suppose that the decomposition algorithm first generates $Clus(c_1)$ and then $Clus(c_2)$. Since y is connected to a vertex in $Clus(c_1)$; it then has an internal representative vertex, say, w. Hence $y \in ExtTree(w, c_1)$. By Proposition 2.1, either $x \in IntTree(w, c_1)$ or $x = w$. Thus, REACHABILITY$(x, y) =$ **true** in $DT(w, c_1)$.

($\Leftarrow$) This part of the proof obviously follows from observing that the decomposition algorithm does not add any new edge. ∎

3.4. Basic data structure. In this section we briefly describe the basic data structure for representing a *dag* satisfying the lattice property and describe how to perform the REACHABILITY operation, since this is the basic operation for performing all the others.

From the previously described invariants and from applying Proposition 3.1, we derive the following simple implicit data structure based on look-up tables (see Figure 3).

The data structure is composed of one look-up table indexed on elements in V (data structure **A**) and two sets of look-up tables (data structures **B** and **C**).

Data structure **A** stores for each vertex the unique identifier of $Clus(c_i)$ to which it belongs (see invariant (i) above) and its sign ("+" or "−" for $Clus^+(c)$ or $Clus^-(c)$, respectively).

Data structure **B** is a set of look-up tables, each table associated with a vertex $x \in V$. For each double tree $DT(u, c_i)$ of the decomposition induced by the cluster $Clus(c_i)$ to which x belongs, data structure **B** stores x's coordinates with respect

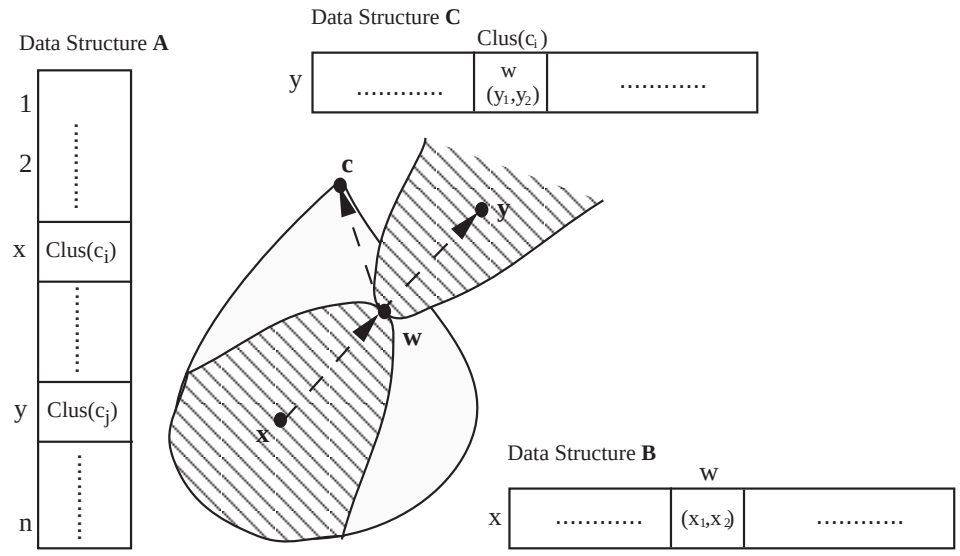


FIG. 3. *Data structure.*

to the representation of $DT(u, c_i)$, whenever x belongs to $DT(u, c_i)$; otherwise, it contains a null value.

Data structure **C** is a set of look-up tables, each table associated with an element $x \in V$. For each cluster $Clus(c_i)$ of the cluster collection $\mathcal{C}$, if $x \in Ext(Clus(c_i))$, then data structure **C** stores the identifier of the double tree associated with $Clus(c_i)$, to which x belongs as an external element, and stores x's coordinates with respect to this double tree representation.

Let us now describe how to perform REACHABILITY(x, y). From data structure **A** we derive the clusters to which x and y belong. Let $x \in Clus(c_i)$ and $y \in Clus(c_j)$. Two cases are possible.

(i) $Clus(c_i) = Clus(c_j)$.

In this case we look in the **B** table associated with x and y for their coordinates with respect to both: (1) the double tree rooted at x; (2) the double tree rooted at y. If, in one of the two double tree, x's coordinates are smaller than y's, then REACHABILITY$(x, y) =$ **true**; otherwise REACHABILITY$(x, y) =$ **false**.

(ii) $Clus(c_i) \neq Clus(c_j)$.

In this case, we first look in the table **C** associated with y, for the double tree identifier corresponding to cluster $Clus(c_i)$. Then we search in the **B** table associated with x, the coordinates of x with respect to this double tree. If x's coordinates are smaller than y's, then REACHABILITY$(x, y) =$ **true**; otherwise, we repeat the search, looking in the **C** table associated with x for the double tree identifier corresponding to cluster $Clus(c_j)$. Then we search in the table **B** associated with y, y's coordinates with respect to this double tree. If x's coordinates are smaller than y's, then REACHABILITY$(x, y) =$ **true**. If both searches fail, then REACHABILITY$(x, y) =$ **false** (see Figure 3).

From the above discussion it is possible to state the following lemma.

LEMMA 3.6. *The* REACHABILITY *operation can be performed in constant time.*

Obviously, for general clusters the space complexity is $O(n^2)$. Therefore, let us suppose, for instance, that the cluster collection $\mathcal{C}$ satisfies the following condition:

$$\forall\, Clus(c_i) \in \mathcal{C} \;\Rightarrow\; \frac{\sqrt{n}}{4} \leq |Clus(c_i)| \leq \frac{\sqrt{n}}{2} \quad .$$

In this case, it is trivial to show that the overall space occupancy is $O(n\sqrt{n})$. Unfortunately, this is a special case and, generally, this condition does not hold. Nevertheless, as shown in the following, it is possible to find a suitable cluster decomposition allowing us to keep the space occupancy within the required bound.

4. Decomposition strategy. As shown in section 3, a proper choice of cluster size permits us to bound the space complexity. We claim that a cluster size within $\frac{\sqrt{n}}{4}$ and $\frac{\sqrt{n}}{2}$ is, in fact, the answer to our problem because this size allows us to balance what is eliminated in one main iteration and what remains to be considered (see line 7 of the `BasicBuildClusters` procedure). When it is not possible to find clusters with the right size, we group them to form a *cluster forest.*

4.1. Elements classification. We need some more definitions.

DEFINITION 4.1. *A vertex $c \in V$ is*

1 good *if*

$$\frac{\sqrt{n}}{4} \leq |Clus^+(c)| \leq \frac{\sqrt{n}}{2} \;\; or \;\; \frac{\sqrt{n}}{4} \leq |Clus^-(c)| \leq \frac{\sqrt{n}}{2};$$

2 fat *if it is not good and if one of the following two conditions holds:*
- (i) $|Clus^+(c)| > \frac{\sqrt{n}}{2}$ *and* $\forall x \in Clus^+(c)$ *then* $|Clus^+(x)| < \frac{\sqrt{n}}{4}$*; or*
- (ii) $|Clus^-(c)| > \frac{\sqrt{n}}{2}$ *and* $\forall x \in Clus^-(c)$ *then* $|Clus^-(x)| < \frac{\sqrt{n}}{4}$*;*

3 thin *if it is neither good nor fat and if*

$$|Clus^+(c)| < \frac{\sqrt{n}}{4} \;\; or \;\; |Clus^-(c)| < \frac{\sqrt{n}}{4}.$$

We establish the correctness of our approach by first showing that it is always possible to find one of the above defined elements.

LEMMA 4.1. *Given a dag $G = (V, E)$ satisfying the lattice property, at least one good, fat, or thin vertex does exist which can be retrieved in time $O(n^2)$.*

Proof. Let us consider the following strategy. We first visit the graph, searching for good elements and meanwhile assigning a weight to each node to represent the size of the clusters (both $Clus^+$ and $Clus^-$) it induces. Whenever this search fails, we look for fat vertices.

If there is one node having weight greater than $\frac{\sqrt{n}}{2}$, then at least one fat vertex exists. In fact, without loss of generality, let c be a vertex such that $|Clus^+(c)| > \frac{\sqrt{n}}{2}$. We search the predecessors set of c for an element x inducing the smallest cluster satisfying condition $|Clus^+(x)| > \frac{\sqrt{n}}{2}$. If such a vertex x exists, then x is fat because there are no good vertices and, by a transitive property, none of its predecessors can induce a cluster of cardinality greater than $\frac{\sqrt{n}}{2}$. Otherwise, c is a fat vertex. If there are no fat vertices, then the vertex with the maximum weight is a thin vertex. Hence, this strategy always returns at least one element.

For the time complexity, let us consider a data structure for representing the graph which maintains for each vertex the number of its predecessors and an ordered list of

the number of predecessors of its immediate predecessors. With this data structure, which can be derived in time $O(n^2)$ by recursively visiting $G = (V, E)$, the above strategy can be implemented in time that is linear in the number of vertices. □

4.2. Cluster collection. Our aim is to show that, given a *dag* $G = (V, E)$, it is possible to build a cluster collection of clusters induced by either good vertices or thin vertices.

First we need to describe how to manage fat vertices.

LEMMA 4.2. *Given a fat vertex c, it is always possible to generate a sequence of clusters induced by good vertices which covers* $Clus(c)$.

Proof. Without loss of generality, let $|Clus^+(c)| > \frac{\sqrt{n}}{2}$. By definition, each one of c's immediate predecessors, $\langle c_1, \ldots, c_t \rangle$, satisfies the following condition:

$$|Clus^+(c_i)| < \frac{\sqrt{n}}{4} \quad for\ all\ 1 \leq i \leq t.$$

We can then group clusters induced by these vertices until the cardinality of each group is within $\frac{\sqrt{n}}{4}$ and $\frac{\sqrt{n}}{2}$.

Let $Clus(c_{i_1}, \ldots, c_{i_k})$, where $\langle c_{i_1}, \ldots, c_{i_k} \rangle \subset \langle c_1, \ldots, c_t \rangle$ is one group of clusters so obtained.

To prevent $Clus(c_{i_1}, \ldots, c_{i_k})$ from violating Lemma 3.1 (an external vertex y could be related to all $\langle c_{i_1}, \ldots, c_{i_k} \rangle$ through the fat vertex c), we add a *dummy good* vertex and the following directed edges (see Figure 4):

(i) $\langle d_i, c \rangle$;

(ii) $\langle c_{i_j}, d_i \rangle$ for $1 \leq j \leq k$.

By construction, the *dag* obtained by adding dummy vertices still satisfies the lattice property. Hence, to each fat vertex c there corresponds a collection $\{Clus(d_i)\}$ of clusters induced by dummy good vertices (see Figure 4) which covers $Clus^+(c)$. □

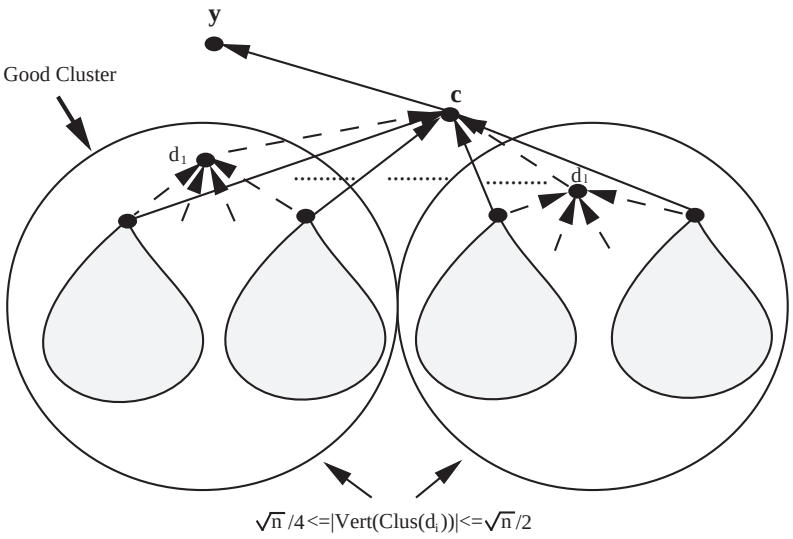


FIG. 4. *Fat node.*

The following procedure shows how our strategy chooses the required cluster collection. First, clusters induced by good vertices and by dummy good vertices

associated with fat vertices are chosen, then clusters induced by thin vertices are considered. More precisely, we have the following:

```
    procedure BuildClusters;
1.  begin;
2.  C := ∅;        {Cluster Collection}
3.  while V ≠ ∅ do
4.  begin;
5.      if  ∃c ∈ V s.t.  c is good then  C := C ∪ Clus(c);
6.      else  if  ∃c ∈ V s.t.  c is fat then
7.            begin
8.            Build the associated good cluster collection {Clus(d_i)};
9.            goto 5;
10.           end
11.       else
12.           begin
13.           Choose a thin vertex c s.t.  Clus(c) has maximum cardinality;
14.           C := C ∪ Clus(c);
15.           end;
16.     V := V − Clus(c);
17. end;
18. return  C;
19. end.
```

Let $\mathcal{C} = \langle Clus(c_1), \ldots, Clus(c_g), \ldots, Clus(c_{g+t})\rangle$ be the cluster collection so obtained, where for $1 \leq i \leq g$, $Clus(c_i)$ is a cluster induced by either actual or dummy vertices and for $g+1 \leq i \leq t$, $Clus(c_i)$ is induced by thin vertices.

To obtain the required space complexity, we group clusters induced by thin vertices. We define a *cluster forest* as any collection of clusters induced by thin vertices. Let $F = \langle Clus(c_1), \ldots, Clus(c_k)\rangle$ be a cluster forest. We define the *external vertex set of a forest* F, denoted $Ext(F)$, as the union of all the external vertex sets of each cluster composing the cluster forest; i.e.,

$$Ext(F) = \bigcup_{i=1}^{k} Ext(Clus(c_i)).$$

Note that according to our definitions, $F \cap Ext(F) \neq \emptyset$ in general. Therefore, for each cluster Lemma 3.3 still holds.

The following version of the decomposition procedure generates for each cluster in $\mathcal{C}$ the corresponding double tree collection. Moreover, it groups clusters induced by thin vertices into cluster forests.

Each forest $F = \langle Clus(c_1), \ldots, Clus(c_k)\rangle$ is generated by choosing clusters from the cluster collection in not increasing order of size until one of the following conditions holds:

1. $|F| \geq \frac{\sqrt{n}}{4}$, or
2. $mk \geq n$, where $m = |Ext(F)|$.

As we will see in section 5, condition 2 allows us to obtain the required space complexity.

```
    procedure Decomposition (C : Cluster Collection);
1.  for each Clus(c) ∈ C;
2.  begin
3.      for each u ∈ Clus(c) Build {DT(u, c)};
4.      return Clus(c) and {DT(u, c)};
5.  end;
6.  for each Clus(c) ∈ ⟨Clus(c_{g+1}), ..., Clus(c_{g+t})⟩;
        {the sequence is in not increasing order of size}
7.  begin
8.      F = ∅;
9.      k = 0;
10.     while |F| < √n/4 and mk < n.
11.     begin
12.         F = F ∪ Clus(c);
13.         k = k + 1;      {Number of clusters in a forest}
14.         m = |Ext(F)|;   {Number of external vertices}
15.         next Clus(c);
16.     end;
17.     return F;    {Returns the current forest}
18. end;
```

The above decomposition strategy returns a sequence $\langle F_1, \ldots, F_g, \ldots, F_{g+f} \rangle$ of clusters and cluster forests, where $F_i = Clus(c_i)$, for $1 \leq i \leq g$, is a cluster induced by either actual or dummy good vertices, while for $g+1 \leq i \leq f$, F_i is a cluster forest. Further, for each cluster the associated double tree collection is produced. Both collections are then inserted into the data structure. It is trivial to show that, even if clusters forests are considered, the double tree decomposition satisfies Theorem 3.5.

5. Data structure and space complexity. The clusters and cluster forests collection and the corresponding double tree decomposition returned by the `Decomposition` procedure satisfy the following invariants. Let $x \in V$:

(i) x belongs to one and only one F_i, where $1 \leq i \leq g + f$;

(ii) x belongs to one and only one cluster $Clus(c_{i,j})$;

(iii) given a cluster $Clus(c_{j,l})$ different from the one to which x belongs according to (ii), at most one $u \in Clus(c_{j,l})$ exists such that $x \in DT(u, c_{j,l})$;

(iv) given a cluster $Clus(c_{i,j})$, each element of the collection $\{DT(u, c_{i,j})\}$ is, by definition, univocally identified by the element $u \in Clus(c_{i,j})$.

To represent cluster forests, we modify the basic data structure described in section 3.3 to take into account the double indirection between a forest and its clusters (see Figure 5).

The new data structure **C** is again a set of look-up tables, each table associated with a vertex $x \in V$. For each forest F_i, if $x \in Ext(F_i)$, then data structure **C** maintains the identifier of a fourth kind of table, **D**, which stores connectivity information between x and F_i. If x is not connected to F_i, then it stores a null value.

Data structure **D** is a set of look-up tables, each table associated with a vertex x and a cluster forest F_i. Table $\mathbf{D}(F_i)$ exists if and only if $x \in Ext(F_i)$. For each cluster $Clus(c_{ij})$ in the cluster forest F_i, the corresponding field in the look-up table $\mathbf{D}(F_i)$ stores the identifier of the unique double tree associated with $Clus(c_{ij})$ to which x belongs as an external vertex and stores x's coordinates with respect to this double

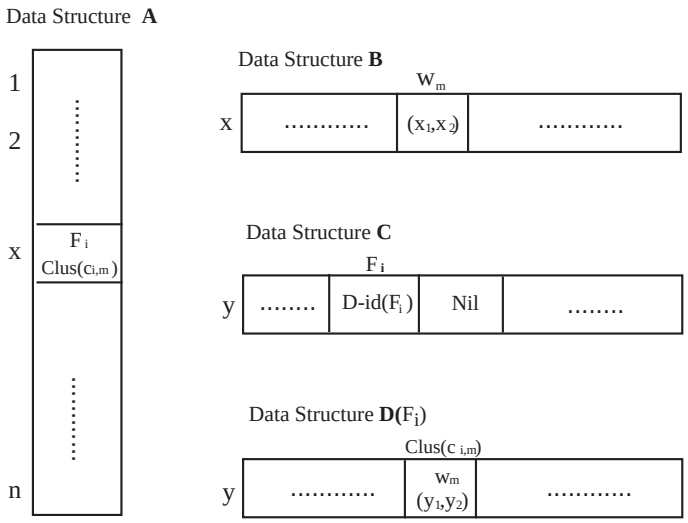


FIG. 5. *Extended Data Structure.*

tree representation.

Data structure **A** stores, for each vertex $x \in V$, the identifiers of both the cluster and forest to which it belongs, whenever they are different, and the cluster sign. Data structure **B** is the same as in section 3.3.

For the REACHABILITY test, it is easy to extend the basic strategy described in section 3.3 for the new data structure.

We now analyze the space complexity. With reference to the clusters and cluster forests sequence $\langle F_1, \ldots, F_g, \ldots, F_{g+f} \rangle$, by construction, the subsequence $\langle F_1, \ldots, F_g \rangle$, together with the corresponding double tree decomposition, requires $O(n\sqrt{n})$-space.

Let us now analyze the subsequence $\langle F_{g+1}, \ldots, F_{g+f} \rangle$ of cluster forests induced by thin vertices.

Recall that forests of clusters are generated choosing clusters in decreasing order until one of the following conditions holds:

1. $|F| \geq \frac{\sqrt{n}}{4}$, or
2. $mk \geq n$, where $m = |Ext(F)|$.

If we denote $m_i = |Ext(F_i)|$, then the overall space complexity of the **D** data structure is $O(\sum_{i=g+1}^{g+f} m_i k_i)$ since only vertices related to a cluster forest have the corresponding **D** look-up table, each table of size $O(k_i)$. Hence, the second condition is used to bound each term of the summation.

We now show that the decomposition strategy returns a number of cluster forests less than $\sqrt{n}$.

Obviously, if it is always possible to generate a forest satisfying both conditions, then the space complexity of the overall data structure is $O(n\sqrt{n})$. Unfortunately, the second condition could prevent us from generating an $O(\sqrt{n})$ collection of cluster forests; that is, each cluster forest could be of size less than $\frac{\sqrt{n}}{4}$. The following technical lemmas show how to manage this case.

First, it is important to underscore one property, shown in Lemma 5.1, of cluster forests useful for the following proofs.

LEMMA 5.1. *Let* $F = \langle Clus(c_1), \ldots, Clus(c_k)\rangle$ *be a cluster forest, where* $c_1, \ldots, c_k$ *are thin vertices, then*

$$|Clus(c_i)| = t \;\Rightarrow |Ext(Clus(c_i))| \leq t^2 \;\; \forall i \in \{1, \ldots, k\}\,.$$

Proof. The proof easily follows observing that, by construction, forests are generated only when there are neither good nor fat vertices. Moreover, each cluster is added to a forest in not increasing order of size. □

Without loss of generality, we denote the size of a cluster $Clus(c_{ij}) \in F_i$ as follows:

$$|Clus(c_{ij})| = n^{\frac{1}{2} - \sum_{p=1}^{j} \delta_{ip}}, \tag{1}$$

where $\delta_{ip} \geq 0 \; \forall\, p \in \{1, \ldots, j\}$.

In fact, if the ordered sequence of clusters $\langle Clus(c_{i1}), \ldots, Clus(c_{ik_i})\rangle$ composing a forest is generated by the `Decomposition` procedure above, then the corresponding sequence of cluster sizes is monotone and not increasing, and by hypothesis each size is less than $\frac{\sqrt{n}}{4}$. Moreover,

$$|F_i| = \sum_{j=1}^{k_i} n^{\frac{1}{2} - \sum_{p=1}^{j} \delta_{ip}}, \tag{2}$$

where $\langle \delta_{i1}, \ldots, \delta_{ik_i}\rangle$ is a sequence of nonnegative real values.

Let us suppose that the ith generated cluster forest satisfies the conditions

1. $|F_i| < \frac{\sqrt{n}}{4}$,
2. $m_i k_i \geq n$,

and let $Clus(c_{i,k_i})$ be the last cluster chosen. Then we have the following.

LEMMA 5.2. $|Clus(c_{i,k_i})| < \frac{n^{\frac{1}{2} - \delta_{i1}}}{4}$, *where* $n^{\frac{1}{2} - \delta_{i1}} = |Clus(c_{i,1})|$.

Proof. From Lemma 5.1, if $|Clus(c_{ij})| = t_{ij}$, then the number of external vertices related to $Clus(c_{ij})$ is at most t_{ij}^2. As a consequence, we have

$$m_i k_i \leq k_i \sum_{j=1}^{k_i} (t_{ij})^2 = k_i \sum_{j=1}^{k_i} \left(n^{\frac{1}{2} - \sum_{p=1}^{j} \delta_{ip}} \right)^2 \tag{3}$$

$$= k_i \sum_{j=1}^{k_i} n^{1 - 2\sum_{p=1}^{j} \delta_{ip}} \leq k_i \sum_{j=1}^{k_i} n^{1 - 2\delta_{i1}} = k_i^2 n^{1 - 2\delta_{i1}}. \tag{4}$$

Hence, by condition $m_i k_i \geq n$, we get

$$k_i^2 n^{-2\delta_{i1}} \geq 1 \implies k_i \geq n^{\delta_{i1}} > 4. \tag{5}$$

(The last inequality follows from a forest termination condition; i.e.,

$$|F_i| = \sum_{j=1}^{k_i} n^{\frac{1}{2} - \sum_{p=1}^{j} \delta_{ip}} < \frac{\sqrt{n}}{4};$$

hence, $n^{-\delta_{i1}} < \frac{1}{4}$.)

Moreover,

$$\frac{\sqrt{n}}{4} > \sum_{j=1}^{k_i} n^{\left(\frac{1}{2}-\sum_{p=1}^{j}\delta_{ip}\right)} \geq \sum_{j=1}^{k_i} n^{\left(\frac{1}{2}-\sum_{p=1}^{k_i}\delta_{ip}\right)} = k_i\left(n^{\left(\frac{1}{2}-\sum_{p=1}^{k_i}\delta_{ip}\right)}\right) \tag{6}$$

and, from relation (5),

$$k_i\left(n^{\left(\frac{1}{2}-\sum_{p=1}^{k_i}\delta_{ip}\right)}\right) \geq n^{\delta_{i1}} n^{\left(\frac{1}{2}-\sum_{p=1}^{k_i}\delta_{ip}\right)} = n^{\left(\frac{1}{2}-\sum_{p=2}^{k_i}\delta_{ip}\right)}. \tag{7}$$

Hence,

$$n^{\left(\frac{1}{2}-\sum_{p=2}^{k_i}\delta_{ip}\right)} < \frac{\sqrt{n}}{4}.$$

Dividing both terms by $n^{\delta_{i1}}$, we have

$$n^{\left(\frac{1}{2}-\sum_{p=1}^{k_i}\delta_{ip}\right)} < \frac{n^{\left(\frac{1}{2}-\delta_{i1}\right)}}{4}. \tag{8}$$

The left-hand side of inequality (8) is, by definition, the size of $Clus(c_{ik_i})$. □

With reference to the cluster forests sequence $\langle F_{g+1}, \ldots, F_{g+f}\rangle$, let $g+1 \leq i < g+f$. We have the following.

LEMMA 5.3. $|Clus(c_{i+1,1})| < \frac{n^{\frac{1}{2}-\delta_{i1}}}{4}$.

Proof. The proof follows trivially from Lemma 5.2, observing that clusters are taken in not increasing order of size. □

From the above technical lemmas, Lemma 5.4 easily follows.

LEMMA 5.4. *The decomposition strategy returns an* $O(\log n)$ *collection* $\langle F_{g+1}, \ldots, F_{g+f}\rangle$ *of clusters forests.*

From Lemma 5.4, we have the following.

THEOREM 5.5. *The data structure for the representation of dags satisfying the lattice property has an* $O(n\sqrt{n})$*-space complexity and allows us to perform the* REACHABILITY *operation in constant time.*

6. Queries implementation. In this section, we describe algorithms for the operations introduced in Theorem 1.1. A sketch of the algorithm for the REACHABILITY operation already has been given in section 3.4 and the time complexity has been proven in Theorem 5.5.

Let us now describe the PATH operation. For this operation we need to augment our data structure. First note that each double tree is, by construction, the union of two rooted trees. Hence, for each vertex x and for each double tree to which it belongs, it is possible to associate one and only one vertex u (the parent in the tree) which is on the path from x to the vertex inducing the double tree. We then store in the data structures **B** and **D** u's coordinates with respect to this double tree. The PATH operation is now straightforward. While testing the presence of a directed path between x and y using the REACHABILITY(x, y) test, we find the double tree to which they both belong, and then we look at the coordinates of the parents of x and y. These vertices are the immediate successors of x and y on the path from x to y. Then we recursively repeat the search starting from these vertices until we reach the root of the double tree.

In this paper, we will not furnish the full algorithm which, although simple, is quite lengthy. The net result is the following.

PROPOSITION 6.1. *The* PATH(x,y) *operation can be implemented in* $O(l)$ *time, where* l *is the path length.*

For implementing the PRED(x) and SUCC(x) operations, we add new data structures to those described in section 3.4. In particular, for each double tree $DT(u,c)$ of the double tree decomposition $\mathcal{T}$, we maintain the corresponding internal tree $IntTree(u,c)$. Moreover, for each vertex $x \in V$, let $Clus(c)$ be the cluster to which x belongs; if $Clus(c)$ is a $Clus^+(c)$, then we maintain the set of x's successors in $Clus^+(c)$; otherwise, we maintain the set of x's predecessors. Let us denote the first set $Succ(x,c)$ and the second $Pred(x,c)$. Finally, for each vertex $x \in V$ we store the set of double tree identifiers to which each vertex is connected as an external vertex. This information eliminates the need to visit all the **D** tables. By construction, the space complexity of the new data structure is still $O(n\sqrt{n})$.

Let us now introduce the PRED(x) operation. For SUCC(x) operation, the algorithm is similar.

```
    procedure PRED (x)
    begin;
1.     if A[x].sign = "+"
2.     then return (IntTree(x))
3.     else return (Pred(x,c)) ;
4.     for each double tree to which x is connected as external vertex
5.     begin;
6.        Let u be the internal representative vertex of x;
7.        return (IntTree(u));
8.     end; end;
```

PROPOSITION 6.2. *The* PRED(x) *and* SUCC(x) *operations require* $O(k)$ *time, where* k *is the size of the returned set.*

Proof. This follows observing that only actual predecessors (successors) are visited. □

For the RANGE(x,y) operation, we have the following result.

LEMMA 6.1. *Let* $x, y \in V$. *If* REACHABILITY(x,y) = **true**, *then one and only one double tree* $DT(u,c_i)$ *exists such that* RANGE$(x,y) \subseteq Vert(DT(u,c_i))$.

Proof. Let $x \in Clus(c)$ and $y \in Clus(c')$. Let us first suppose that $Clus(c)$ has been generated before $Clus(c')$.

Two cases are possible.

1. $Clus(c)$ is a $Clus^+(c)$.

Let u be the internal representative vertex of y with respect to $Clus(c)$. We claim that $DT(u,c)$ is the required double tree.

Let $z \in$ RANGE(x,y) and $z \in Clus(c'')$. $Clus(c'')$ cannot be generated before $Clus(c)$; otherwise, by cluster definition and according to $Clus(c'')$ sign, either $x \in Clus(c'')$ or $y \in Clus(c'')$.

Hence, $Clus(c'')$ has been generated after $Clus(c)$ and $z \in Ext(Clus(c))$. Since $x \prec z \prec y$, by double tree definition, $x, y, z \in Vert(DT(u,c))$.

2. $Clus(c)$ is a $Clus^-(c)$.

By cluster definition, we have $Clus(c) = Clus(c')$. In this case, we want to prove that $DT(x,c)$ is the required double tree. Since $x \prec y$, then $y \in Vert(DT(x,c))$.

Let $z \in$ RANGE(x,y) and $z \in Clus(c'')$. As before, $Clus(c'')$ cannot be generated before $Clus(c)$. Moreover, in this case, $Clus(c'')$ cannot also be generated after $Clus(c)$; then $Clus(c) = Clus(c'')$ and $z \in Vert(DT(x,c))$.

In a similar way, it is possible to prove that, if $Clus(c')$ has been generated before $Clus(c)$, then either $\text{RANGE}(x, y) \subseteq Vert(DT(u, c'))$, where u is the internal representative vertex of x with respect to $Clus(c')$, or $\text{RANGE}(x, y) \subseteq Vert(DT(y, c'))$, depending on the $Clus(c')$ sign. □

PROPOSITION 6.3. *The* $\text{RANGE}(x, y)$ *operation can be implemented in* $O(k \log d)$ *time, where* k *is the size of the returned set and* d *is the maximum vertex degree (either in-degree or out-degree) of the transitive reduction graph* $G^T = (V, E^T)$.

Proof. Representing all double trees using the implicit data structure of Corollary 3.4, we can first identify in constant time the unique double tree that contains $\text{RANGE}(x, y)$, and we can then report $\text{RANGE}(x, y)$ by visiting this double tree within the required time bound. □

It is important to emphasize that, by means of the $\text{RANGE}(x, y)$ and PATH operations, it is possible to report the transitive reduction subgraph having x as a source and y as a sink.

Let us now describe the $\text{GLB}(x, y)$ and $\text{LUB}(x, y)$ operations. We consider only the $\text{GLB}(x, y)$, as results obtained can be stated, mutatis mutandis, for $\text{LUB}(x, y)$.

Let $x \in Clus(c_h)$ and $y \in Clus(c_k)$ and let $\{Clus(c_1), \ldots, Clus(c_p)\}$ be the subset of clusters of the cluster collection generated by the `BuildCluster` procedure, having as internal vertices at least one predecessor of both x and y. Moreover, let $\{x_1, \ldots, x_p\}$ and $\{y_1, \ldots, y_p\}$ be the corresponding internal representative vertices of x and y, respectively.

Using an argument similar to the one used to prove Lemma 3.1, the following two lemmas can be established.

LEMMA 6.2. *If any* $Clus(c_i)$, *for* $1 \leq i \leq p$, *is a* $Clus^+(c_i)$, *then*

$$if\ IntTree(x_i, c_i) \cap IntTree(y_i, c_i) = \mathcal{I},\ then\ \ LUB(\mathcal{I}) \in \mathcal{I}.$$

LEMMA 6.3. *If any* $Clus(c_i)$, *for* $1 \leq i \leq p$, *is a* $Clus^-(c_i)$, *then*

$$if\ Pred(x_i, c_i) \cap Pred(y_i, c_i) = \mathcal{I},\ then\ \ LUB(\mathcal{I}) \in \mathcal{I}.$$

Let $\{u_1, \ldots, u_p\}$ be the corresponding set of least upper bounds. We have the following.

LEMMA 6.4. *The partial order associated with* $\{u_1, \ldots, u_p\}$ *is a total order.*

Proof. To prove the lemma we show that there exists a permutation $\langle u_{i_1}, \ldots, u_{i_p} \rangle$ such that $u_{i_j} \prec u_{i_m}$, for all $i_1 \leq i_j < i_m \leq i_p$.

Two cases are possible.

1. Either $Clus(c_h)$ is a $Clus^+(c_h)$ or $Clus(c_k)$ is a $Clus^+(c_k)$.

Let the clusters in $\langle Clus(c_1), \ldots, Clus(c_p) \rangle$ be ordered according to the generation order.

If $Clus(c_h)$ is a $Clus^+(c_h)$, then $p \leq h$. In fact, by cluster definition, once $Clus^+(c_h)$ has been taken, then all x's predecessors have been considered. Analogously, if $Clus(c_k)$ is a $Clus^+(c_k)$, then $p \leq k$.

Moreover, all $Clus(c_i)$ in the sequence are $Clus^+(c_i)$. In fact, suppose for contradiction that $Clus(c_j)$, for $1 \leq j \leq p$, is a $Clus^-(c_j)$. If $Clus(c_h)$ is a $Clus^+(c_h)$, then $x \in Clus^-(c_j)$, or, if $Clus(c_k)$ is a $Clus^+(c_k)$, then $y \in Clus^-(c_j)$, but this is a contradiction.

We claim that the sequence of least upper bounds $\langle u_1, \ldots, u_p \rangle$, ordered according to the order in which the corresponding clusters are generated, is the required permutation. Let us assume that two values $1 \leq i < j \leq p$ exist such that either $u_i \sim u_j$ or $u_j \prec u_i$. In the first case the following relations hold simultaneously:

(i) $u_i \prec x$ and $u_i \prec y$;
(ii) $u_j \prec x$ and $u_j \prec y$;
(iii) $u_i \sim u_j$.

However, this contradicts the lattice property.

In the second case, since by hypothesis $Clus(c_i)$ has been generated before cluster $Clus(c_j)$, we have $u_j \in Clus(c_i)$. The proof follows by contradiction.

2. $Clus(c_h)$ is a $Clus^-(c_h)$ and $Clus(c_k)$ is a $Clus^-(c_k)$.

Let us consider the ordered sequence of clusters

$$\langle Clus^+(c_{i_1}), \ldots, Clus^+(c_{i_m}), Clus^-(c_{i_{m+1}}), \ldots, Clus^-(c_{i_p})\rangle$$

defined as follows.

The subsequence $\langle Clus^+(c_{i_1}), \ldots, Clus^+(c_{i_m})\rangle$ is composed of all clusters in $\{Clus(c_1), \ldots, Clus(c_p)\}$ having sign "+" and ordered according to the generation order. The subsequence $\langle Clus^-(c_{i_{m+1}}), \ldots, Clus^-(c_{i_p})\rangle$ is made up of all clusters in $\{Clus(c_1), \ldots, Clus(c_p)\}$ having sign "−", ordered according to the inverse generation order.

We claim that the corresponding sequence $\langle u_{i_1}, \ldots, u_{i_m}, u_{i_{m+1}}, \ldots, u_{i_p}\rangle$ is the desired permutation.

By an argument similar to the one used in the previous case, it is possible to prove that the two subsequences $\langle u_{i_1}, \ldots, u_{i_m}\rangle$ and $\langle u_{i_{m+1}}, \ldots, u_{i_p}\rangle$ are totally ordered. We have to prove that for all $u_{i_j} \in \langle u_{i_1}, \ldots, u_{i_m}\rangle$ and for all $u_{i_l} \in \langle u_{i_{m+1}}, \ldots, u_{i_p}\rangle$, $u_{i_j} \prec u_{i_l}$. By the lattice property either $u_{i_j} \prec u_{i_l}$ or $u_{i_l} \prec u_{i_j}$. In the first case the claim is proved. Suppose for contradiction that $u_{i_l} \prec u_{i_j}$; then, by cluster definition, $u_{i_l} \in Clus^+(c_{i_j})$. This once again contradicts the hypothesis.

This completes the proof. □

Using the above lemma, it is possible to state the following proposition.

PROPOSITION 6.4. *The* GLB(x, y) *and* LUB(x, y) *operations can be implemented in* $O(\sqrt{n})$*-time.*

Proof. To implement these operations we augment the data structure as follows. For each internal tree of a cluster, we maintain the set of least upper bounds (greatest lower bounds) of the sets derived from the intersection with all the other internal trees in the same cluster. Analogously, we maintain for each vertex x the set of least upper bounds (greatest lower bounds) of the sets derived from the intersection between $Pred(x, c)$ ($Succ(x, c)$) and $Pred(y, c)$ ($Succ(y, c)$) for all y in $Clus(c)$.

By construction, the overall space occupancy is still $O(n\sqrt{n})$.

Given x and y, using data structures **D** and **B**, in $O(\sqrt{n})$-time, it is possible to find the set $\langle u_1, \ldots, u_p\rangle$. The maximal element of this sequence is the greatest lower bound of x and y. □

It is worth noting that, by means of LUB(x, y), GLB(x, y) and SUCC(x), PRED(x) operations it is possible to easily implement the following.

1. COMMANC(x, y). Given $x, y \in V$, returns the set of all common ancestors of x and y.

2. COMMSUCC(x, y). Given $x, y \in V$, returns the set of all common successors of x and y.

In particular, from Propositions 6.2 and 6.4, Proposition 6.5 is derived.

PROPOSITION 6.5. *The* COMMANC(x, y) *and* COMMSUCC(x, y) *operations can be implemented in* $O(\sqrt{n} + k)$ *time, where* k *is the size of the returned set.*

To prove Theorem 1.1, it remains to describe LUB$(x_1, \ldots, x_k)$ and GLB$(x_1, \ldots, x_k)$. Note that a straightforward application of Proposition 6.4 leads to an $O(k\sqrt{n})$

worst-case time bound. In order to obtain the desired complexity, we have to underscore some additional properties of the proposed decomposition.

Let us analyze only the GLB$(x_1, \ldots, x_k)$ operation, as LUB$(x_1, \ldots, x_k)$ can be dually derived.

LEMMA 6.5. *If all vertices* $(x_1, \ldots, x_k)$ *belong to the same cluster, then operation* GLB$(x_1, \ldots, x_k)$ *can be performed in* $O(k)$ *steps.*

Proof. The lemma easily follows by exploiting the information added for the GLB(x, y) operation. Starting from the internal trees of the k vertices, we can first compute the $\frac{k}{2}$ least upper bounds associated with pairs of internal trees. Iterating the process on the $\frac{k}{2}$ least upper bounds we find the GLB$(x_1, \ldots, x_k)$, whenever it exists. □

On the other hand, let the vertices $(x_1, \ldots, x_k)$ belong to different clusters. Consider any two vertices, say, x_1 and x_2.

Let $\langle u_1, \ldots, u_p \rangle$ be the sequence of least upper bounds of the set of common predecessors with respect to clusters $\langle Clus(c_1), \ldots, Clus(c_p) \rangle$ (see Lemma 6.4). By construction, any other vertex in the sequence $(x_1, \ldots, x_k)$ satisfies the following property.

LEMMA 6.6. *Let* $x_i \in (x_1, \ldots, x_k)$. *If* u_j *is the greatest vertex in the sequence* $\langle u_1, \ldots, u_p \rangle$ *such that* $u_j \prec x_i$, *then either* GLB$(x_1, x_2, x_i) = u_j$ *or* GLB$(x_1, x_2, x_i) \in IntTree(u_{j+1}, c_{j+1})$.

PROPOSITION 6.6. *The* LUB$(x_1, \ldots, x_k)$ *and* GLB$(x_1, \ldots, x_k)$ *operations can be implemented in* $O(\sqrt{n} + k \log n)$*-time.*

Proof. First observe that, by Proposition 6.4, we can derive the sequence $\langle x_1, \ldots, x_p \rangle$ in $O(\sqrt{n})$-time. Hence, by means of a binary search, in $O(k \log n)$-time, it is possible to derive the maximum element u_j such that $u_j \prec x_i$ for $1 < i < k$. Then, by Lemma 6.6 either GLB$(x_1, \ldots, x_k) = u_j$ or GLB$(x_1, \ldots, x_k) \in IntTree(u_{j+1}, c_{j+1})$. In the former case, the proposition is proved. In the latter case, let $(v_1, \ldots, v_k)$ be the internal representative vertices of $(x_1, \ldots, x_k)$ with respect to cluster $Clus(c_{j+1})$. Now the problem reduces to finding the GLB$(v_1, \ldots, v_k)$, where $(v_1, \ldots, v_k)$ belong to the same cluster. The proof follows by Lemma 6.5. □

By a straightforward application of Lemma 5.5 and Propositions 6.1–6.4 and 6.6, the proof of the main theorem (Theorem 1.1) is completed.

7. Conclusions and open problems. In this paper, a general technique for the representation of a *dag* satisfying the lattice property has been presented. This technique, based on a two-level graph decomposition strategy, is very efficient for the reachability problem resolution, from either a space or a time complexity point of view. In fact, when $m = \Omega(n\sqrt{n})$, it performs a compression of the given *dag*. Note that the complexity bound we derive is optimal, as it matches the theoretical lower bound for this problem (see Proposition 1.1).

The data structure proposed can be efficiently used not only for testing the presence of a path between two given vertices but also for performing a set of basic operations for this class of graphs: namely, find a path between two vertices whenever one exists; given two vertices, find all vertices on the directed paths connecting them in the transitive reduction graph; compute all the successors and/or predecessors of a given vertex; given two vertices, find the least common ancestor and/or the greatest common successor; given a set of vertices, find all common ancestors and/or all common successors.

Furthermore, a stronger result can be derived. In particular, it is possible to represent a partial lattice in space $O(n\sqrt{t})$ with the same time bound for all the

operations, where t is the minimum number of disjoint chains which partition the element set. It is worth noting that this represents an interesting result, since it provides a tight characterization of the complexity of partial lattices. In fact, based on Dilworth's theorem [12], t is equal to the width of the lattice.

Moreover, the graph decomposition strategy, introduced in this paper, has been implemented. Performance evaluations indicate that this strategy can provide a storage occupation substantially less than $O(n\sqrt{n})$-space while maintaining efficiency in reachability query resolution.

Note that the class of graphs under investigation has applications in many fields such as computational geometry, object oriented programming, and distributed systems.

An interesting research direction is to apply our approach for coping with secondary memory management problems. The digraphs representation introduced could represent a powerful clustering technique for minimizing the total number of accessed pages. Our conjecture is supported by results in [16], where two-dimensional lattices are used to produce efficient data structures on paged memory for the half-plane search in two dimensions.

A natural direction for further work is to adopt the same strategy for general directed graphs. In fact, the main problem in this case is that they usually violate Lemma 3.1. In fact, given a general *dag* $G = (V, E)$ and a cluster $Clus(c)$, let u be a vertex in $G - Clus(c)$; if $Clus^+(c) \cap Clus^+(u) \neq \emptyset$, then it can have more than one maximal element. Dually, if $Clus^-(c) \cap Clus^-(u) \neq \emptyset$, then it can have more than one minimal element. In such a case, a constant time reachability test cannot be guaranteed. This prevents us from obtaining a constant time reachability test. In fact, as there is not a one-to-one relation between a given cluster and an external vertex, it is not possible, given a pair of vertices $\langle x, y \rangle$, to univocally identify a double tree containing both x and y.

Nevertheless, the proposed decomposition strategy represents, in this case, a heuristic method for testing reachability which takes into account the sparseness of the graph.

Acknowledgments. The authors wish to thank Prof. Jaroslav Nešetřil for many interesting discussions on the topics presented here and Prof. Rao Kosaraju and the anonymous referee for their detailed comments and suggestions.

REFERENCES

[1] R. Agrawal, *Alpha: An extension of relational algebra to express a class of recursive queries*, IEEE Trans. Software Engrg., 14 (1988), pp. 879–885.

[2] R. Agrawal, A. Borgida, and H. V. Jagadish, *Efficient management of transitive relationship in large data and knowledge bases*, in Proc. ACM Internat. Conf. on Management of Data, ACM, New York, 1989, pp. 253–262.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[4] M. Ajtai and R. Fagin, *Reachability is harder for directed than for undirected finite graphs*, J. Symbolic Logic, 55 (1990), pp. 113–150.

[5] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovasz, and C. Rackoff, *Random walks, universal traversal sequences, and the complexity of maze problems*, in Proceedings 20th Annual IEEE Symposium on Foundations of Computer Science, San Juan, Puerto Rico, IEEE Computer Society Press, Los Alamitos, CA, 1979, pp. 218–223.

[6] G. Barnes and W. L. Ruzzo, *Undirected s–t connectivity in polynomial and sublinear space*, Comput. Complex., 6 (1996), pp. 1–28.

[7] G. Birkhoff, *Lattice Theory*, American Mathematical Society Colloquium Publications 25, AMS, Providence, RI, 1979.

[8] G. Birkhoff and O. Frink, *Representation of lattices by sets*, Trans. AMS, 64 (1948), pp. 299–316.
[9] J. Biskup and H. Stiefeling, *Evaluation of upper bounds and least nodes as database operations*, in Lecture Notes in Comput. Sci. 730, Springer-Verlag, New York, pp. 197–214.
[10] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa, *Two applications of inductive counting for complementation problems*, SIAM J. Comput., 18 (1989), pp. 559–578.
[11] B. A. Davey and H. A. Priestly, *Introduction to Lattices and Order*, Cambridge University Press, Cambridge, 1990.
[12] R. Dilworth, *A Decomposition Theorem for Partially Ordered Sets*, Ann. Math., 51 (1950), pp. 161–165.
[13] B. Dushnik and E. Miller, *Partially ordered sets*, Amer. J. Math., 63 (1941), pp. 600–610.
[14] D. Eppstein, Z. Galil, G. Italiano, and A. Nissenzweig, *Sparsification—a technique for speeding up dynamic graph algorithms*, J. ACM, 44 (1997), pp. 669–696.
[15] S. Even, *Graph Algorithm*, Computer Science Press, Rockville, MD, 1979.
[16] P. G. Franciosa and M. Talamo, *Orders, k-sets and fast halfplane search on paged memory*, in Orders, Algorithms, and Applications, International Workshop ORDAL '94, Lecture Notes in Comput. Sci. 831, V. Bouchitté and M. Morvan, eds., Springer-Verlag, Berlin, 1994, pp. 117–127.
[17] G. Gambosi, M. Protasi, and M. Talamo, *An efficient implicit data structure for relation testing and searching in partially ordered sets*, BIT, 33 (1993), pp. 29–45.
[18] M. Habib and L. Nourine, *Bit-vector encoding for partially ordered sets*, in Orders, Algorithms, and Applications, International Workshop ORDAL '94, V. Bouchitté and M. Morvan, eds., Lecture Notes in Comput. Sci. 831, Springer-Verlag, New York, 1994, pp. 1–12.
[19] H. V. Jagadish, *Incorporating hierarchy in a relational model of data*, SIGMOD Record (ACM Special Interest Group on Management of Data), 18 (1989), pp. 78–87.
[20] T. Kameda, *On the vector representation of the reachability in planar directed graphs*, Inform. Process. Lett., 3 (1974/75), pp. 75–77.
[21] D. Kelly, *On the dimension of partially ordered sets*, Discrete Math., 35 (1981), pp. 135–156.
[22] D. J. Kleitman and K. J. Winston, *The asymptotic number of lattices*, Ann. Discrete Math., 6 (1980), pp. 243–249.
[23] G. Markowsky, *The Representation of posets and lattices by sets*, Algebra Universalis, 11 (1980), pp. 173–192.
[24] R. H. Möhring, *Computationally tractable classes of ordered sets*, Tech. Report 87468-OR, Bonn University, Germany, 1987.
[25] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, Berlin, New York, 1985.
[26] F. P. Preparata and R. Tamassia, *Fully dynamic point location in a monotone subdivision*, SIAM J. Comput., 18 (1989), pp. 811–830.
[27] I. Rival, *Graphical data structures for ordered sets*, in Algorithms and Order, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1989, pp. 3–31.
[28] M. Talamo and P. Vocca, *Fast lattice browsing on sparse representation*, in Orders, Algorithms, and Applications, International Workshop ORDAL '94, Lecture Notes in Comput. Sci. 831, V. Bouchitté and M. Morvan, eds., Springer-Verlag, Berlin, 1994, pp. 186–204.
[29] M. Talamo and P. Vocca, *A data structure for lattices representation*, Theoret. Comput. Sci., 175 (1997), pp. 373–392.
[30] R. Tamassia and J. G. Tollis, *Reachability in planar digraphs with one source and one sink*, Theoret. Comput. Sci., 119 (1993), pp. 331–343.
[31] M. Tompa, *Two familiar transitive closure algorithms which admit no polynomial time, sublinear space implementations*, SIAM J. Comput., 11 (1982), pp. 130–137.
[32] J. W. T. Trotter and J. J. I. Moore, *The dimension of planar posets*, J. Combin. Theory, 22 (1977), pp. 54–57.
[33] M. Yannakakis, *Graph-theoretic methods in database theory*, PODS '90, Proceedings Ninth ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, ACM, New York, 1990, pp. 230–242.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1806–1828

BANDWIDTH ALLOCATION WITH PREEMPTION*

AMOTZ BAR-NOY†, RAN CANETTI‡, SHAY KUTTEN§, YISHAY MANSOUR¶, AND BARUCH SCHIEBER‡

Abstract. Bandwidth allocation is a fundamental problem in the design of networks where bandwidth has to be reserved for connections in advance. The problem is intensified when the overall requested bandwidth exceeds the capacity and not all requests can be served. Furthermore, acceptance/rejection decisions regarding connections have to be made online, without knowledge of future requests. We show that the ability to *preempt* (i.e., abort) connections while in service in order to schedule "more valuable" connections substantially improves the throughput of some networks. We present bandwidth allocation strategies that use preemption and show that they achieve *constant competitiveness* with respect to the throughput, given that any single call requests at most a constant fraction of the bandwidth. Our results should be contrasted with recent works showing that nonpreemptive strategies have at most inverse logarithmic competitiveness.

Key words. bandwidth allocation, online algorithms, preemption, call control, call admission

AMS subject classifications. 68M20, 68Q20, 68Q25, 90B12, 90B35

PII. S0097539797321237

1. Introduction.

Bandwidth allocation is one of the most important problems in the management of networks that have guaranteed bandwidth policy (e.g., asynchronous transfer mode (ATM) [5], PARIS [9], IBM BBNS [10]). In such networks the application has to reserve in advance sufficient bandwidth for its communication. The guaranteed bandwidth policy is contrasted with the more traditional policy (e.g., TCP/IP), where information packets are routed as they come to the network without prior knowledge about the connections. The advantages of the guaranteed bandwidth policy are many and include bounded latency for real-time tasks; fairness (e.g., one user cannot overtake the entire network's resources); and simple pricing (the application can be charged for the bandwidth it allocated). The major drawback of the guaranteed bandwidth policy is inefficiency: the communication links may be underutilized. Thus, a "good" bandwidth allocation strategy is crucial for such networks.

The bandwidth allocation problem becomes more difficult in view of the large variety of applications that use the network simultaneously. These applications have very different requirements in terms of bandwidth, duration, delay, information loss,

*Received by the editors May 9, 1997; accepted for publication (in revised form) May 29, 1998; published electronically May 13, 1999.

http://www.siam.org/journals/sicomp/28-5/32123.html

†Electrical Engineering Department, Tel Aviv University, Tel Aviv 69978, Israel (amotz@eng.tau.ac.il). The research of this author was supported in part by a grant from the Israeli Ministry of Science and Technology. Part of the research of this author was done while at the IBM Research Division, T.J. Watson Research Center.

‡IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598 (canetti@watson.ibm.com, sbar@watson.ibm.com). Part of the research of R. Canetti was done at the Laboratory of Computer Science, MIT, and was supported by American-Israeli Binational Science Foundation grant 92-00226.

§Department of Industrial Engineering, The Technion, Haifa 32000, Israel (kutten@ie.technion.ac.il). Part of the research of this author was done at the IBM Research Division, T.J. Watson Research Center.

¶Computer Science Department, Tel Aviv University, Tel Aviv 69978, Israel (mansour@cs.tau.ac.il). The research of this author was supported in part by the Israel Science Foundation, administered by the Israel Academy of Science and Humanities, and by a grant from the Israeli Ministry of Science and Technology.

etc. Furthermore, since the communication volume may be much larger than buffer space, decisions regarding current requests cannot be delayed and have to be made without knowledge of future requests. The problem is further intensified when the available bandwidth cannot accommodate all requests for bandwidth and some have to be rejected.

In this context it is natural to consider the possibility of "softening" the rigidity of the guaranteed bandwidth policy by allowing preemption (i.e., abortion) of connections in service, in order to schedule "more valuable" connections that would otherwise be rejected. Preempting a connection has obvious disadvantages: all the work that was done so far may be lost, and the transmission, in some applications, has to start again later. However, the ability to preempt certain types of connections as a policy may greatly improve the performance of the network. Understanding the power of preemption in this context may shed new light on the value of the guaranteed bandwidth policy.

Indeed, some types of connections should never be preempted (e.g., phone calls). However, there exist other scenarios where preemption is acceptable and even essential. For instance, a high-priority (say, real-time) connection should be allowed to preempt a low-priority connection. (See [18] for an implementation of a channel where this type of preemption is used.) Another case is preempting other connections of the same user [19] or of different users based on a pricing model [21]. The willingness of users to communicate over preemptable, low-priority connections would probably depend on the price discount they would receive, compared to a nonpreemptable connection. Our work shows that preemptable connections allow far better utilization of the network, and thus users may be charged at a considerably lower rate.

In this work, we concentrate on preemption as a tool for enhancing the throughput, or utilization, of the network. We develop various preemption strategies (specifying *when* and *which connections* to preempt) for maximizing the throughput *of connections that eventually complete.* Our strategies perform provably and significantly better (in terms of throughput) than any nonpreemptive strategy for bandwidth allocation.

We study two models for bandwidth allocation. The first is a single link, where requests for connections (or *calls*) arrive one by one as time proceeds; each call has duration and bandwidth requirements (specified in advance). Requests have to be either served immediately or rejected (for example, due to limited buffer space). This model is an abstraction of a single *virtual path* in an ATM network that has a single entry and exit. A virtual path in an ATM network serves as a "highway" that is used by many *virtual circuits* (i.e., connections) simultaneously. The bandwidth of the virtual path has to be divided among the various virtual circuits. The second model is a line of processors where each connection has source, destination, and required bandwidth. Here we assume that all requests are for permanent connections (or, alternatively, that all calls arrive at the same time, in some arbitrary order, and have the same duration). This model can be viewed as an abstraction of a virtual path with multiple entries and exits.

As suggested by the online nature of the problem (decisions on current requests are made without knowledge of future requests), we use competitive analysis to measure the performance of bandwidth allocation algorithms. We define the performance of a bandwidth allocation algorithm on a sequence of requests as the throughput of completed calls (i.e., calls that are neither rejected nor preempted). This throughput can be measured as the sum, over completed calls, of the duration times the bandwidth

requirement, or equivalently as the integral over time of the bandwidth used by calls that eventually complete. The competitiveness of an algorithm is the *infimum* over all request sequences of the ratio of the performance of the algorithm to the performance of the best (offline) schedule for this sequence.

The competitiveness of our algorithms depends only on δ, the ratio between the largest bandwidth requested by any *single* call and the capacity of the line. We note that whereas the capacity of the network can be arbitrarily large, the ratio δ is typically a constant smaller than one. In both models we achieve constant competitiveness if δ is a constant smaller than one. We contrast our results with the fact that nonpreemptive bandwidth allocation strategies have inverse logarithmic competitiveness. (We elaborate on this point later.)

Our algorithms are simple and efficient, however surprising and nonintuitive at first. They suggest the following approach to bandwidth allocation: in deciding which calls to reject or preempt, the algorithms consider only the duration of a call and the time for which a call in service already has been running, *completely ignoring the throughput of calls.* In particular, a call with very large throughput may be preempted in order to make room for a call with longer duration and much smaller throughput. Still, our algorithms achieve constant competitiveness (if δ is bounded away from one) where more straightforward strategies fail.

We show that our algorithms have optimal competitiveness, up to a small constant, for all values of δ. Furthermore, we show that (i) deterministic algorithms have a very poor competitive ratio if a single call may request the entire bandwidth (that is, if $\delta = 1$); (ii) in the line model, if we let calls have arbitrary duration and δ be more than half, then constant competitiveness cannot be achieved by any deterministic algorithm. (Bounds on the competitiveness of *randomized* preemptive bandwidth allocation algorithms in this and related models are shown in [8].)

We also consider a special case of the single link model where all calls have identical bandwidth, which is $1/k$ of the capacity of the link for some integer k. This model can be visualized as k parallel links, each of unit capacity. Even for this restricted model, called the *parallel links* model, we show that any deterministic online algorithm has a competitive ratio of at most 0.66 (the bound holds *for all* k). The parallel links model is closely related to online preemptive task scheduling under overload [7, 6, 14, 15, 22, 20]. Our impossibility result applies to this problem as well.

Our work extends previous work of Garay and Gopal [12] and Garay et al. [13]. These papers also consider online bandwidth allocation with preemption on networks with line topology. However, they simplify the model by assuming that the bandwidth requirement of each call is equal to the bandwidth of the links (and thus, in particular, only one call at a time can be served on a link).

Considerable work has been recently done on nonpreemptive online bandwidth allocation (also referred to as call control); we mention here only some of this work. In [2] (and also in [1]) the problem of nonpreemptive online bandwidth allocation and virtual circuit routing on an arbitrary network was considered. Under the assumption that no call may request more than a logarithmic fraction of the bandwidth, Awerbuch, Azar, and Plotkin [2] and Aspnes et al. [1] presented a strategy with competitiveness inversely proportional to the logarithm of the size of the network times the ratio between the largest and smallest value of a call, and they proved that no online strategy has better performance. *Randomized* algorithms for nonpreemptive call control on tree-like networks are given in [3], where competitiveness inversely logarithmic in similar parameters is shown with matching impossibility results. Other

topologies are considered in [4] with similar results. Lipton and Tomkins [16] considered nonpreemptive online "interval scheduling" in a model similar to our line model. They achieved a slightly worse than inverse logarithmic competitive ratio and showed that no online algorithm can achieve an inverse logarithmic competitive ratio. In summary, whenever preemption is not allowed the "logarithmic barrier" seems to be unbreakable, even by randomized algorithms.

Finally, we remark that Faigle and Nawijn [11] also showed that preemption is a useful tool. They considered the special case in which all calls have identical bandwidth. However, their goal was to minimize the number of rejected calls, while our goal is to maximize the throughput. They described an optimal deterministic online algorithm (competitiveness 1) when preemption is allowed, whereas it was known before that without preemption no algorithm with constant competitiveness exists.

In section 2 we introduce the single link model and define bandwidth allocation algorithms in this model. In section 3 we describe our algorithms for the single link model. In section 4 we introduce the line model and show how our algorithms can be adapted to this model. In section 5 we demonstrate the optimality of our algorithms.

2. The single link model. Consider a communication link with bandwidth capacity $\mathcal{B}$ (where $\mathcal{B}$ may be very large). A call is a connection established between the two endpoints of the link. Each call c is characterized by the required bandwidth b_c, the request issue time t_c, and the duration d_c (known in advance). Let $e_c \stackrel{\text{def}}{=} t_c + d_c$ be the ending time of a call c. For simplicity we assume that the time is discrete. Our convention is that the minimum bandwidth requested by a call is 1. Let δ denote the maximum fraction of the capacity used by a single call. We have $\frac{1}{\mathcal{B}} \leq \delta \leq 1$.

The requests arrive one by one in an online fashion. A request for a call c can be either served immediately or rejected. The algorithm may *preempt* (i.e., stop or abort) calls during service. The operation of the algorithm run by the control center can thus be described as follows. Upon a request for a call c, if serving c does not violate the bandwidth capacity of the link, then serve c. Otherwise, either reject the request or preempt some calls that are currently being served so that call c can be served within the capacity of the link.

Let the throughput of a call c, denoted by v_c, be its bandwidth times its duration, i.e., $v_c = b_c \cdot d_c$. Once a call is completed, a value equal to the throughput of the call is gained. No gain is accrued for preempted calls. Note that the throughput of a completed call is a measure for the amount of information contained in it.

We use competitive analysis to measure the performance of our bandwidth allocation algorithms. The competitive ratio of an algorithm is the infimum over all possible request sequences of the total throughput of the algorithm on a sequence divided by the total throughput of the best (offline) algorithm on this sequence. More formally, for a sequence $S = c_1, \ldots, c_n$ of call requests, let the bandwidth requested by S at time t be

$$\tilde{B}_S(t) \stackrel{\text{def}}{=} \sum_{\{c \in S \mid t \in [t_c \ldots t_c + d_c]\}} b_c.$$

We say that S is *feasible* if $\tilde{B}_S(t) \leq \mathcal{B}$ for all times t. Let the *cover* of a sequence S be $V(S) \stackrel{\text{def}}{=} \sum_t B_S(t)$, where $B_S(t) \stackrel{\text{def}}{=} \min\{\mathcal{B}, \tilde{B}_S(t)\}$. If S is feasible, then we say that $V(S)$ is the throughput of S (that is, the cover equals the throughput). Note that if S is feasible, then $V(S) = \sum_{c \in S} v_c$.

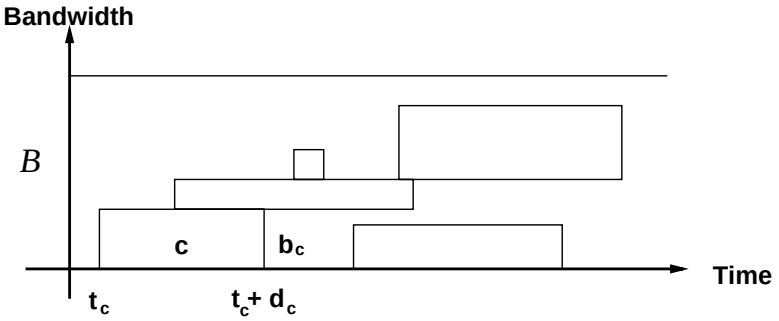


FIG. 3.1. *A geometric representation of calls on a single link.*

For an algorithm $\mathcal{A}$ and sequence S, let $A(S) \subseteq S$ be the sequence of calls completed by $\mathcal{A}$ on input S. (We use $S' \subseteq S$ to denote that S' is a subsequence of S.) Algorithm $\mathcal{A}$ is a valid *bandwidth allocation* algorithm if for any sequence of requests S, $A(S)$ is feasible. Algorithm $\mathcal{A}$ is ρ-*competitive* if

$$\rho \leq \inf_{S} \min_{\{S' \subseteq S | S' \text{ feasible}\}} \frac{V(A(S))}{V(S')}.$$

Note that $0 \leq \rho \leq 1$, and the closer ρ is to 1 the better the algorithm performs.

Remark. Say that algorithm $\mathcal{A}$ is *strictly* ρ-*competitive* if

$$\rho \leq \inf_{S} \frac{V(A(S))}{V(S)}.$$

Since the link cannot serve more than $\mathcal{B}$ bandwidth at any given time, the cover is an upper bound on the throughput of any feasible subsequence of S. Thus, any strictly ρ-competitive algorithm is also ρ-competitive. We show that our algorithms are strictly competitive. While for the positive results we consider this stronger notion of competitiveness, all our impossibility results (upper bounds) are for the regular notion of competitiveness.

3. Bandwidth allocation on a single link. We present two algorithms for bandwidth allocation on a single link. The first algorithm, called the left-right (LR) algorithm, is $(\frac{1}{2} - \delta)$-competitive for $\delta < \frac{1}{2}$. The second algorithm, called the effective time (EFT) algorithm, is $\frac{1-\delta}{4}$-competitive for all $\delta < 1$. For $\delta > \frac{1}{3}$ algorithm EFT has better competitiveness, whereas for $\delta < \frac{1}{3}$ algorithm LR has better competitiveness. In particular, for very small values of δ, the LR algorithm is about $\frac{1}{2}$-competitive. (In section 5 it is shown that for small δ no algorithm can have a better competitive ratio than 0.66.) We stress that, although $\mathcal{B}$ can be arbitrarily large, our algorithms have constant competitiveness whenever the fraction δ is a constant smaller than one. Furthermore, our algorithms do not depend on δ. Therefore, their performance on any request sequence depends on the δ fraction *of this particular sequence.*

The following geometric representation of the scenario may be helpful (see Figure 3.1). Let the x axis represent time and the y axis represent bandwidth. Each call c is a rectangle of length d_c and height b_c. We have to fit the rectangles above the x axis and below the line $y = \mathcal{B}$, under the constraint that the rectangle of call c has to start at $x = t_c$. Note, however, that a call need not use the same "bandwidth pieces" for its total duration. Thus we are allowed to "break" the rectangles vertically, as long

as enough bandwidth is allocated at all times. (This distinguishes our problem from many other problems, e.g., memory allocation.)

In subsection 3.1 we briefly and informally explain why some straightforward strategies for bandwidth allocation fail. We present our algorithms in the two subsections that follow.

3.1. First tries. We first show that basing a bandwidth allocation strategy on the throughput of the calls is a bad idea, although we want to maximize the total throughput of completed calls. This applies both to a greedy strategy (e.g., always prefer calls with larger throughput) and to a "double-the-gain" strategy (e.g., serve an incoming call if enough bandwidth can be freed by only preempting calls whose combined throughput is at most half of the throughput of the incoming call). Consider the following request sequence. First, $\sqrt{\mathcal{B}}$ calls are requested at time 0, each of bandwidth $\sqrt{\mathcal{B}}$ and duration $\sqrt{\mathcal{B}}$. Next, still at time 0, $\mathcal{B}$ calls, each of bandwidth 1 and duration $\frac{\mathcal{B}}{2}$, are requested. Serving any of the last $\mathcal{B}$ calls requires preempting one of the first $\sqrt{\mathcal{B}}$ calls. However, each of the last calls has smaller throughput than each of the first calls. Thus, both the greedy and the double-the-gain algorithms serve only the first $\sqrt{\mathcal{B}}$ calls, gaining $\mathcal{B}^{3/2}$. The best schedule is the last $\mathcal{B}$ calls with throughput $\frac{\mathcal{B}^2}{2}$; thus the competitiveness is at most $\frac{2}{\sqrt{\mathcal{B}}}$. The moral is that calls with longer duration are preferable *even if the longer calls have smaller throughput,* as there may be many similar calls coming in the future.

An alternative strategy may thus be to consider the duration of calls (say, use a "double-the-duration" algorithm). It turns out that considering only the duration is also not good enough (we omit further counterexamples). An additional parameter should be considered, namely the amount of time a call in service has been running (or, alternatively, the amount of time it will run in the future). Each of our algorithms considers a different combination of these parameters.

3.2. The LR algorithm. The LR algorithm implements a compromise between the need to hold on to jobs that have been running for the longest time (thus capitalizing on work done) and the need to hold on to jobs that will run for the longest time in the future (thus guaranteeing future work). The compromise is simple: half of the capacity is dedicated to each of these two classes of jobs. Surprisingly, this simple compromise yields a good competitive ratio. The LR algorithm is described in Figure 3.2.

THEOREM 3.1. *For $\delta < \frac{1}{2}$, algorithm* LR *is $(\frac{1}{2} - \delta)$-competitive.*

Proof. First, note that the total bandwidth required by the calls in $L \cup R$ at any time t is at most $B_L(t) + B_R(t) \le \frac{\mathcal{B}}{2} + \frac{\mathcal{B}}{2} = \mathcal{B}$. Thus the algorithm is valid. Next we show its competitiveness.

Consider an input sequence $S = c_1, \ldots, c_n$ of call requests and assume that $\delta \stackrel{\text{def}}{=} \max_i\{\frac{b_{c_i}}{\mathcal{B}}\}$ is at most $\frac{1}{2}$ (otherwise the competitive claim in the theorem is vacuous). Let $E \stackrel{\text{def}}{=} \text{LR}(S)$ be the set of calls completed by LR on input S. Let S_i be the prefix of S composed of the first i calls in S, and let E_i be the set of calls completed by the algorithm, assuming that the input sequence is only S_i. Note that E_i is the union of two sets: (i) the set of calls completed up to time t_{c_i} (that is, up to the time call c_i is requested), and (ii) the set of calls being served at time t_{c_i}. Below we show, by induction on i, that for all i and for all times t, $B_{E_i}(t) \ge \min\{B_{S_i}(t), (\frac{1}{2} - \delta)\mathcal{B}\}$. Since $S = S_n$ and $E = E_n$, we have $B_E(t) \ge (\frac{1}{2} - \delta)B_S(t)$ for all t. (The worst case is when $B_S(t) = \mathcal{B}$.) Thus the total throughput of the LR algorithm, $V(E)$, is at least $(\frac{1}{2} - \delta)V(S)$, and the theorem follows.

Let F be the set of calls currently in service. Upon the request of a call c do:

1. Add c to F.
2. Find the following two sets of calls, L and R:
   (a) Sort the calls by *increasing* order of *starting* time, and let L be the maximal set of calls at the top of the list (i.e., earliest starting times) such that the total bandwidth required by the calls in L is at most $\frac{\mathcal{B}}{2}$.
   (b) Sort the calls by *decreasing* order of *ending* time, and let R be the maximal set of calls at the top of the list (i.e., latest ending times) such that the total bandwidth required by the calls in L is at most $\frac{\mathcal{B}}{2}$.
3. Preempt/reject calls that are neither in L nor in R to fit in the link capacity.

FIG. 3.2. *Algorithm LR.*

The inductive claim trivially holds for $i = 0$. Let $i > 0$, and fix some $t' \geq 0$. We distinguish two cases.

Case 1. No call p that requested bandwidth for time t' was rejected or preempted in the ith step (that is, when processing the ith request). In this case, if the ith request, c_i, requests bandwidth for time t', then $B_{E_i}(t') - B_{E_{i-1}}(t') = b_c \geq B_{S_i}(t') - B_{S_{i-1}}(t')$. If c_i does not request bandwidth for time t', then $B_{S_i}(t') - B_{S_{i-1}}(t') = 0 = B_{E_i}(t') - B_{E_{i-1}}(t')$. Thus the inductive claim holds.

Case 2. There exist calls that requested bandwidth for time t' and were rejected or preempted in the ith step. We show that in this case $B_{E_i}(t') \geq (\frac{1}{2} - \delta)\mathcal{B}$. Let p be a call that was rejected or preempted in the ith step, for which $t' \in [t_p \dots e_p]$. (Otherwise, $B_{E_i}(t')$ is not affected by call p.) Since the bandwidth requested by p is at most $\delta\mathcal{B}$, both $B_L(t_{c_i})$ and $B_R(t_{c_i})$ must be at least $(\frac{1}{2} - \delta)\mathcal{B}$, otherwise p would be in either L or R. (See also Figure 3.3.) Furthermore, $p \notin L$; thus we must have $B_L(t) \geq (\frac{1}{2} - \delta)\mathcal{B}$ for all times $t \in [t_p \dots t_{c_i}]$. (In other words all the calls in L started before t_p and are still running at t_{c_i}.) Similarly, since $p \notin R$, we have $B_R(t) \geq (\frac{1}{2} - \delta)\mathcal{B}$ for all times $t \in [t_{c_i} \dots e_p]$. (In other words all the calls in R end after e_p and started running before t_{c_i}.) Thus, $B_{E_i}(t') = B_{L \cup R}(t') \geq (\frac{1}{2} - \delta)\mathcal{B}$. □

3.3. The EFT algorithm. The EFT algorithm implements a different compromise between banking on past profit and ensuring future profit. Rather than dividing the bandwidth between the two classes, this algorithm attaches a time-value, called the *effective time,* to each single call. Calls with later effective times are preempted first. The effective time τ_c of a call c is its arrival time minus its duration, i.e., $\tau_c \stackrel{\text{def}}{=} t_c - d_c$. This strategy reflects (in a way described below) the idea that work that is done is worth twice as much as work to be done. Algorithm EFT is described in Figure 3.4.

The effective time strategy may be viewed as a variant of a "doubling strategy" as follows. Let r be a requested call and c be a call in service. By preempting c and serving r, we "gain" the time interval $[e_c \dots e_r]$ and "lose" the time interval $[t_c \dots t_r]$. We will preempt c to make room for r if the time-gain is more than twice the loss, that is, if $e_r - e_c > 2(t_r - t_c)$. This condition is equivalent to $t_r - d_r < t_c - d_c$, or $\tau_r < \tau_c$.

THEOREM 3.2. *For $\delta < 1$, algorithm* EFT *is $(\frac{1-\delta}{4})$-competitive.*

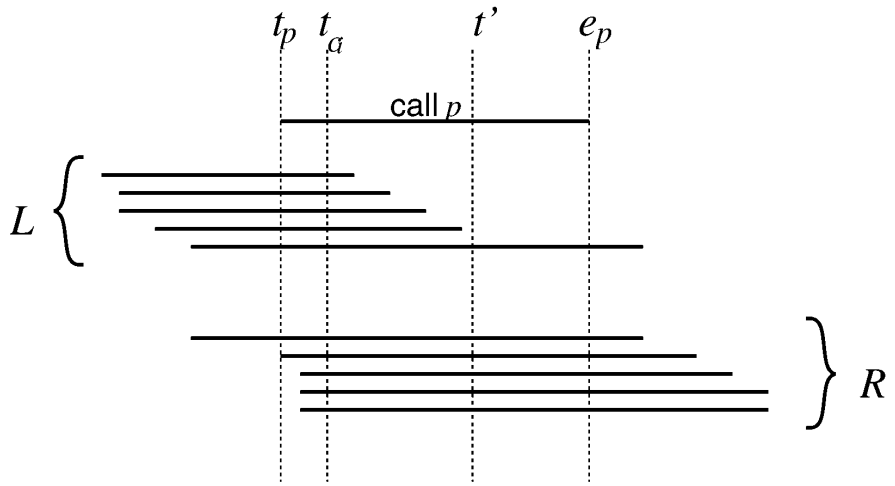


FIG. 3.3. *Call p was either rejected or preempted in the ith step. The total bandwidth of each of the sets L and R is at least $(\frac{1}{2}-\delta)\mathcal{B}$. Since t' intersects either all calls in L or all calls in R, we have $B_{E_i}(t') \geq (\frac{1}{2}-\delta)\mathcal{B}$.*

Let the effective time of a call c be $\tau_c \stackrel{\text{def}}{=} t_c - d_c$.
Maintain a list L of the calls in service, sorted by increasing order of effective time.
(Calls that have the same effective time are ordered arbitrarily.)
Upon the request of a call c, do:

1. Add c to L in place.
2. Reject/preempt calls from the end of the list (i.e., "latest" effective times), to fit in the link capacity.

FIG. 3.4. *Algorithm EFT.*

Proof. The validity of algorithm EFT is immediate from the description. We show its competitiveness. Unlike algorithm LR, here there may be times when the optimal schedule has the link used to capacity while algorithm EFT has almost no bandwidth allocated to calls that eventually complete. Therefore, we use a different "bookkeeping" method for proving the competitiveness of EFT.

Consider some input sequence $S = c_1, \ldots, c_n$ of call requests, and let $\delta \stackrel{\text{def}}{=} \max\{\frac{b_{c_i}}{\mathcal{B}}\}$. Let $E \stackrel{\text{def}}{=} \text{EFT}(S)$ be the set of calls completed by algorithm EFT on input S. We introduce a sequence, E', of "virtual calls" and show that $V(E) \geq \frac{1-\delta}{4} \cdot V(E')$ and $V(E') \geq V(S)$. Therefore, the total throughput of the algorithm (i.e., $V(E)$) is at least $\frac{1-\delta}{4} \cdot V(S)$.

We define virtual calls as follows. For each call c, let the virtual call c' have arrival time $t_{c'} = \tau_c = t_c - d_c$, ending time $e_{c'} = e_c + 2d_c$, and bandwidth $b_{c'} = \frac{1}{1-\delta} b_c$. For a set A of calls, let $A' \stackrel{\text{def}}{=} \{c' : c \in A\}$. Note that the duration of each virtual call is $e_{c'} - t_{c'} = (e_c + 2d_c) - (t_c - d_c) = 4d_c$ and therefore the throughput of each virtual call c' is $\frac{4}{1-\delta}$ times the throughput of the corresponding real call c. Thus, $V(E) \geq \frac{1-\delta}{4} \cdot V(E')$. In Lemma 3.4 below we show that $B_{E'}(t) \geq B_S(t)$ for all t. Thus, $V(E') \geq V(S)$, and the theorem follows. □

Let us first prove a technical claim. (See Figure 3.5.)

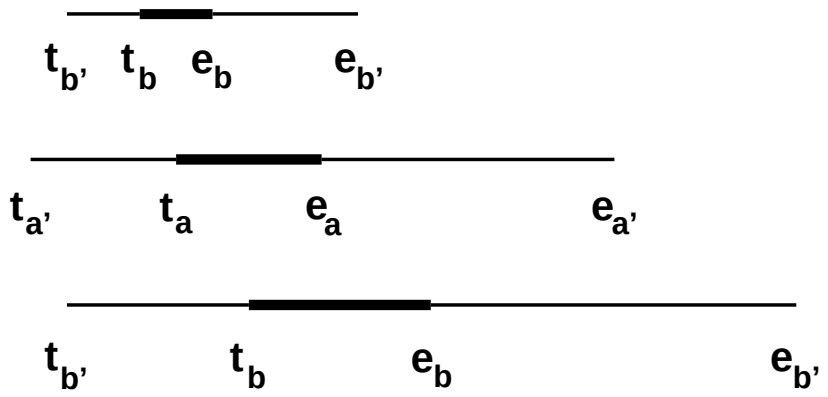


FIG. 3.5. *The proof of the technical claim.*

CLAIM 3.3. *Let a and b be two calls scheduled by the algorithm at time t.*
(1) *If $t_{a'} \le t_{b'}$ and $t_a \ge t_b$, then $e_{a'} \ge e_{b'}$.*
(2) *If $t_{a'} \le t_{b'}$, then $e_{a'} \ge e_b$.*
Proof. Part (1): If $t_a \ge t_b$ and $t_{a'} \le t_{b'}$, then it must be that $d_a \ge d_b$. Thus,

$$e_{a'} = t_a + 3d_a \ge t_b + 3d_b = e_{b'} .$$

Part (2): Consider the case not covered in part (1), i.e., $t_a < t_b$ and $t_{a'} \le t_{b'}$. Call a has not ended by time t_b, thus $d_a \ge t_b - t_a$. Also, it follows from $t_{a'} \le t_{b'}$ that $d_a \ge d_b - (t_b - t_a)$. Thus,

$$\begin{aligned} e_{a'} &= t_a + d_a + 2d_a \\ &\ge t_a + d_b - (t_b - t_a) + 2(t_b - t_a) \\ &= t_b + d_b = e_b. \end{aligned}$$

□

LEMMA 3.4. *For any request sequence S and for all times t we have $B_{E'}(t) \ge B_S(t)$.*

Proof. Say that a time t is *i-quiet* if, up to and including the ith request, all the calls that requested bandwidth for time t (that is, calls c such that $t \in [t_c \dots e_c]$) were completed by the algorithm. Note that if time t is i-quiet, it is also j-quiet for all $1 \le j \le i$. A time t is quiet if it is i-quiet for all i. If a time t is quiet, then certainly $B_{E'}(t) > B_E(t) = B_S(t)$. It remains to deal with times that are not quiet (i.e., times that are not i-quiet for some request i). Define S_i and E_i as in the proof of Theorem 3.1. (That is, S_i is the prefix of S composed of the first i calls in S, and E_i is the set of calls completed by the algorithm assuming that the input sequence is only S_i.) We show, by induction on i, that $B_{E'_i}(t) = \mathcal{B}$ for all times t that are not i-quiet. Since $E = E_n$ and $S = S_n$ we get $B_{E'}(t) \ge B_S(t)$ for all times t.

The inductive claim holds vacuously for $i = 0$. For $i > 0$ we distinguish three cases.

Case 1. The ith request, c_i, was served without preempting other calls. In this case, any time that is not i-quiet is also not $(i-1)$-quiet, and $B_{E'_i}(t) \ge B_{E'_{i-1}}(t)$ for all times t. Thus the claim follows from the induction hypothesis.

Case 2. The ith request was rejected without preempting other calls. In this case, the inductive hypothesis holds for any time t that is not $(i-1)$-quiet. Also, all times that are $(i-1)$-quiet and are not i-quiet are in the time range $[t_{c_i} \dots e_{c_i}]$. Thus it is

enough to show that $B_{E'_i}(t) = \mathcal{B}$ for all $t \in [t_{c_i} \dots e_{c_i}]$. Let F be the set of calls being served at time t_{c_i}. Since c_i is rejected, and $b_{c_i} \leq \delta\mathcal{B}$, the total bandwidth of the calls in F is at least $(1-\delta)\mathcal{B}$. We have $t_{f'} \leq t_{c'_i}$ for all $f \in F$; thus by Claim 3.3, part (2), we have $e_{c_i} \leq e_{f'}$. It follows that $B_{E'_i}(t) \geq B_{F'}(t) \geq \mathcal{B}$ for all times $t \in [t_{c_i} \dots e_{c_i}]$. (In fact this holds for all times $t \in [t_{c'_i} \dots e_{c_i}]$.)

Case 3. Calls were preempted while processing request i. Let P_i be the set of calls that were preempted (or rejected) while processing request i.[1] For a set A of calls let t_A (resp., e_A) denote the earliest starting time (resp., latest ending time) of a call in A. Here we have to consider the time range $[t_{P'_i} \dots e_{P'_i}]$ (rather than only $[t_{P_i} \dots e_{P_i}]$) since calls in P_i that contributed to $B_{E'_{i-1}}(t)$ are now preempted.

It can be seen, similar to the proof of Case 2, that $B_{E'_i}(t) \geq \mathcal{B}$ for all times $t \in [t_{P'_i} \dots e_{P_i}]$. It is left to show that $B_{E'_i}(t) \geq \mathcal{B}$ for all times in $[e_{P_i} \dots e_{P'_i}]$ that are not i-quiet.

Let t be a time in $[e_{P_i} \dots e_{P'_i}]$ that is not i-quiet. Thus there exists a request $j \leq i$ that caused either rejection or preemption of a call that requested bandwidth at time t. We complete the proof by showing, for each request k, $j \leq k \leq i$, a set $G_k \subseteq E_k$ of calls such that $B_{G'_k}(t) \geq \mathcal{B}$. These sets are defined inductively as follows. Define F_k to be the set of calls in service immediately after request k was processed. Let $G_j \stackrel{\text{def}}{=} F_j$. For $k > j$, if c_k is rejected or if c_k is placed in the ordered list L of calls in service after all the calls in G_{k-1}, then $G_k = G_{k-1} - P_k$. Otherwise, $G_k = G_{k-1} \cup \{c_k\} - P_k$. Certainly $G_k \subseteq E_k$. We show by induction on k that the following two properties hold for all $j \leq k \leq i$. (We are interested in Property 2.)

Property 1. For all calls $g \in G_k$, $t \in [t_{g'} \dots e_{g'}]$.

Property 2. $B_{G'_k}(t) = \mathcal{B}$.

Consider the case $k = j$. When request j was processed, a call asking bandwidth for time t was rejected or preempted, so therefore by Claim 3.3 all calls in $F_j = G_j$ have Property 1. Furthermore, the total bandwidth of all calls in G_j must be at least $(1-\delta)\mathcal{B}$; otherwise one of the calls that was either rejected or preempted while processing request j would have been kept by the algorithm. Property 2 thus follows as well.

Now fix some $k > j$ and assume that the two properties hold for G_{k-1}. We distinguish two subcases.

Case 3.1. Call c_k either is rejected or appears in the ordered list L of calls in service after all the calls in G_{k-1}. In this case, Property 1 holds since $G_k \subseteq G_{k-1}$. If call c_k is rejected, then Property 2 is shown as in the base case $k = j$. If call c_k is served, then since it appears after all the calls in G_{k-1} in the list L, all these calls are served as well, and $G_k = G_{k-1}$. In this case Property 2 follows from the induction hypothesis.

Case 3.2. Call c_k is served and appears in the sorted list L before some call $g \in G_{k-1}$. Here, $G_k = G_{k-1} \cup \{c_k\} - P_k$. We first show Property 1. From the induction hypothesis, Property 1 holds with respect to all calls in G_k other than c_k. Note that $t_g \leq t_{c_k}$. By our assumption also $\tau_{c_k} \leq \tau_g$, and thus it follows from Claim 3.3, part (1) that $t_{c'_k} \leq t_{g'}$ and $e_{g'} \leq e_{c'_k}$. Since $t \in [t_{g'} \dots e_{g'}]$ we have $t \in [t_{c'_k} \dots e_{c'_k}]$, which proves Property 1 for call c_k.

Next we show that G_k satisfies Property 2. If $P_k \cap G_{k-1} = \emptyset$, then $G_{k-1} \subseteq G_k$ and Property 2 holds. Otherwise, a call in G_{k-1} was preempted while processing

[1] It is possible that calls are preempted and also that the incoming call is rejected. For simplicity we include the rejected call with the preempted calls.

request k. From the definition of G_{k-1} it follows that in this case all calls not in G_{k-1} that were in service upon the arrival of request k must have been preempted as well. Consequently, $F_k = G_k$. However, the total bandwidth of calls in F_k must be at least $(1-\delta)\mathcal{B}$, otherwise one of the calls that was either rejected or preempted while request k was processed would have been kept by the algorithm. Thus, the total bandwidth of calls in G_k must be at least $(1-\delta)\mathcal{B}$. By Property 1, all calls in G_k request bandwidth for time t. Property 2 follows. □

4. Bandwidth allocation on a line within a single time slot. In this section we consider bandwidth allocation in the *line model*, in which a sequence of stations are connected on a line by communication links. For simplicity we assume that all links have unit length and the same bandwidth capacity, $\mathcal{B}$. Our results generalize to networks with individual capacities and lengths of the links. A call is a connection between two stations on the line. We consider the case where all calls arrive at the same time unit (in some arbitrary order) and have the same duration. In the next section we show that if we let calls have arbitrary duration, and if δ is more than half, then constant competitiveness cannot be achieved by any deterministic algorithm. (We note that this holds even if randomization is allowed [8].)

More formally, the line model is defined as follows. Assume the stations are labeled by consecutive integers increasing from left to right. A call c is characterized by its left endpoint l_c, its right endpoint r_c, and its bandwidth requirement b_c. Let the length of call c be $h_c \stackrel{\text{def}}{=} r_c - l_c$. The throughput of c is now $v_c = h_c \cdot b_c$. Again, bearing in mind that our goal is to maximize the throughput of the network, we define the additive throughput accrued from serving a completed call c to be its throughput v_c. Here the throughput can also be regarded as the amount of network resources allocated for the call. The definitions of the cover, feasibility, and throughput of a request sequence, as well as the competitiveness and validity of bandwidth allocation algorithms, are similar to those of the single link model (see section 2), where time t is replaced, in the natural way, by location (link) e on the line and earlier times are translated to locations with smaller labels. This model is a generalization of the single link model, in the sense that any algorithm for the line model is valid, and has the same performance, in the single link model (when "location" is translated to "time"). The converse is not true, due to reasons described later.

We adapt our algorithms to this more general scenario. The adapted LR algorithm, called ALR, is still $(\frac{1}{2}-\delta)$-competitive for $\delta < \frac{1}{2}$. The adapted EFT algorithm, called AEFT, is $\frac{1-\delta}{2\phi+1}$-competitive for all $\delta < 1$, where $\phi = 1 + \frac{1}{\phi} = \frac{1+\sqrt{5}}{2} \approx 1.6$ is the golden ratio. The first difficulty in adapting our algorithms to this model is as follows. In the single link model there is a "current time" at which all calls in service require bandwidth and where the new call must start; thus all bandwidth conflicts are at the current time. In the line model, bandwidth conflicts upon the arrival of a new call are not limited to a single location; at some locations, it may seem beneficial to accept the incoming call, where at other locations it may seem beneficial to reject. It turns out that the following technique provides a sufficient solution for this difficulty in both adapted algorithms. Upon the arrival of a request, we first add the requested call to the list of calls in service. Next, we go over the links one by one, in an arbitrary order, and resolve remaining conflicts on each link by rejecting or preempting calls according to a scheme similar to the original one (where "time" is replaced by "location"). The ALR algorithm requires no further modifications. The AEFT algorithm encounters an additional difficulty, described below.

Let LR* be the same algorithm as LR, where time is translated to location.
Let F be the set of calls currently in service. Upon the request of a call c do:

1. Add c to F.
2. While there are links e with a conflict (i.e., $\tilde{B}_F(e) > \mathcal{B}$) do:
   Let e be a link with a conflict, and let $F_e \subseteq F$ be the set of calls currently in service that are using link e. Run algorithm LR* on the set of calls F_e.
   Remark: Clearly, after running LR* on F_e link e has no conflict. However, since the rejected/preempted calls may span other links, this may also resolve conflicts along other links as well.

FIG. 4.1. *Algorithm ALR.*

4.1. The ALR algorithm. The ALR algorithm is described in Figure 4.1.

THEOREM 4.1. *For $\delta < \frac{1}{2}$, algorithm ALR is $(\frac{1}{2} - \delta)$-competitive.*

Proof. The proof is similar to the proof of Theorem 3.1. Consider a link e with a conflict. Let $L_e \subseteq F_e$ and $R_e \subseteq F_e$ be the two sets of calls computed by LR* when run on F_e. Note that the total bandwidth required by the calls in $L_e \cup R_e$ is at most $\frac{\mathcal{B}}{2} + \frac{\mathcal{B}}{2} = \mathcal{B}$. Thus the algorithm is valid. Next we show competitiveness.

Consider an input sequence $S = c_1, \ldots, c_n$ of call requests and assume that $\delta = \max_i\{\frac{b_{c_i}}{\mathcal{B}}\}$ is at most $\frac{1}{2}$. Let $E \stackrel{\text{def}}{=} \text{LR}^*(S)$ be the set of calls completed by LR*. Let S_i be the prefix of S composed of the first i calls in S, and let E_i be the set of calls completed by the algorithm, assuming that the input sequence is only S_i. Below we show, by induction on i, that for all i and for all links e, $B_{E_i}(e) \geq \min\{B_{S_i}(e), (\frac{1}{2} - \delta)\mathcal{B}\}$. Since $S = S_n$ and $E = E_n$, we have $B_E(e) \geq (\frac{1}{2} - \delta)B_S(e)$ for all e. Thus the total throughput of the ALR algorithm, $V(E)$, is at least $(\frac{1}{2} - \delta)V(S)$. The theorem follows.

The inductive claim trivially holds for $i = 0$. For $i > 0$ we distinguish two cases. Fix some link e.

Case 1 (identical to the proof of Theorem 3.1). No call p that requested bandwidth for link e (i.e., $e \in [l_p \ldots r_p]$) was rejected or preempted in the ith step (that is, when processing the ith request). In this case, if the ith request, c_i, requests bandwidth for link e, then $B_{E_i}(e) - B_{E_{i-1}}(e) = b_c \geq B_{S_i}(e) - B_{S_{i-1}}(e)$. If c_i does not request bandwidth for link e, then $B_{S_i}(e) - B_{S_{i-1}}(e) = 0 = B_{E_i}(e) - B_{E_{i-1}}(e)$. Thus the inductive claim holds.

Case 2. There exist calls that requested bandwidth for link e and were rejected or preempted in the ith step. We show that in this case $B_{E_i}(e) \geq (\frac{1}{2} - \delta)\mathcal{B}$. Let $p_1, \ldots, p_a$ be the calls that were rejected or preempted in the ith step, sequenced by the order of their rejection or preemption. We prove, by induction on $1 \leq j \leq a$, that when p_j is either rejected or preempted, the total bandwidth of the calls in service on e is at least $(\frac{1}{2} - \delta)\mathcal{B}$. Suppose that this was the case before the rejection or preemption of p_j. If $e \notin [l_{p_j} \ldots r_{p_j}]$, then this is clearly the case also after the rejection or preemption of p_j. Otherwise, let e' be the link such that p_j was rejected or preempted while LR* was run on $F_{e'}$, and let $L_{e'}$ (resp., $R_{e'}$) be the set L (resp., R) found by LR* when operating on e'. Assume e is to the left of e' (i.e., $e \in [l_{p_j} \ldots e']$). All the calls in $L_{e'}$ include e since they start to the left of (or at) l_{p_j}. Since $p_j \notin L_{e'}$, and the bandwidth requested by p_j is at most $\delta\mathcal{B}$, $B_L(e')$ and thus also $B_L(e)$ must be no less than $(\frac{1}{2} - \delta)\mathcal{B}$. Similarly, if e is to the right of e' (i.e., $e \in [e' \ldots r_{p_j}]$), then $B_R(e')$ and thus also $B_R(e)$ must be no less than $(\frac{1}{2} - \delta)\mathcal{B}$.

Let F be the list of calls currently in service. Upon the request of a call c do:

1. Add c to F.
2. While there are links e with a conflict (i.e., $\tilde{B}_F(e) > \mathcal{B}$) do:
   Let e be a link with a conflict, and let $F_e \subseteq F$ be the list of calls using e, sorted so that if $a \succ b$ then a is ahead of b in the list.
   Reject/preempt calls from the end of F_e to fit in the link capacity.

FIG. 4.2. *Algorithm AEFT.*

4.2. The AEFT algorithm. The following difficulty is encountered in adapting the EFT method to the line model. The original EFT algorithm weighed past work and future work differently, relying on the fact that time is directional (i.e., the starting time of a new call is no earlier than the starting time of the calls in service). In the line model, past and future lose their meaning: the left endpoint of a new call is not necessarily "more to the right" than the left endpoints of the calls in service. Instead, the adapted algorithm will use "effective endpoints" both to the left and to the right, as follows. For a call c with left endpoint l_c, right endpoint r_c, and length $h_c = l_c - r_c$, let the *effective span* be the range $s_c = [l_c - g \cdot h_c \ldots r_c + g \cdot h_c]$, where g is some constant to be computed later. We define a complete order, denoted $\succ$, on the calls. For calls a and b, $a \succ b$ either if the effective span of a contains the effective span of b (i.e., $s_b \subset s_a$), or if the effective spans of a and b are not contained in each other and a is requested before b. Algorithm AEFT is described in Figure 4.2.

THEOREM 4.2. *For $\delta < 1$, Algorithm AEFT is $(\frac{1-\delta}{2m+1})$-competitive, where $m = \max\{g, 1 + \frac{1}{g}\}$, and g is the constant used for determining effective spans.*

COROLLARY 4.3. *Let $g = \phi$ be the golden ratio (that is, $\phi = 1 + \frac{1}{\phi}$ and $\phi = \frac{1+\sqrt{5}}{2} \approx 1.6$). For $\delta < 1$, algorithm AEFT is $\frac{1-\delta}{2\phi+1}$-competitive.*

Proof of Theorem 4.2. We follow the outline of the proof of Theorem 3.2. (We also use the notation of Theorem 3.2.) Let $m = \max\{g, 1 + \frac{1}{g}\}$. Define virtual calls as follows. The virtual call c' that corresponds to a call c has left endpoint $l_{c'} = l_c - m \cdot h_c$ and right endpoint $r_{c'} = r_c + m \cdot h_c$ and requires bandwidth $b_{c'} = \frac{1}{1-\delta} b_c$. Consider a sequence S of incoming requests and let $E \stackrel{\text{def}}{=} AEFT(S)$. Clearly, $V(E) \geq \frac{1-\delta}{2m+1} V(E')$. We show below that $B_{E'}(e) \geq B_S(e)$ for all links e. Consequently $V(E') \geq V(S)$, and the algorithm is $\frac{1-\delta}{2m+1}$-competitive.

We show, by induction on the number of calls in the input sequence, that $B_{E'}(e) \geq \mathcal{B}$ for each link e where a call requested bandwidth and was rejected or preempted. Similar to the proofs of Theorem 3.2 and Lemma 3.4, the inductive claim is shown by proving the following: (1) Whenever a call c is *rejected* (when considering a link e), its span is contained in the spans of all the virtual calls that correspond to the calls that remain in service and use link e. (The span of a call c is the range $[l_c \ldots r_c]$.) (2) Whenever a call c is *preempted* (when considering a link e), the links that are not quiet in the span of the virtual call c' are contained in the spans of virtual calls of bandwidth at least $\mathcal{B}$ that remain in service.

Case 1. Suppose that an incoming call c is rejected while considering some link e in step 2 of Algorithm AEFT (given in Figure 4.2). Here it suffices to show that $l_{f'} \leq l_c$ and $r_c \leq r_{f'}$ for all remaining calls $f \in F_e$. This clearly holds for all calls in F_e whose effective span contains the effective span of c. Consider a call f that remained in F_e whose effective span does not contain the effective span of c. Without

loss of generality assume that f is to the right of c (that is, $l_c < l_f < r_c < r_f$). Since c was rejected, we have $f \succ c$, thus the effective span of c does not contain the effective span of call f. Observe that $l_c - gh_c < l_f - gh_f$ (otherwise $r_f + gh_f \geq r_c + gh_c$, implying that the effective span of f contains the effective span of c). This implies that $r_c + gh_c \leq r_f + gh_f$, or

$$gh_c \leq r_f - r_c + gh_f. \tag{4.1}$$

By our assumption that $l_f < r_c$, we have $r_f - r_c < h_f$. Substituting this inequality in (4.1) we get $gh_c < (1+g)h_f$, or equivalently

$$h_c < \left(1 + \frac{1}{g}\right) h_f \leq mh_f.$$

Therefore, $l_{f'} = l_f - mh_f \leq l_f - h_c \leq l_c$, and $r_{f'} = r_f + mh_f \geq r_f + h_c \geq r_c$.

Case 2. Suppose that a call p is preempted when a call c is requested (and call c is accepted). We show that in this case $l_{c'} \leq l_{p'}$ and $r_{p'} \leq r_{c'}$. The proof that the spans of the virtual calls that correspond to the rest of the calls that remain in service contain the links that are not quiet in the span of call c' is the same as in Case 1. We have $c \succ p$, since call c is accepted. Since call c was requested after call p, the only way for the relation $c \succ p$ to hold is if the effective span of c contains the effective span of p; that is, $l_c - gh_c \leq l_p - gh_p$ and $r_c + gh_c \geq r_p + gh_p$ (and at least one of these inequalities is strict). Since $m \geq g$, then $l_{c'} \leq l_{p'}$ and $r_{p'} \leq r_{c'}$. ∎

5. Optimality of the algorithms. We prove impossibility results for the competitive ratio of any *deterministic* online bandwidth allocation algorithm *on a single link*, demonstrating that our algorithms achieve close to optimal competitiveness for all values of δ. First we show that if a single call requires the entire bandwidth (i.e., $\delta = 1$), then no algorithm can be more than $\frac{1}{\mathcal{B}}$-competitive. (Note that $\mathcal{B}$ may be unboundedly large.) Next we show that if $\delta \geq \frac{1}{2}$, then no algorithm can be more than $\min\{\frac{1}{8}, (1-\delta)\}$-competitive. In particular, if $\delta = \frac{1}{2}$, then no algorithm can be more than $\frac{1}{8}$-competitive. Recall that algorithm EFT is $\frac{1-\delta}{4}$-competitive; therefore for $\delta = \frac{1}{2}$ EFT is optimal, for $\frac{1}{2} < \delta \leq \frac{7}{8}$ it is at most a factor of $\frac{1}{2(1-\delta)}$ from optimality, and for $\frac{7}{8} \leq \delta < 1$ it is at most a factor of 4 from optimality. For $\frac{1}{3} < \delta < \frac{1}{2}$ the competitiveness of any algorithm is shown to be at most $\frac{1}{2}$, therefore our algorithm is at most a factor of $\frac{2}{1-\delta}$ from optimality. Finally, for $0 < \delta \leq \frac{1}{3}$ the competitiveness of any algorithm is shown to be at most $\frac{2}{3}$, therefore our algorithm is at most a factor of $\frac{4}{3(1-2\delta)}$ from optimality. We summarize our results for the different ranges in Table 5.1 and the graph in Figure 5.1.

We show the impossibility results by demonstrating, for any algorithm, an input sequence of call requests which forces any algorithm to perform poorly compared to the best (offline) schedule. We describe input sequences as if they may change "on the fly," depending on the choices of the algorithm so far. Such a description is valid since the behavior of a deterministic algorithm on any prefix of the input sequence can be thought of as known beforehand. We sometimes call the sequence creator the *adversary*.

This section is organized as follows. First, we describe the known impossibility results for the case in which calls always request the whole bandwidth. Next we prove our impossibility result for the case $\delta = 1$. Using the techniques of these two bounds we show the $\frac{1}{8}$ bound for the case $\frac{1}{2} \leq \delta < 1$. Next we prove the $1 - \delta$ bound for the case $\frac{1}{2} < \delta < 1$. We proceed for the case $0 \leq \delta < \frac{1}{2}$ and prove our bounds using the

TABLE 5.1
All bounds in the different ranges of $0 < \delta \leq 1$.

Range	Lower bound (algorithm)	Upper bound (impossibility result)	Optimality ratio $\left(\frac{\text{upper bound}}{\text{lower bound}}\right)$
$\delta = 1$	$\frac{1}{\mathcal{B}}$	$\frac{1}{\mathcal{B}}$	1
$\frac{7}{8} \leq \delta < 1$	$\frac{1-\delta}{4}$	$1 - \delta$	4
$\frac{1}{2} \leq \delta \leq \frac{7}{8}$	$\frac{1-\delta}{4}$	$\frac{1}{8}$	$\frac{1}{2(1-\delta)} \leq 4$
$\frac{1}{3} \leq \delta \leq \frac{1}{2}$	$\frac{1-\delta}{4}$	$\frac{1}{2}$	$\frac{2}{1-\delta} \leq 4$
$0 < \delta \leq \frac{1}{3}$	$\frac{1}{2} - \delta$	$\frac{2}{3}$	$\frac{4}{3(1-2\delta)} \leq 4$
$\delta \to 0$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{4}{3}$

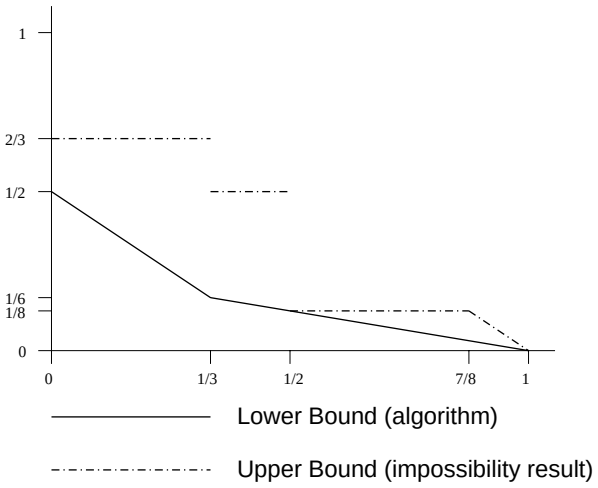


FIG. 5.1. *Graph of bounds in the different ranges of* $0 < \delta \leq 1$.

more restricted parallel links model (to be defined there). At the end of the section, we consider the more general case of calls with arbitrary durations on a general linear network. We show that any deterministic online algorithm for this case is at most $\frac{1}{\mathcal{B}}$-competitive, if $\delta > \frac{1}{2}$.

The whole bandwidth case. The sequence S is an adaptation of a sequence demonstrated in [13] to show that no algorithm can be more than $\frac{1}{4}$-competitive for the case where all calls require bandwidth *exactly* $\mathcal{B}$. (The sequence in [13] is an adaptation of a bound for scheduling algorithms in the presence of overload, shown in [6].) Let us review their construction. For any $\alpha > \frac{1}{4}$, construct a sequence $C = c_1, \ldots, c_n$ of calls where each call c_i arrives "just before call c_{i-1} ends" (i.e., $t_{c_i} = e_{c_{i-1}} - 1$, where $e_{c_i} \stackrel{\text{def}}{=} t_{c_i} + d_{c_i}$ is the ending time of c_i), and $d_{c_{i-1}} < \alpha e_{c_i}$ for all i, $d_{c_n} \leq d_{c_{n-1}}$. The offline algorithm is able to schedule calls that cover the entire duration $0 \leq t \leq e_i$, in a way described below. Since $d_{c_{i-1}} < \alpha e_{c_i}$ for all i, the online algorithm is always forced to preempt the single call in service (that is, c_{i-1}) when a new call arrives; otherwise it is not α-competitive since it covers only $d_{c_{i-1}}$ time units while the offline algorithm covers e_{c_i} time units. Thus the algorithm completes

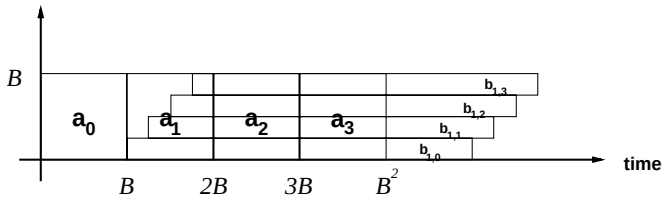


FIG. 5.2. *The impossibility result for the case* $\delta = 1$.

either only c_n or only c_{n-1} with value at most $d_{c_{n-1}}$, whereas the offline can cover the entire duration $0 \le t \le e_{c_n}$. Consequently, once the call c_n is requested the online algorithm can no longer be α-competitive. To complete the construction, set $d_{c_1} = \mathcal{B}$ and interleave the c_i calls with many additional service calls, each of duration 1. The service calls cannot be served by the online algorithm since once a service call is served the sequence stops and the algorithm is no longer competitive. The offline algorithm serves all service calls and accrues throughput e_{c_n}.

The case $\delta = 1$. We show that if $\delta = 1$, then for any online algorithm $\mathcal{A}$ there exists an input sequence S such that the throughput $V(A(S)) = \mathcal{B}^2$, whereas some feasible subsequence $S' \subseteq S$ has a gain $V(S') = \mathcal{B}^3$. Consequently, the competitiveness of $\mathcal{A}$ is at most $\frac{1}{\mathcal{B}}$. (See Figure 5.2.)

The sequence S consists of two types of calls: squares and slices. A square call has duration $\mathcal{B}$ time units and bandwidth $\mathcal{B}$. A slice call has duration $\mathcal{B}^2$ time units and bandwidth 1. Both have throughput $\mathcal{B}^2$. Observe that no algorithm can serve both a square and a slice call that intersect (that is, require bandwidth for the same time). The sequence S starts with a square call at time 0. The algorithm $\mathcal{A}$ must serve this call, otherwise it is 0-competitive. Next, the following procedure is repeated until S contains either $\mathcal{B}$ square calls or $\mathcal{B}$ slice calls:

(a) As long as $\mathcal{A}$ serves a square call, then at each time unit a slice call is requested.

(b) As long as $\mathcal{A}$ serves a slice call, then every $\mathcal{B}$ time units a square call is requested.

Analysis. At the end of the process A has completed at most one call, hence $V(A(S)) \le \mathcal{B}^2$. Conversely, a feasible $S' \subseteq S$ may contain $\mathcal{B}$ calls of the same type (either squares or slices) and be of value $V(S') = \mathcal{B} \cdot \mathcal{B}^2 = \mathcal{B}^3$.

The case $\delta = \frac{1}{2}$. Consider the sequence $C = c_1, \ldots, c_n$ as described above for a given $\alpha > \frac{1}{4}$. From now on we omit references to α and think of the sequence that is associated with $\alpha = \frac{1}{4} + \epsilon$, where ϵ is negligible. Rigorous proof follows in a straightforward way. Although we present the bound only for $\delta = \frac{1}{2}$, it will be clear from the presentation that the bound holds for any $\delta > \frac{1}{2}$.

The basic idea behind our proof is to force the algorithm to serve c_i with bandwidth $\frac{B}{2}$ while the offline can serve calls that cover the whole bandwidth until e_{i+1}. This will add an extra factor of 2 and therefore the $\frac{1}{8}$ bound.

In the sequence C, let the length of call c_i be γ_i and let its beginning time and ending time be τ_i and ϵ_i, respectively. Define a new call c_{n+1} with parameters $\tau_{n+1} = \tau_n$, $\gamma_{n+1} = \gamma_n$, and $\epsilon_{n+1} = \epsilon_n$, and choose $\mathcal{B} \gg n \cdot \epsilon_n$. We construct a sequence S that contains three types of calls. First, for each i there are two calls f_i with bandwidth $\frac{1}{2}\mathcal{B}$, arrival time $t_i = \mathcal{B}^3\tau_i$, ending time $e_i = \mathcal{B}^3\epsilon_i$, and duration $d_i = \mathcal{B}^3\gamma_i$. Next, there are many service calls of type a_i with duration $\mathcal{B}^3(2\epsilon_{i+1})$ and bandwidth 1 and many service calls of type b with duration $\mathcal{B}$ and bandwidth $\frac{1}{2}\mathcal{B}$.

The starting time of the service call would depend on the behavior of the algorithm, as explained later.

Recall that v_c is the throughput of call c. We get that $v_b = \frac{\mathcal{B}^2}{2}$, $v_{a_i} = \mathcal{B}^3(2\epsilon_{i+1}) = o(\mathcal{B}^4)$, and $v_{f_i} = \frac{1}{2}\mathcal{B}d_i = \frac{\mathcal{B}^4}{2}\gamma_i = O(\mathcal{B}^4)$. Note also that since $\epsilon_n \cdot n \ll \mathcal{B}$ it follows that $\sum_{j=1}^{n} v_{a_j} = o(\mathcal{B}^4)$. These equations, and the fact that

$$\frac{d_i}{e_{i+1}} = \frac{\gamma_i}{\epsilon_{i+1}} \to \frac{1}{4},$$

imply the following proposition.

PROPOSITION 5.1.

$$\frac{v_b + \sum_{j=1}^{i} v_{a_j}}{v_{f_i}} \ll \frac{1}{8}, \tag{5.1}$$

$$\frac{v_b + v_{f_i} + \sum_{j=1}^{i} v_{a_j}}{\mathcal{B}v_{a_i}} \le \frac{2v_{f_i} + \sum_{j=1}^{i} v_{a_j}}{\mathcal{B}v_{a_i}} = \frac{\gamma_i}{2\epsilon_{i+1}} + \frac{\sum_{j=1}^{i} v_{a_j}}{\mathcal{B}^4\epsilon_{i+1}} \to \frac{1}{8}, \tag{5.2}$$

$$\frac{v_{f_i} + \sum_{j=1}^{i+1} v_{a_j}}{\mathcal{B}e_{i+1}} = \frac{\gamma_i}{2\epsilon_{i+1}} + \frac{\sum_{j=1}^{i+1} v_{a_j}}{\mathcal{B}^4\epsilon_{i+1}} \to \frac{1}{8}. \tag{5.3}$$

Say that we are in the ith phase if the following three conditions hold.

Condition 1. The algorithm has not completed any calls of type b or f_j for $1 \le j < i$ before time t_i.

Condition 2. The algorithm serves one call of type f_i and possibly one call of type a_j for $1 \le j \le i$.

Condition 3. There exists a feasible subsequence $S' \subseteq S$, consisting only of calls of type b, that covers the entire bandwidth for times in $[0 \dots t_i]$.

The sequence S starts by requesting an f_1 call and an a_1 call, both at time 0. The algorithm must serve the f_1 call in order to be competitive. Hence, we are now in the first phase. We show below how to force the algorithm to reach the $(i+1)$st phase from the ith phase. The bound will follow once the nth phase is reached. At that stage, the gain of the online algorithm is at most $v_{f_n} + \sum_{j=1}^{n+1} v_{a_j} = v_{f_{n-1}} + \sum_{j=1}^{n+1} v_{a_j}$ while the gain of the offline is $\mathcal{B}e_n$. Equation (5.3) of Proposition 5.1 proves the $\frac{1}{8}$ bound.

Once in the ith phase, the $(i+1)$st phase is reached as follows.

1. At each time t, $t_i \le t \le e_i - \mathcal{B}$, request two calls of type b. In order to serve any of these calls, the algorithm has to preempt either the f_i call or the a_i call currently in service.
   (a) Suppose that the call f_i is preempted. In this case we stop. The algorithm serves a call of type b and calls of type a_j for $1 \le j \le i$, whereas the offline can serve an f_i call. The bound is yielded by (5.1).
   (b) Suppose that the call a_i is preempted. Request calls of type a_i at times $t+1, t+2, \dots$ until the algorithm preempts the f_i call, preempts the b type call, or rejects $\mathcal{B}$ calls of type a_i. The first case is the same as the previous case (a). In the second case we are back in the ith phase and we continue offering the calls of type b. In the third case we stop. The algorithm serves a call of type b, a call of type f_i, and some calls of type a_j for $1 \le j < i$, whereas the offline can serve $\mathcal{B}$ calls of type a_i. The bound is yielded by (5.2).

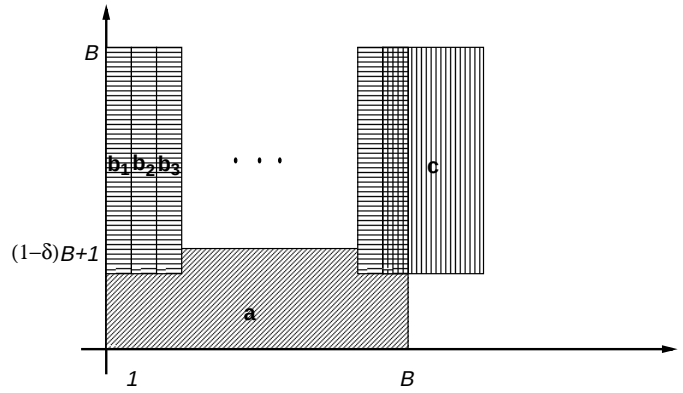


FIG. 5.3. *The impossibility result for the case* $\frac{1}{2} < \delta < 1$. *Here,* $\mathcal{B} = 12$ *and* $\delta = \frac{3}{4}$.

We thus assume that none of these b calls are served. Note that now there exists a feasible subsequence $S' \subseteq S$, consisting only of b calls, which covers the entire bandwidth for times in $[0 \dots e_i]$.

2. At time $e_i - \mathcal{B}$, when the f_i call is about to end, request two calls of type f_{i+1}. Now, the algorithm has several options.
   (a) The algorithm rejects both calls. In this case we stop. The offline can serve b type calls to cover the whole bandwidth up to t_{i+1} and the two f_{i+1} calls to cover the whole bandwidth up to e_{i+1}, whereas the algorithm can serve one f_i call and some a_j type calls for $1 \le j \le i$. The bound follows from (5.3).
   (b) The algorithm serves one of the f_{i+1} calls and preempts the f_i call. In this case we reached the $(i+1)$st phase.
   (c) The algorithm serves one of the f_{i+1} calls and preempts the a_i call. This means that at this stage, the algorithm serves a call of type f_i and a call of type f_{i+1}. In this case, request calls of type a_{i+1} at times $e_i - \mathcal{B}, e_i - \mathcal{B} + 1, \dots$ until the algorithm does one of the following: preempts the f_i call, preempts the f_{i+1} call, or rejects $\mathcal{B}$ calls of type a_{i+1}. In the first case we reach the $(i+1)$st phase. The second case is the same as the previous case (a) in which the algorithm rejects the two f_{i+1} calls. In the third case we stop. Since $v_{f_i} < v_{f_{i+1}}$ it follows that the throughput of the algorithm is at most $2v_{f_{i+1}} + \sum_{j=1}^{i} v_{a_j}$, whereas the throughput of the offline can be $\mathcal{B}v_{a_{i+1}}$. The bound is yielded by (5.2).
   (d) The algorithm serves both f_{i+1} calls. Again we request calls of type a_{i+1} at times $e_i - \mathcal{B}, e_i - \mathcal{B} + 1, \dots$ until the algorithm either preempts one of the f_{i+1} calls or rejects $\mathcal{B}$ calls of type a_{i+1}. Both cases appeared in the previous case (c).

The case $\frac{1}{2} < \delta < 1$. We show that in this case any online algorithm is at most $(1 - \delta + o(1))$-competitive. Note that for $\frac{1}{2} \le \delta \le \frac{7}{8}$ the previous bound (of $\frac{1}{8}$) is better. Consider the following request sequence to an online algorithm (see Figure 5.3):

1. Request a call a at time 0, with duration $\mathcal{B}$ time units and bandwidth $(1 - \delta)\mathcal{B} + 1$. If the algorithm does not serve this call, then end the sequence.
2. At each time $0 \le i < \mathcal{B}$ request a call b_i of duration 1 and bandwidth $\delta\mathcal{B}$. No b_i call can be served together with a. If the algorithm preempts a to serve one of the b_i calls, then end the sequence.

3. Request a call c at time $\mathcal{B}-1$ with duration $1+\frac{1-\delta}{\delta}\mathcal{B}$ and bandwidth $\delta\mathcal{B}$.

Analysis. Note that the value of a is $(1-\delta)\mathcal{B}^2+\mathcal{B}$, the value of c is $(1-\delta)\mathcal{B}^2+\delta\mathcal{B}$, and the value of any b_i is $\delta\mathcal{B}$. Assume call a is served (otherwise the algorithm is 0-competitive). If the algorithm preempts a to serve some b_i, then the algorithm is at most $\frac{\delta\mathcal{B}}{(1-\delta)\mathcal{B}^2+\mathcal{B}} = O\left(\frac{1}{\mathcal{B}}\right)$-competitive. Therefore, we can assume that all the b_i calls are rejected. Calls a and c both cannot be served. Thus, the value gained by the algorithm (by serving either a or c) is at most $(1-\delta)\mathcal{B}^2+\mathcal{B}$. The subsequence $S' = \{b_0, \ldots, b_{\mathcal{B}-2}, c\}$ is feasible with $V(S') = \mathcal{B}^2$. Hence, the competitive ratio of the algorithm is at most $\frac{(1-\delta)\mathcal{B}^2+\mathcal{B}}{\mathcal{B}^2} = (1-\delta)+o(1)$.

The case $\delta < \frac{1}{2}$. We show that for $\frac{1}{3} < \delta < \frac{1}{2}$ no algorithm can be more than $\frac{1}{2}$-competitive and that for $0 < \delta \le \frac{1}{3}$ no algorithm can be more than $\frac{2}{3}$-competitive. This impossibility result is shown for the parallel links model. In this model, the bandwidth of all calls is exactly $\frac{\mathcal{B}}{k}$ for some integer $k > 1$. This can be viewed as if the two stations are connected via k parallel links, and each call occupies exactly one link for its duration. (For convenience we assume that calls can be transferred between links during service at no cost.) Clearly, this model is a special case of our single link model.

Let r_k be an upper bound on the competitiveness of any online algorithm in the parallel links model. We show that in the single link model no online algorithm can be more than r_k competitive, in case a call may request at most $\delta\mathcal{B}$ bandwidth, for $\delta > \frac{1}{k+1}$. Consider any online algorithm $\mathcal{A}$ for the single link model. Let $\mathcal{A}'$ be the algorithm for the k parallel links model that corresponds to algorithm $\mathcal{A}$. Let $c_1, c_2, \ldots$ be a sequence of calls that causes $\mathcal{A}'$ to be at most r_k competitive in the k parallel links model. We claim that this sequence in which the bandwidth of each c_i is $\delta\mathcal{B}$ also causes $\mathcal{A}$ to be at most r_k competitive in the single link model. To see this, note that since $\delta > \frac{1}{k+1}$ it follows that there cannot be $k+1$ calls that are served concurrently. Therefore, at any point in time, the calls in service can be viewed as arranged on k parallel links such that the width of each link is $\delta\mathcal{B}$.

It remains to bound r_k. We remark that there exist algorithms for the parallel links model that have slightly better competitiveness than our algorithms for the single link model. Algorithms for $k = 1$ and $k = 2$ were previously known [13, 6]. A simple extension of them yields algorithms for even k with a competitive ratio of $\frac{1}{2}$ and for odd k with competitive ratio $\frac{1}{2} - \frac{1}{4k}$.

We first describe a simple impossibility result for $k = 2$ (the case $k = 1$ was dealt with in [13], as outlined above). At time 0, request two calls of length x. If the algorithm serves only one call, then we are done. At every time unit $0 \le i \le x-1$, request two service calls of length 1. If the algorithm takes one of these service calls it must preempt one of the original calls. In this case, the sequence ends and the algorithm achieves competitiveness of $\frac{x+1}{2x} = \frac{1}{2}+o(1)$. Otherwise, at time $x-1$ request two new requests of length x. In this case the competitiveness is $\frac{2x}{4x-2} = \frac{1}{2}+o(1)$. For the cases $k = 3$ and $k = 4$ we have an impossibility result of $\frac{1}{2}$ and $\frac{11}{20}$, respectively. However, the proofs are tedious and are omitted.

We now describe the impossibility result for the competitiveness of any bandwidth allocation algorithm in this model for all $k > 1$. We show by induction on k that the competitive ratio for k links is at most r, where r satisfies $e^r = \frac{r}{1-r}$, in particular $r \le \frac{2}{3}$.

The basic strategy is an extension of the strategy for the two lines case. At time 0, request k calls of length x and at time $x-1$ request another set of k calls of length

x. We refer to these calls as *long* calls. During the times $[0 \dots x-1]$ many service calls (*short* calls) will be offered in a way that the offline algorithm will be able to cover this range and therefore will have a throughput of $(2x-1)k$. We will choose $x \gg 1$ and therefore, without loss of generality, assume that the throughput of the offline is $2xk$. We do not know how to prevent the algorithm from serving all the service calls and thus to achieve a $\frac{1}{2}$ bound. In what follows we describe how to prevent the algorithm from serving some of the service calls to achieve the $\frac{2}{3}$ bound.

A general scenario of the execution of the online algorithm is as follows. The online algorithm starts by serving $w \leq k$ out of the k long calls requested at time 0 (and all the short calls that can be served). In case the adversary does not stop the request sequence, after t_1 units of time the online algorithm preempts one of the w calls to serve a service call. Again, in case the adversary does not stop the request sequence, after an additional t_2 units of time it preempts another long call and so on, for $w-z$ times. Note that by definition of $t_1, \dots, t_{w-z}$ we must have that $\sum_{j=1}^{w-z} t_j \leq x$.

At time $x-1$ the online algorithm remains with z long calls. In case the adversary does not stop the request sequence, k long calls are requested at this time. In this case the throughput of the online algorithm will be xk plus the sum of the lengths of all the service calls it served before time $x-1$.

Assume first a very simple strategy for the adversary. The sequence will have only two types of calls: long calls of length x and short calls of unit length. Assume further that the algorithm is r-competitive. Then the following inequalities must hold:

$$\begin{aligned}
\frac{xw}{xk} &\geq r && (0),\\
\frac{t_1(k-w)+x(w-1)}{xk} &\geq r && (1),\\
\frac{t_1(k-w)+t_2(k-w+1)+x(w-2)}{xk} &\geq r && (2),\\
&\vdots\\
\frac{\sum_{j=1}^{i} t_j(k-w+j-1)+x(w-i)}{xk} &\geq r && (i),\\
&\vdots\\
\frac{\sum_{j=1}^{w-z} t_j(k-w+j-1)+xz}{xk} &\geq r && (w-z),\\
\frac{\sum_{j=1}^{w-z} t_j(k-w+j-1)+xk}{2xk} &\geq r && (k).
\end{aligned}$$

Inequality (0) is true since otherwise the adversary could stop immediately after presenting the k long calls and thus preventing the algorithm from being r competitive. The rest of the inequalities are true since otherwise the adversary could stop after the jth preemption of a long call by the algorithm.

The value of r is maximized when all the inequalities become equalities. Note that although this solution may not correspond to a valid adversary strategy (in case it does not obey the integrality constraints) it can still be used to upper bound the

competitiveness. By comparing the $(j-1)$th equality with the jth equality we get that for $1 \le j \le w-z$

$$t_j(k-w+j-1) = x \ .$$

Thus equality (k) is equivalent to

$$\frac{w-z+k}{2k} = r \ .$$

The above equality together with equality (0) implies that

$$w+z = k \ .$$

In addition, plugging the value of t_j in the equality $\sum_{j=1}^{w-z} t_j = x$ we have

$$\sum_{j=1}^{w-z} \frac{x}{k-w+j-1} = x \ .$$

This is equivalent to

$$\frac{1}{k-w} + \frac{1}{k-w+1} + \cdots + \frac{1}{k-z-1} = 1 \ .$$

Using the estimation $\sum_{j=1}^{n} \frac{1}{j} = \ln n + O(1)$, we get that $\sum_{j=m}^{n} \frac{1}{j} > \sum_{j=m+1}^{n+1} \frac{1}{j} \approx \ln\left(\frac{n+1}{m}\right)$. Substituting $n = k-z-1$ and $m = k-w$, it follows that

$$\ln\left(\frac{k-z}{k-w}\right) < 1 \ .$$

This implies that $\frac{k-z}{k-w} < e$. Since $w = k-z$ we get that $\frac{w}{k-w} < e$, and since $w = kr$ we get that $\frac{r}{1-r} < e$, which is equivalent to

$$r < \frac{e}{1+e} \approx 0.731059 < \frac{3}{4} \ .$$

Now, assume a more complicated strategy for the adversary. Note that in the simple strategy the adversary does not try to minimize the number of service calls served by the online algorithm; that is, whenever the online algorithm has a "free" line it can use the line to serve service calls. In the more complicated strategy the adversary tries also to minimize the number of service calls served by the online algorithm. This is done by using the strategy "recursively." Whenever the algorithm is serving j long calls and has $k-j$ free lines (for $w \le j \le z$), the adversary offers service calls according to the strategy for $k-j$ parallel links. Namely, instead of offering service calls of a fixed size, the service calls become the long calls for $k-j$ parallel links, while scaling down the length of the service calls for the $k-j$ lines accordingly. Furthermore, we assume by induction that the bound r can be achieved for any number of lines less than k. We have already seen that the inductive claim holds for $k = 2$.

The analysis is almost the same as the one for the simpler adversary. Here, we add a factor of r in the inequalities as follows:

$$\frac{\sum_{j=1}^{i} t_j(k-w+j-1)r + x(w-i)}{xk} \ge r \ .$$

Again, the value of r is maximized when all the inequalities become equalities. This implies that

$$t_j(k-w+j-1)r = x,$$

which as before yields the equality

$$w + z = k\ .$$

In addition, we get that

$$\frac{1}{k-w} + \frac{1}{k-w+1} + \cdots + \frac{1}{k-z-1} = r$$

and therefore

$$\ln\left(\frac{k-z}{k-w}\right) < r\ .$$

Using arguments as before, we get that

$$r < \frac{e^r}{1+e^r}\ .$$

This inequality does not hold for $r > 0.66$. Hence, we proved that

$$r < \frac{2}{3}\ .$$

Line networks with arbitrary durations. Finally, we consider networks with a line topology in which the calls have arbitrary duration (rather than unit time duration). In this case the throughput of a call is the product of its length, bandwidth, and duration. Suppose that the line consists of $n+1$ stations, all the links have the same bandwidth $\mathcal{B}$, and the maximum allowable bandwidth of a single call is $\delta\mathcal{B}$, for $\delta > \frac{1}{2}$. We show that any deterministic online algorithm for this model is at most $1/n$-competitive. The sequence and the analysis are as in the case of the single link model with $\delta = 1$, where instead of bandwidth $\mathcal{B}$ we consider a line of nodes $0, \ldots, n$. Specifically, we simulate the procedure of the adversary presented in the impossibility result for the single link model with $\delta = 1$. In the simulation, whenever a square call a_i is requested in the procedure for the single link model with $\delta = 1$, the adversary here requests a square call of bandwidth $\delta\mathcal{B}$ at time in that connects nodes 0 and n, for n time units. Note that the throughput of this call is $n^2\delta\mathcal{B}$. Similarly, instead of requesting the jth slice call b that intersects the square call a_i as described in the impossibility result for the single link model with $\delta = 1$, the request made here is for a call at time $in + j$ that connects nodes j and $j+1$ for n^2 time units. Again, the throughput of this call is also $n^2\delta\mathcal{B}$. Since $\delta > \frac{1}{2}$, a square call and a slice call that intersect cannot both be served. Thus, following the analysis for the single link case with $\delta = 1$ we get the $1/n$ upper bound.

Acknowledgment. We are indebted to Hugo Krawczyk for very helpful discussions at the early stages of this work.

REFERENCES

[1] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts, *Online load balancing with applications to machine scheduling and virtual circuit routing*, in Proceedings Twenty-Fifth ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 623–631.

[2] B. Awerbuch, Y. Azar, and S. Plotkin, *Throughput-competitive of online routing*, in Proceedings Thirty-Fourth IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 32–40.

[3] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén, *Competitive non-preemptive call control*, in Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1994, pp. 312–320.

[4] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani, *Online admission control and circuit routing for high Performance computing and communication*, in Proceedings Thirty-Fifth IEEE Symposium on Foundations of Computer Science, Sante Fe, NM, 1994, pp. 412–423.

[5] *Special Issue on Asynchronous Transfer Mode*. Internat. Digital Analog Cabled Systems, 1 (1988).

[6] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang, *On the competitiveness of online real-time task scheduling*, in Proceedings Twelfth IEEE Symposium on Real Time Systems, San Antonio, TX, 1991, pp. 106–115.

[7] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, *Online scheduling in the presence of overload*, in Proceedings Thirty-Second IEEE Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1991, pp. 101–110.

[8] R. Canetti and S. Irani, *Bounding the power of preemption in randomized scheduling*, SIAM J. Comput., 27 (1998), pp. 993–1015.

[9] I. Cidon and I. Gopal, *PARIS: An approach to integrated high-speed private networks*, Internat. J. Digital Analog Cabled Systems, 1 (1988), pp. 77–86.

[10] P. F. Chimento, J. E. Drake, L. Gun, W. A. Hervatic, C. P. Immanuel, G. A. Marin, R. O. Onvural, S. A. Owen, and T. E. Tedijanto, *Broadband Network Services for High Speed Multimedia Networks*, Tech report 29.1761, IBM, Research Triangle Park, NC, 1993.

[11] U. Faigle and W. M. Nawijn, *Note on scheduling intervals online*, Discrete Appl. Math., 58 (1995), pp. 13–17.

[12] J. A. Garay and I. S. Gopal, *Call preemption in communication networks*, in Proceedings INFOCOM '92, Florence, Italy, 1992, pp. 1043–1050.

[13] J. A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung, *Efficient online call control algorithms*, in Proceedings Second Israel Conference on Theory of Computing and Systems, Netanya, Israel, 1993, pp. 285–293.

[14] G. Koren and D. Shasha, *D^{over}: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems*, SIAM J. Comput., 24 (1995), pp. 318–339.

[15] G. Koren and D. Shasha, *MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling*, Theoret. Comput. Sci., 128 (1994), pp. 75–97.

[16] R. J. Lipton and A. Tomkins, *Online interval scheduling*, in Proceedings Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1994, pp. 302–311.

[17] C. Lund, S. Phillips, and N. Reingold, *IP over connection-oriented networks and distributional paging*, in Proceedings Thirty-Fifth IEEE Symposium on Foundations of Computer Science, Santa Fe, NM, 1994, pp. 424–434.

[18] K. K. Ramakrishnan, L. Vaitzblit, C. Gray, U. Vahalia, D. Ting, P. Tzelnic, S. Glaser, and W. Duso, *Operating system support for a video-on-demand file service*, in Proceedings Network and Operating System Support for Digital Audio and Video: Fourth International Workshop, Lecture Notes in Comput. Sci. 846, D. Shepherd, G. Blair, G. Coulson, N. Davies, F. Garcia, eds., Springer-Verlag, Berlin, New York, 1994, pp. 216–227.

[19] N. Shacham, *Preemption based admission control in multi media multi party communications*, in Proceedings INFOCOM '95, Boston, MA, 1995, pp. 827–834.

[20] D. B. Shmoys, J. Wein, and D. P. Williamson, *Scheduling parallel machines on-line*, SIAM J. Comput., 24 (1995), pp. 1313–1331.

[21] F. Toutain and O. Huber, *A general preemption-based admission control policy using smart market approach*, in Proceedings INFOCOM '96, San Francisco, CA, 1996, pp. 794–801.

[22] F. Wang and D. Mao, *Worst Case Analysis for On-Line Scheduling in Real-Time Systems*, Tech. report 91-54, Department of Computer and Information Science, University of Massachusetts, Amherst, MA, 1991.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1829–1847

AN OPTICAL SIMULATION OF SHARED MEMORY*

LESLIE ANN GOLDBERG†, YOSSI MATIAS‡, AND SATISH RAO§

Abstract. We present a work-optimal randomized algorithm for simulating a shared memory machine (PRAM) on an optical communication parallel computer (OCPC). The OCPC model is motivated by the potential of optical communication for parallel computation. The memory of an OCPC is divided into modules, one module per processor. Each memory module only services a request on a timestep if it receives exactly one memory request.

Our algorithm simulates each step of an $n \lg \lg n$-processor EREW PRAM on an n-processor OCPC in $O(\lg \lg n)$ expected delay. (The probability that the delay is longer than this is at most $n^{-\alpha}$ for any constant α.) The best previous simulation, due to Valiant, required $\Theta(\lg n)$ expected delay.

Key words. PRAM, PRAM simulation, optical networks

AMS subject classifications. 68Q22, 68R05

PII. S0097539795290507

1. Introduction.

The huge bandwidth of the optical medium makes it possible to use optics to build communication networks of very high degree. Eshaghian [8, 9] first studied the computational aspects of parallel architectures with complete optical interconnection networks. The OCPC model is an abstract model of computation which formalizes important properties of such architectures. It was first introduced by Anderson and Miller [2] and Eshaghian and Kumar [10]. In an n-processor *completely connected optical communication parallel computer* (n-OCPC) n processors with local memory are connected by a complete network. A computation on this computer consists of a sequence of communication steps. During each communication step each processor can perform some local computation and then send one message to any other processor. If a processor is sent a single message during a communication step, then it receives this message successfully, but if it is sent more than one message, then the transmissions are garbled and it receives none of them.

While the OCPC seems a reasonable model for optical computers, it has not been used as a programming model to date. The PRAM model, however, has been used extensively for parallel algorithmic design (e.g., [19, 22, 34]). The convenience of programming on the PRAM is largely due to the fact that the programmer does not have to specify interprocessor communication or to allocate storage in a distributed memory. For the very same reason, the PRAM is considered highly theoretical, and the task of emulating the PRAM on more realistic models has attracted considerable attention; emulations may enable automatic mapping of PRAM algorithms to weaker models, as well as a better understanding of the relative power of different models.

*Received by the editors August 21, 1995; accepted for publication (in revised form) September 23, 1997; published electronically May 13, 1999. A preliminary version of this paper appeared in Proc. 6th ACM Symp. on Parallel Algorithms and Architectures, June 1994.

http://www.siam.org/journals/sicomp/28-5/29050.html

†Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK (leslie@dcs.warwick.ac.uk). Part of the work of this author was performed at Sandia National Laboratories and was supported by the U.S. Department of Energy under contract DE-AC04-76AL85000. Part of this work was supported by ESPRIT LTR Project 20244—ALCOM-IT and ESPRIT Project 21726—RAND-II.

‡AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974 (matias@research.att.com).

§NEC Research Institute, 4 Independence Way, Princeton, NJ 08540 (satish@research.nec.com).

Indeed, many emulations of the PRAM on bounded degree networks were introduced (see, e.g., [1, 21, 23, 31, 32, 36, 37] or [24] for a survey).

In this paper, we present a simulation of an EREW PRAM on the OCPC. In particular, we present a randomized simulation of an $n \lg \lg n$-processor EREW PRAM on an n-processor OCPC in which, with high probability, each step of the PRAM requires $O(\lg \lg n)$ steps on the OCPC.[1] Our simulation is work optimal, to within a constant factor.

Our results are closely related to previous work on the well-studied *distributed memory machine* (DMM), which consists of n processors and n memory modules connected via a complete network of communication. Each processor can access any module in constant time, and each module can service at most one memory request (read or write) at any time. The DMM is thus a weaker model than the shared memory PRAM in that the memory address space is partitioned into modules with a restricted access imposed on them. We remark that there are several variants of DMM models that differ in their contention rules.

Several papers have studied the emulation of a PRAM on various DMM models [31, 21, 38, 35, 6, 20, 7]. Karp, Luby, and Meyer auf der Heide [20] present $O(\lg \lg n)$ expected delay simulations of various types of PRAM on a CRCW DMM in which each memory module allows concurrent read or write access to at most one of its memory locations during any step. Dietzfelbinger and Meyer auf der Heide [7] improve upon this paper by presenting an $O(\lg \lg n)$ expected delay simulation of an EREW PRAM on the (weaker) c-collision DMM in which any memory module that receives c or fewer read or write requests serves all of them. Although Dietzfelbinger and Meyer auf der Heide require $c \geq 3$ for their analysis to work, they report that experiments show that $c = 2$ works as well. The 1-collision DMM is equivalent to the OCPC.

Our result improves on the result of [7] in two ways. First, it is work-optimal. Second, it works for the OCPC (or 1-collision DMM). The previous best-known work-optimal simulation of a PRAM on the OCPC is an $O(\lg n)$ delay simulation of Valiant [39]. In addition, unlike [39, 7] we explicitly consider the construction and evaluation of the hash functions used in our simulation algorithm.

1.1. Related work.

1.1.1. The OCPC model. The OCPC model was first introduced by Anderson and Miller [2] and Eshaghian and Kumar [10] and has been studied by Valiant [39], Geréb-Graus and Tsantilas [12], Gerbessiotis and Valiant [11], Rao [33], Goldberg et al. [17], and Goldberg, Jerrum, and MacKenzie [18]. The feasibility of the OCPC from an engineering point of view is discussed in [2, 12]. See also the survey paper of McColl [30] and the references therein.

1.1.2. Computing *h*-relation on the OCPC. A fundamental problem that deals with contention resolution on the OCPC is that of realizing an *h-relation*. In this problem, each processor has at most h messages to send and at most h messages to receive. Following Anderson and Miller [2], Valiant [39], and Geréb-Graus and Tsantilas [12], Goldberg et al. [17] solved the problem in time $O(h + \lg \lg n)$ for an n-processor OCPC. A lower bound of $\Omega(\sqrt{\lg \lg n})$ expected time was recently obtained by Goldberg, Jerrum, and MacKenzie [18].

1.1.3. Simulating PRAMs on OCPCs. Valiant described a simulation of an EREW PRAM on an OCPC in [39]. More specifically, Valiant gave a constant delay

[1] We will refer to the time required to simulate one PRAM step as the *delay* of the simulation.

simulation of a bulk synchronous parallel (BSP) computer on the OCPC (there called the S*PRAM), and also gave an $O(\lg n)$ randomized simulation of an $n \lg n$-processor EREW PRAM on an n-processor BSP computer. A simpler simulation with delay $O(\lg n \lg\lg n)$ was given by Geréb-Graus and Tsantilas [12]. Valiant's result is the best previously known simulation of a PRAM on the OCPC.

Independently of our work, MacKenzie, Plaxton, and Rajaraman [28] and Meyer auf der Heide, Scheideler, and Stemann [27] have shown how to simulate an n-processor EREW PRAM on an n-processor OCPC. Both simulations have $\Theta(\lg\lg n)$ expected delay. However, neither simulation is work-optimal, and both simulations require $n^{\Omega(1)}$ storage at each processor.

1.1.4. Simulating PRAMs on DMMs. Mehlhorn and Vishkin [31] used a $(\lg n/\lg\lg n)$-universal class of hash functions to achieve a simple simulation of a CRCW PRAM on a CRCW DMM with expected delay $O(\lg n/\lg\lg n)$. An n-processor CRCW PRAM can be simulated on an n-processor EREW DMM in $O(\lg n)$ expected delay using techniques from [39]. The work of this simulation is thus a $\Theta(\lg n)$ factor away from optimality. The best work-optimal simulation of a PRAM on an EREW DMM has delay $O(n^{\epsilon})$ [23].

Recently, Karp, Luby, and Meyer auf der Heide [20] presented a simulation of an n-processor CRCW PRAM on an n-processor CRCW DMM with $O(\lg\lg n)$ delay. They also presented a work-optimal simulation of an $(n \lg\lg n \lg^* n)$-processor EREW PRAM on an n-processor CRCW DMM in $O(\lg\lg n \lg^* n)$ expected delay and a nearly work-optimal simulation of an $n \lg\lg n$-processor CRCW PRAM on an n-processor CRCW DMM with the same delay. Subsequently, Dietzfelbinger and Meyer auf der Heide [7] presented a simplified (nonoptimal) simulation of an n-processor EREW PRAM on an n-processor DMM with $O(\lg\lg n)$ expected delay. The simulation in [20] introduces a powerful technique that incorporates the use of two or three hash functions to map the memory address space into the memory modules, combined with the use of a CRCW PRAM algorithm for perfect hashing (see [16] and references therein). It heavily uses the concurrent read capability of the CRCW DMM. The simulation in [7] circumvents the need for using the CRCW PRAM perfect hashing by an elegant use of an idea from Upfal and Wigderson [38].

1.2. Overview of the algorithm. Our simulation algorithm incorporates techniques and ideas from the simulation algorithms of [20, 7], as well as from the h-relation routing algorithm of [17], as follows.

The simulation in [7] uses three hash functions to map each memory cell of the EREW PRAM to three processors (and memory cells) in the DMM. A write on an EREW memory cell is implemented by writing a value and a time stamp to at least two out of the three associated DMM memory cells. A read of an EREW memory cell is implemented by reading two out of three of the memory cells and choosing the value with the most recent time stamp. Dietzfelbinger and Meyer auf der Heide's proof that their simulation requires only $O(\lg\lg n)$ delay on a 3-collision DMM relies on the fact that, given a randomly generated tripartite hypergraph on $3n$ nodes with ϵn edges, one can, with high probability, remove all the nodes in the hypergraph by using the following process.

Repeat $O(\lg\lg n)$ times:

1. Remove all of the nodes with degree at most 3.
2. Remove all resulting trivial hyperedges (hyperedges in which only one incident node remains.)

Each hyperedge corresponds to a read or write of a PRAM memory location: the three vertices correspond to the three processors in the DMM associated with that memory location. Thus, one step of an ϵn node EREW PRAM is implemented by using the process above to deliver at least two out of three of the messages associated with each memory request.

Since we are simulating an $n \lg\lg n$-processor PRAM on an n-node OCPC, we must simultaneously implement the process above for $O(\lg\lg n)$ $3n$-node hypergraphs using only n processors. To do this, we start by sparsifying all of the hypergraphs using ideas from the $(\lg\lg n)$-relation routing algorithm in [17]. That is, we route all but $O(n/\lg^c n)$ messages and we ensure that at most one undelivered message remains at any processor. Even so, implementing the process above in parallel could still require $\Omega(\lg\lg n)$ timesteps per iteration since each destination may participate in as many as $(\lg\lg n/\epsilon)$ different hypergraphs. Thus, we must also "copy" each destination in such a manner that each message can locate the appropriate copy of its destination. We then perform the process in each hypergraph, ensuring that the process delivers at most a constant number of messages to each copy of a destination. After that, the messages can be sequentially forwarded to their true destinations in $O(\lg\lg n)$ time.

We remark that, in fact, we cannot directly perform the process above on any of the $O(\lg\lg n)$ hypergraphs since our processors can only receive one message in a timestep whereas the processors in [7] can receive three messages in a timestep. The details of our solution to this problem can be found in the technical sections.

1.3. Paper outline. We proceed in section 2 with a high-level description of our simulation. In section 3, we present our algorithm in detail and prove correctness. In section 4 we deal with the evaluation of the hash function that maps the virtual shared memory to the memory modules.

2. The simulation. Our objective is to show how to simulate one step of an $n \lg\lg n$ processor EREW PRAM in $O(\lg\lg n)$ timesteps on an n-processor OCPC. Our simulation follows [7] in using the following idea from [38]. The memory of the PRAM is hashed using three hash functions, h_1, h_2, and h_3. Thus, each memory cell of the PRAM is stored in three memory cells of the OCPC. To write memory cell x, a processor of the OCPC sends a message to at least two of the processors in $\{h_1(x), h_2(x), h_3(x)\}$. The message contains the new value for cell x and also a time stamp. To read memory cell x, a processor p of the OCPC sends a message to at least two of the processors in $\{h_1(x), h_2(x), h_3(x)\}$. Each of these two processors sends p the value that it has for cell x and also its time stamp for cell x. Processor p uses the value with the later timestep. The hash functions h_1, h_2, and h_3 are chosen from the "highly" universal family $\overline{R}^{d,j}_{m,n}$ from [20], which guarantees randomlike behavior.

Each OCPC processor will simulate $\lg\lg n$ PRAM processors. Thus, at the start of a PRAM step, each of the OCPC processors will wish to access up to $\lg\lg n$ cells of the PRAM memory. Each processor uses h_1, h_2, and h_3 to obtain the three destinations where each memory cell is stored. Thus, each OCPC processor wants to send messages to up to $3\lg\lg n$ destinations. Our objective is to deliver at least two of the messages associated with every request.

As in [17], we will divide the processors of the OCPC into target groups of size $k = \lg^c n$. We will also divide the $n \lg\lg n$ memory requests into $\lg\lg n/\epsilon$ groups of ϵn requests each for a sufficiently small constant ϵ. We will refer to the set of messages associated with a particular group of memory requests as a "group of messages." The messages will be delivered using the following procedures:

- *Thinning and deliver to target groups.* Initially, the number of messages destined for any given target group may be as high as $4k \lg\lg n$. (We will show that, with high probability, it is no larger than this.) We will use techniques from [17] to route the messages to their target groups. With high probability, when this procedure is finished every message will be in the target group of its destination. Furthermore, each processor will have at most one message left to send. For a sufficiently large constant c_2, we will allocate a contiguous block of c_2 processors from the target group to each unfinished message for that destination. All senders will know which processors are allocated for their destination. For a sufficiently large constant c_1, we will ensure that for any of the $\lg\lg n/\epsilon$ groups of ϵn messages, with high probability, all but $O(n/(\lg n)^{c_1})$ of the messages in the group will be delivered to their final destinations.
- *Divide into subproblems and duplicate.* We now divide the OCPC into $\lg\lg n/\epsilon$ sub-OCPCs, each with $n' = n\epsilon/\lg\lg n$ processors. Each sub-OCPC will work on the subproblem of delivering the messages corresponding to a particular group of messages. For each sub-OCPC we now make $\lg^2 n'$ copies of the relevant subproblem, all of which will reside in its processors $1, \ldots, n'/2$. We will also allocate its processors $n'/2 + 1, \ldots, n'$ as follows. For each outstanding memory request (i.e., for each memory request which has the property that at most one of its three messages was delivered during the previous procedure), we will allocate $\lg^2 n'$ processors. These $\lg^2 n'$ processors will do the bookkeeping concerning the request in the $\lg^2 n'$ copies of the subproblem. Each message will know the identity of the processors responsible for the bookkeeping concerning its memory request.
- *Route messages for each subproblem.* In each copy of each subproblem we route messages according to the c_2-collision access schedule from section 3 of [7]. Dietzfelbinger and Meyer auf der Heide prove that with high probability each subproblem is "good" (this term will be defined later on). We will prove that if a subproblem is good, then for any particular memory request in any particular copy of the subproblem, the probability that the memory request is satisfied in the c_2-collision access schedule routing is at least $1/2$. Also, no destination in any copy of any subproblem receives more than a constant number $(3c_2)$ of messages during the c_2-collision access schedule routing.
- *Combining problem copies and combining subproblems.* In this procedure we identify a subset S of the set of messages that were delivered by the various copies of the c_2-collision access schedule routing procedure. The messages in S are chosen in such a way that every processor is the destination of $O(\lg\lg n)$ messages in S. We show that with high probability every memory request in every subproblem that was created in the "divide into subproblems and duplicate" procedure will be satisfied if the messages in S are delivered. We deliver the messages in S, using the routing algorithm in [17].

3. Simulation details and analysis. Before giving the details and analysis, we define the class of hash functions $\overline{R}^{d,j}_{m,n}$ being used and describe its properties that are used in the analysis. In the subsequent subsections we will give the details of each of the procedures described in the previous section.

3.1. The hash functions. The class $\overline{R}^{d,j}_{m,n}$ is taken from [20] and is defined as follows.

DEFINITION OF $\overline{R}^{d,j}_{m,n}$. *A function from* $\overline{R}^{d,j}_{m,n}$ *is a combination of functions taken from several classes. Carter and Wegman* [4] *introduced* $H^d_{m,n} \subseteq \{g : [1,\ldots,m] \to [1,\ldots,n]\}$, *the class of universal functions* $P(x) \bmod n$, *where* P *is a polynomial of degree* $d-1$ *over* $[1,\ldots,m]$. *Siegel* [35] *introduced a class of functions* $\overline{H}_{n^j,n} \subseteq \{h : [1,\ldots,n^j] \to [1,\ldots,n]\}$. (*More details on this class are given in section* 4.1.) *To choose a random hash function* $h : [1,\ldots,m] \to [1,\ldots,n]$ *from* $\overline{R}^{d,j}_{m,n}$, *one first chooses*

- *a function* f, *chosen uniformly at random from* $H^d_{m,\sqrt{n}}$;
- *a function* r, *chosen uniformly at random from* $\overline{H}_{n^j,n}$;
- *a function* s, *chosen uniformly at random from* H^1_{m,n^j};
- $\sqrt{n}$ *integers* $a_1,\ldots,a_{\sqrt{n}}$, *each chosen uniformly at random from the range* $[1,\ldots,n]$.

The function h *is defined by* $h(x) = (r(s(x)) + a_{f(x)}) \bmod n$.

As in [20], we say that a family $H_{p,n}$ of hash functions is (μ,k)-universal if for each $x_1 < \cdots < x_j \in \{1,\ldots,p\}$, $\ell_1,\ldots,\ell_j \in \{1,\ldots,n\}$, $j \le k$, the following holds: If the hash function h is drawn uniformly at random from $H_{p,n}$, then $\Pr[h(x_1) = \ell_1,\ldots,h(x_j) = \ell_j] \le \mu/n^j$.

Let ℓ be an arbitrary constant and let j and d be large enough relative to ℓ. Let ϵ' be a sufficiently small positive constant. We will use the following properties of the hash functions with respect to a set $S \subseteq [1,\ldots,m]$, $n \le |S| \le n^{11/10}$. The first two properties are proven in [20].

PROPERTY 3.1. *Let* $\overline{R}^{d,j}_{m,n}(s)$ *be the restriction of* $\overline{R}^{d,j}_{m,n}$ *induced by fixing* $s \in H^1_{m,n^j}$. *If* s *is chosen uniformly at random from* H^1_{m,n^j}, *then* s *is "*1*-perfect" on* S, *with probability at least* $1 - n^{-\ell}$. *If* s *is "*1*-perfect" on* S, *then* $\overline{R}^{d,j}_{m,n}(s)$ *is* $(2, n^{\epsilon'})$*-universal. (Hence,* $\overline{R}^{d,j}_{m,n}(s)$ *is* $(2, n^{\epsilon'})$*-universal with probability at least* $1 - n^{-\ell}$.*)*

Remark. Karp, Luby, and Meyer auf der Heide actually prove a stronger version of Property 3.1, which states that $\overline{R}^{d,j}_{m,n}(s)$ is $(1,\sqrt{n})$-universal with probability at least $1 - n^{-\ell}$. We use the weaker version because (in section 4.1) we will use the space-efficient implementation of the class $\overline{H}_{n^j,n}$ from [35], which is $(2, n^{\epsilon'})$-universal (in fact, it is $((1+o(1)), n^{\epsilon'})$-universal) but is not necessarily $(1,\sqrt{n})$-universal (see Section 2 of [35]). Thus, with the space-efficient implementation, we get only the weaker version of Property 3.1.

PROPERTY 3.2. *Let* f *be drawn randomly from* $H^d_{m,\sqrt{n}}$. *Then with probability at least* $1 - n^{-\ell}$ *every set* $f^{-1}(i) \cap S$ *has size at most* $2|S|/\sqrt{n}$.

We can now derive the third property.

PROPERTY 3.3. *Let* Z *be a subset of* $[1,\ldots,n]$ *and let* i *be an integer in* $[1,\ldots,\sqrt{n}]$. *Suppose that* $\beta \le n^{\epsilon'}$. *Let* h *be chosen randomly from* $\overline{R}^{d,j}_{m,n}$. *(That is, let* f, r, s, *and* $a_1,\ldots,a_{\sqrt{n}}$ *be chosen as described above.) The probability that* β *or more members of* $S \cap f^{-1}(i)$ *are mapped to* Z *by* h *is at most* $2n^{-\ell} + \binom{2|S|/\sqrt{n}}{\beta} 2\left(\frac{|Z|}{n}\right)^{\beta}$.

Proof. By Property 3.2, with probability at least $1 - n^{-\ell}$ every set $f^{-1}(i) \cap S$ has size at most $2|S|/\sqrt{n}$. By Property 3.1, with probability at least $1 - n^{-\ell}$, the hash destinations are $(2, n^{\epsilon'})$-universal. ∎

3.2. Thinning and deliver to target groups. We start by running the "thinning" procedure from [17], which is based on the algorithm of Anderson and Miller [2]. The procedure runs for $O(\lg\lg n)$ steps. During each step each sender chooses a

message uniformly at random from the set of messages that it has not yet sent successfully, and it sends the message to its destination with a certain probability. Let $h = 32e \lg \lg n$. We prove further below the following lemma.

LEMMA 3.1. *With probability at least* $1 - 2n^{-\alpha}$ *(for any constant* α*), after the thinning procedure from* [17] *terminates, there are at most* $k/h\lceil c_3 \lg \lg n\rceil$ *undelivered messages destined for any particular target group.* (c_3 *is a constant which must be sufficiently large; it is the constant* c_2 *from* [17].)

The proof of Lemma 3.1 will use the following lemma.

LEMMA 3.2. *With probability at least* $1 - n^{-\alpha}$ *(for any constant* α*), each target group of size* k *is the destination of at most* $9k \lg \lg n$ *messages.*

Proof. Consider a target group T. By Property 3.1 of the the hash functions, $\overline{R}^{d,j}_{m,n}(s)$ is $(2, n^{\epsilon'})$-universal with high probability. If $\overline{R}^{d,j}_{m,n}(s)$ is $(2, n^{\epsilon'})$-universal, then the probability that at least $9k \lg \lg n$ messages have destinations in T is at most

$$\binom{3n \lg \lg n}{9k \lg \lg n} 2\left(\frac{k}{n}\right)^{9k \lg \lg n},$$

which is at most $2(e/3)^{9k \lg \lg n}$ by Stirling's approximation. □

In order to continue with the proof of Lemma 3.1 we need some notation. For every target group T let $S(T)$ denote the set containing all senders that have messages destined for target group T. We will say that a sender is *bad* if it has some message that have the same destination as at least h other messages. We will use the following lemma.

LEMMA 3.3. *With probability at least* $1 - n^{-\alpha}$ *(for any constant* α*) every set* $S(T)$ *contains at most* $k/(2h^2\lceil c_3 \lg \lg n\rceil)$ *bad senders.*

Proof. This proof is similar to the proof of Claim 2 in [17]. We include it here for completeness and also to demonstrate how the limited independence is handled. Let $h' = h/2$. For a given target group T let $M(S(T))$ denote the set of messages that are sent by senders in $S(T)$. We will say that a message is *externally bad* with respect to a target group T if the message has the same destination as at least h' other messages that are not sent from senders in $S(T)$. We will say that a message is *internally bad* with respect to a target group T if it has the same destination as at least h' other messages that are sent from senders in $S(T)$. We wish to prove that with probability at least $1 - n^{-\alpha}$ at most $k/(2h^2\lceil c_3 \lg \lg n\rceil)$ of the messages in $M(S(T))$ are either externally or internally bad.

First we consider externally bad messages. We will say that a processor P is *externally crowded* with respect to a target group T if there are at least h' messages which are not in $M(S(T))$ and have destination P. A set of b members of a target group are all externally crowded only if at least bh' messages have destinations in the set. Property 3.1 of the hash functions tells us that with high probability the destinations are chosen from a $(2, n^{\epsilon'})$-universal family of hash functions. In this case, as long as $b \le n^{\epsilon'}/h'$ the probability that there is a set of b members of a target group that are all externally crowded is at most $n^{-\alpha}$ (for any constant α),[2] plus

$$\binom{n}{k}\binom{k}{b}\binom{9k \lg \lg n}{bh'} 2\left(\frac{b}{k}\right)^{bh'}.$$

[2]By Lemma 3.2, $n^{-\alpha}$ is an upper bound on the probability that more than $9k \lg \lg n$ messages are destined for any target group.

We can use Stirling's approximation to show that for $b = k/h'^6$ this quantity is at most $(2n/k)(3/5)^{k/h'^5}$. Therefore, with probability at least $1-n^{-\alpha}-(2n/k)(3/5)^{k/h'^5}$ every target group has at most k/h'^6 processors which are externally crowded with respect the T. Suppose that this is the case. Since the family of hash functions is $(2, n^{\epsilon'})$-universal, the probability that $3\,|M(S(T))|/h'^6$ messages in $M(S(T))$ choose a destination which is externally crowded with respect to T is at most

$$2\binom{|M(S(T))|}{3\,|M(S(T))|/h'^6}\left(\frac{1}{h'^6}\right)^{3\,|M(S(T))|/h'^6},$$

which is at most $2(e/3)^{3\,|M(S(T))|/h'^6}$ by Stirling's approximation. Note that, as long as n is sufficiently large, $3\,|M(S(T))|/h'^6 \le k/(4h^2\lceil c_3 \lg\lg n\rceil)$. Also, as long as $|M(S(T))| \ge k/(4h^2\lceil c_3 \lg\lg n\rceil)$ and the constant c (in the definition of k) is sufficiently large, the sum of $(2n/k)(3/5)^{k/h'^5}$ and $(2n/k)(e/3)^{3\,|M(S(T))|/h'^6}$ is at most $n^{-\alpha}$.

We now consider internally bad messages. We start by calculating an upper bound on the probability that a message is internally bad. Lemma 3.2 tells us that with high probability at most $9k \lg\lg n$ messages are destined for any target group. Thus, with high probability, at most $9k \lg\lg n$ messages in $M(S(T))$ are destined for the same target group as the given message. Property 3.1 of the hash functions tells us that with high probability the destinations are chosen from a $(2, n^{\epsilon'})$-universal family of hash functions. Therefore, the probability that the given message is internally bad is at most

$$2\binom{9k \lg\lg n}{h'}\left(\frac{1}{k}\right)^{h'} \le (2/3)^h.$$

So the expected number of messages in $M(S(T))$ which are internally bad is at most $|M(S(T))|(2/3)^h$.

In order to prove that with high probability the number of internally bad messages is not far from the expectation we will use the following theorem of McDiarmid [29]. (The inequality is a development of the "Azuma martingale inequality"; a similar formulation was also derived by Bollobás in [3].)

THEOREM 3.4 (McDiarmid). *Let $x_1, \ldots, x_n$ be independent random variables, with x_i taking values in a set A_i for each i. Suppose that the (measurable) function $f : \prod A_i \to \mathbb{R}$ satisfies $|f(\overline{x}) - f(\overline{x}')| \le c_i$ whenever the vectors $\overline{x}$ and $\overline{x}'$ differ only in the ith coordinate. Let Y be the random variable $f(x_1, \ldots, x_n)$. Then for any $t > 0$,*

$$\Pr\big(|Y - \mathrm{E}(Y)| \ge t\big) \le 2\exp\big(-2t^2/\textstyle\sum_{i=1}^n c_i^2\big).$$

If the hash functions h_1, h_2, and h_3 were chosen uniformly at random from the set of functions from $[1, \ldots, m]$ to $[1, \ldots, n]$, the application of the bounded differences inequality would be straightforward. We would take as the random variable x_i the destination of the ith message in $M(S(T))$. We would let Y be the random variable denoting the number of internally bad messages in $M(S(T))$. If we change the value of one of the x_i, the value of Y would change by at most $h' + 1$. Plugging these values into the inequality, we would get a sufficiently small failure probability.

However, since h_1, h_2, and h_3 are in fact drawn from the family $\overline{R}_{m,n}^{d,j}$, the x_i are not independent so we cannot apply Theorem 3.4 to them. Instead, we follow the approach used in the proof of Lemma 6.1 in [20]. Consider the independent

random variables $a_1, \ldots, a_{\sqrt{n}}$. As before, let Y be a random variable denoting the number of internally bad messages in $M(S(T))$. Let Z be the set of all destinations of messages in $M(S(T))$. (The size of Z is at most $|M(S(T))|$, which is at most $27k(\lg\lg n)^2$ (with high probability), by Lemma 3.2.) Suppose that we change one of the a_i. By Property 3.3 of the hash functions, the probability that β or more members of $M(S(T))$ change destination is at most $2n^{-\alpha} + 2\binom{6n\lg\lg n/\sqrt{n}}{\beta}\left(\frac{27k(\lg\lg n)^2}{n}\right)^{\beta}$. This probability is sufficiently small as long as the constant β is sufficiently large. So suppose that at most β members of $M(S(T))$ change destination. Each of those may make at most $h'+1$ members of $M(S(T))$ become internally bad. Therefore, if we change one a_i we change Y by at most $\beta(h'+1)$. Therefore, by Theorem 3.4 the probability that $Y \geq k/(4h^2\lceil c_3 \lg\lg n\rceil)$ is at most

$$2\exp\left(\frac{-2\left(\frac{k}{4h^2\lceil c_3\lg\lg n\rceil} - \mathrm{E}(Y)\right)^2}{(|M(S(T))|\,\beta^2(h'+1)^2)}\right).$$

(The $|M(S(T))|$ appears in the denominator because there are $|M(S(T))|$ random variables in $\{a_1, \ldots, a_{\sqrt{n}}\}$ that affect the destinations of messages in $M(S(T))$. Changing any of these random variables could change Y by at most $\beta(h'+1)$. Changing any of the other random variables in $\{a_1, \ldots, a_{\sqrt{n}}\}$ does not change Y.) Since $\mathrm{E}(Y) \leq \frac{k}{8h^2\lceil c_3\lg\lg n\rceil}$ (for big enough n) and, with high probability (by Lemma 3.2), $|M(S(T))| \leq 27k(\lg\lg n)^2$, the probability is at most

$$2\exp(-k/(32h^4\lceil c_3\lg\lg n\rceil^2 27(\lg\lg n)^2\beta^2(h'+1)^2)).$$

This quantity is at most $\frac{1}{2}n^{-\alpha}\,(k/n)$ as long as c is sufficiently large. This concludes the proof of Lemma 3.3. □

The following lemma is proved in [17] (just after Lemma 3′). (The proof of the lemma uses the fact that $|S(T)| \leq 9k\lg\lg n$, which is true with high probability, according to Lemma 3.2.)

LEMMA 3.5. *With probability at least $1 - n^{-\alpha}$ the number of messages destined for any target group that start at good senders but are not delivered during the thinning procedure from* [17] *is at most $k/(2h\lceil c_3\lg\lg n\rceil)$.*

Proof of Lemma 3.1. We conclude that with probability at least $1 - 2n^{-\alpha}$ the number of undelivered messages destined for any given target group after the thinning procedure terminates is at most $k/(h\lceil c_3\lg\lg n\rceil)$. □

After the "thinning" procedure from [17] terminates we will use the "spreading" procedure from [17] to spread the unfinished requests so that each processor has at most one unfinished message to deliver. As part of the spreading procedure we will allocate one processor to do the bookkeeping associated with each memory request and we will ensure that all messages associated with the request know the identity of this processor. During this procedure of our simulation the three messages associated with a request may be sent to various processors, but they will keep the bookkeeping processor informed about their whereabouts.

After using "spreading," we will use the "deliver to target groups" procedure from [17] to deliver the rest of the messages to their target groups in $O(\lg\lg n)$ steps. With probability at least $1 - n^{-\alpha}$ (for any constant α) every message will be in its target group at the end of the "deliver to target group" procedure. Furthermore, each sender will have at most two undelivered messages to send, and (by Lemma 3.1) the

number of unfinished messages in a target group will be less than k. At this point we can sort the messages in the target groups by destination. After the sorting, each sender will have at most one message to send.

We now wish to allocate a contiguous block of c_2 processors from the appropriate target group to each unfinished destination (for a sufficiently large constant c_2). We wish to do the allocation in such a way that all senders know which processors are allocated for their destination. We do this as follows. If a destination is the destination of fewer than c_2 requests, we simply deliver them. Otherwise, we allocate c_2 processors for the destination. The processors allocated will be the first c_2 processors with requests for that destination.

At this point we wish to send all but $O(n2^{-c_1 \lg\lg n})$ of the messages in any group to their final destinations. We will say that a message is *bad* if its destination is also the destination of at least $c_1 \lg\lg n$ other messages. We will use the following lemma.

LEMMA 3.6. *With probability at least $1 - n^{-\alpha}$ (for any constant α) at most $O(n2^{-c_1 \lg\lg n})$ of the messages in any group of messages are bad.*

Proof. This proof is similar to the second part of the proof of Lemma 3.3. By Property 3.1 of the hash functions, the destinations are chosen from a $(2, n^{\epsilon'})$-universal family of hash functions with high probability. In this case, the probability that a given message is bad is at most $2\binom{3n \lg\lg n}{c_1 \lg\lg n} n^{-c_1 \lg\lg n}$. By Stirling's approximation, this is at most $2(3e/c_1)^{c_1 \lg\lg n}$, which is at most $2^{-c_1 \lg\lg n}$ for $c_1 \geq 7e$. Therefore, the expected number of bad messages in a group is at most $\epsilon n 2^{-c_1 \lg\lg n}$.

We now use Theorem 3.4 (the bounded differences inequality) to prove that with high probability the number of bad messages in a group is not much more than the expectation.

As in the case of Lemma 3.3, the bounded differences inequality would be straightforward if the hash functions h_1, h_2, and h_3 were chosen uniformly at random from the set of functions from $[1, \ldots, m]$ to $[1, \ldots, n]$. We would take as the random variable x_i the destination of the ith message and we would let Y be the random variable denoting the number of bad messages. If we change the value of one of the x_i, the value of Y would change by at most $c_1 \lg\lg n + 1$. Therefore, we would obtain the following inequality:

$$\Pr(Y \geq 2E) \leq 2\exp(-2E^2/(\epsilon n(c_1 \lg\lg n + 1)^2)).$$

However, since h_1, h_2, and h_3 are in fact drawn from the family $\overline{R}^{d,j}_{m,n}$, we again follow the approach used in the proof of Lemma 6.1 in [20]. Consider the independent random variables $a_1, \ldots, a_{\sqrt{n}}$. Let Y be a random variable denoting the number of bad messages. If we change the value of one of the a_i, then with high probability at most $6n \lg\lg n/\sqrt{n}$ messages get new destinations. (This follows from Property 3.2 of the hash functions.) Each new destination could cause at most $c_1 \lg\lg n + 1$ messages to become bad. Thus, changing one of the a_i could change Y by at most $6\sqrt{n} \lg\lg n(c_1 \lg\lg n + 1)$. Thus, by the bounded differences inequality,

$$\Pr(Y \geq 2E) \leq 2\exp(-2E^2/(\sqrt{n}36n(\lg\lg n)^2(c_1 \lg\lg n + 1)^2)),$$

which is sufficiently small. □

Given Lemma 3.6, it suffices to route $c_1 \lg\lg n$ messages to each destination. This can be done in $O(\lg\lg n)$ steps since the messages are sorted by destination. At this point we have finished the "thinning and deliver to target groups" procedure. The bookkeeping processor associated with every memory request now cancels the request

if at least two of its messages were delivered. If the request is canceled, then the third message is deleted.

3.3. Divide into subproblems and duplicate. Our goal is to divide the OCPC into $\lg\lg n/\epsilon$ sub-OCPSs, each of which has $n' = n\epsilon/\lg\lg n$ processors. Each sub-OCPC will work on the subproblem of delivering the messages corresponding to a particular group of messages. For each sub-OCPC we wish to make $\lg^2(n')$ copies of the relevant subproblem, all of which will reside in its processors $1, \ldots, n'/2$.

We will use an approximate compaction tool to divide the problem into subproblems and to make copies of the problem. (For similar tools see [5, 15, 25, 26].) Given

- an n-OCPC in which at most s senders each have one message to send,
- a set of βs receivers which is known to all of the senders,

the (s, β) *approximate compaction problem* is to deliver all of the messages to the set of receivers in such a way that each receiver receives at most one message.

The following lemma is from [17].

LEMMA 3.7. *For any positive constant α there is a positive constant c_3 such that the $(s, \lceil c_3 \lg\lg n\rceil)$ approximate compaction problem can be solved in $O(\lg\lg n)$ communication steps with failure probability at most $\alpha^{-\sqrt{s}} + s^{-\alpha}$.*

We proved in the previous subsection that with high probability, when the "thinning and deliver to target groups" procedure terminates, the number of undelivered messages is at most $3n\lg\lg n 2^{-c_1 \lg\lg n}$. Furthermore, every message is in the target group of its destination, and each processor will have at most one message left to send.

The number of unfinished target groups is at most the number of unfinished messages, which is at most

$$3n\lg\lg n 2^{-c_1\lg\lg n} \le \frac{n'}{2\lg^2(n')k^2\lceil c_3\lg\lg n\rceil}$$

for a sufficiently large c_1. Therefore, with high probability (by Lemma 3.7), we can compact one message from the first processor in each unfinished target group to the first $n'/(2\lg^2(n')k^2)$ processors in the n-OCPC. Having done that, we can copy each of the unfinished target groups to one of the first $n'/(2\lg^2(n')k)$ target groups in the n-OCPC. Next, we can use doubling to make $\lg^2(n')$ copies of each unfinished target group. All of these copies will reside in the first $n'/(2k)$ target groups in the n-OCPC.

At this point, the entire problem is copied $\lg^2(n')$ times into the first $n'/(2k)$ target groups in the n-OCPC. These $n'/(2k)$ target groups will form the first half of the processors in the first n'-processor sub-OCPC. Our objective is to use the first sub-OCPC to solve the subproblem of delivering the messages in the first group of messages. The sub-OCPC will do this by simply ignoring all messages that are not in the first group of messages.

The $\lg^2(n')$ copies of the entire problem can now be copied into the remaining $\lg\lg n/\epsilon - 1$ sub-OCPCs. The jth sub-OCPC will ignore all messages that are not in the jth group of messages.

Our next goal is to allocate the processors $n'/2, \ldots, n'$ of each sub-OCPC such that for each outstanding memory request (i.e., for each memory request which has the property that at most one of its three messages was delivered during the previous procedure) we allocate $\lg^2(n')$ processors. (These $\lg^2(n')$ processors will do the bookkeeping concerning the request in the $\lg^2(n')$ copies of the subproblem.)

The allocation can be done in the same way that the problem was split and copied because the number of remaining requests is at most $3n \lg\lg n 2^{-c_1 \lg\lg n}$.

3.4. Route messages for each subproblem. Consider a particular copy of a particular subproblem. Lemma 3.6 tells us that with high probability at most $O(n2^{-c_1 \lg\lg n})$ of the memory requests from the ϵn memory requests associated with this subproblem remain. Although each processor has at most one message to send, there is a bookkeeping processor allocated to each memory request and each message knows the identity of its bookkeeping processor. Furthermore, there is a block of c_2 contiguous processors allocated to each unfinished destination and each sender knows which processors are allocated to its destination. For $i \in \{1, 2, 3\}$ we will say that a message is an "i-message" if it obtained its destination using hash function i.

We now route messages according to the c_2-collision access schedule from Section 3 of Dietzfelbinger and Meyer auf der Heide's paper [7]. Each round of the access schedule is defined as follows.

For $i = 1, 2, 3$, do the following:

a. For all destinations d in parallel, repeat $\lceil c_2 \lg(2c_2) \rceil$ times: Each i-message with destination d that is not already waiting at one of the c_2 processors allocated to d picks a random processor from those allocated to d and sends there. Each of the allocated processors will accept only one message.

b. Each destination d now checks whether there are any other i-messages destined for d (that is, whether there are any i-messages with destination d that are not at the allocated processors). To do this, the first of the c_2 processors allocated to d sends to d. Also, any i-messages with destination d that have not yet been successful in reaching one of the c_2 processors allocated to d now send to d. Then the first of the c_2 processors allocated to d tells d whether or not it had a collision.

c. For each destination d, if all of the i-messages destined for d are at the processors allocated to d, then these messages are delivered. Otherwise, no requests are delivered.

d. The bookkeeping processor associated with each memory request checks which of the messages associated with the requests were delivered. If at least two of the messages associated with the request have been delivered, then the request is canceled and the third message is deleted.

Note that no destination receives more than $3c_2$ messages during the c_2-collision access schedule routing. We use the following lemma.

LEMMA 3.8. *During one round of the c_2-collision access schedule routing procedure any processor that is the destination of at most c_2 i-messages gets* all *of the i-messages with probability at least $\frac{1}{2}$ (and none of them with the remaining probability). Any processor that is the destination of more than c_2 i-messages receives none of them.*

Proof. If d is the destination of at most c_2 i-messages, then the probability that one of them fails to reach the allocated processors in $\ell = \lceil c_2 \lg(2c_2) \rceil$ attempts is at most $c_2(1 - 1/c_2)^{\ell} \leq \frac{1}{2}$. □

In their analysis of the c_2-collision access schedule routing procedure (as implemented on a c_2-collision DMM), Dietzfelbinger and Meyer auf der Heide define a hypergraph $H = (V, E)$ for a set of memory requests $x_1, \ldots, x_{\epsilon n}$ with vertex set $V = \{v_{rt} \mid 1 \leq r \leq 3, 1 \leq t \leq n\}$ and hyperedge set

$$E = \{\{v_{1,h_1(x_i)}, v_{2,h_2(x_i)}, v_{3,h_3(x_i)}\} \mid 1 \leq i \leq \epsilon n\}.$$

In light of Lemma 3.8, we can view the c_2-collision access schedule routing as a process on H. In each round, the process removes each node with degree at most c_2 (i.e., the i-messages destined for the processor are delivered) with probability at least $\frac{1}{2}$. Then the process removes each hyperedge that consists of only one node (i.e., memory requests are canceled if at least two of the messages associated with the request are delivered).

Following Dietzfelbinger and Meyer auf der Heide, we will say that H is s-good if

1. the largest connected component in H has at most $\alpha = \alpha(s) \lg n$ nodes,
2. every set $A \subseteq V$ intersects fewer than $|A| + s$ hyperedges from E in at least two points.

Dietzfelbinger and Meyer auf der Heide prove the following lemma. (The proof presented in [7] is based on the assumption that h_1, h_2, and h_3 are chosen uniformly at random from the set of functions from $[1, \ldots, m]$ to $[1, \ldots, n]$. However, the lemma is also true if h_1, h_2, and h_3 are chosen randomly from $\overline{R}^{d,j}_{m,n}$.)

LEMMA 3.9. *The probability that H is s-good is $1 - \mathrm{O}(n^{-s})$.*

We will prove the following lemma.

LEMMA 3.10. *Suppose that H is s-good for some positive constant s. Then the probability that any particular memory request is satisfied after $\mathrm{O}(\lg \lg n)$ rounds of routing according to the c_2-collision access schedule is at least $\frac{1}{2}$.*

Proof. Let H_t denote the hypergraph obtained by applying t rounds of the c_2-collision access schedule routing process to H. Dietzfelbinger and Meyer auf der Heide have made the following observation [7].

OBSERVATION 3.1. *If H is s-good and $A \subseteq V$ is a component of H_t for some $t \geq 0$, then A contains at most $3|A|/(c_2+1) + 3s/(c_2+1)$ nodes of degree larger than c_t in H_t.*

We will use the following lemma.

LEMMA 3.11. *Suppose that H is s-good. Let r be an edge in a component of size $\ell \geq s$ of H_t for some $t \geq 0$. If $c_2 \geq 23$, then with probability at least $1 - \exp(-\ell/54)$ the component of r in H_{t+1} has size at most $5\ell/6$.*

Proof. Let $b = 3(\ell+s)/(c_2+1)$. By Observation 3.1 and Lemma 3.8, the expected number of nodes in the component of r in H_{t+1} is at most $\ell/2 + b/2$. Using a Chernoff bound, we see that the probability that there are at most $4/3(\ell/2 + b/2) \leq 5\ell/6$ nodes is at least $1 - \exp(-(\ell/2 + b/2)/27)$. □

Using Lemma 3.11, we conclude that for some constant $c_4 \geq s$, with probability at least $\frac{3}{4}$, $\mathrm{O}(\lg \lg n)$ rounds of the c_2-collision access schedule routing procedure reduce the size of the component of a given memory request r to at most c_4. We conclude the proof of Lemma 3.10 by observing that, as long as $c_2 > 3s + 2$, $\mathrm{O}(1)$ rounds will, with probability at least $\frac{3}{4}$, further reduce the component to size 1. □

3.5. Combining problem copies and combining subproblems. Let us focus our attention on the jth subproblem. Let S_j be the set of messages that were in the subproblem when it was created. Let S'_j be the subset containing all messages in S_j that are delivered in at least $\lg^2(n')/9$ copies of the c_2-collision access schedule routing procedure.

Note that when the c_2-collision access schedule routing procedure terminates the $\lg^2(n')$ processors per memory request that were allocated in the "divide and copy" procedure to do bookkeeping can inform all of the messages in S_j (in the first copy of the subproblem) whether or not they are in S'_j.

We will prove the following lemma.

LEMMA 3.12. *With probability at least* $1 - n^{-\alpha}$ *(for any positive constant* α*), each set* S'_j *has the following properties.*

1. *Each processor is the destination of at most* $27c_2$ *messages in* S'_j.
2. *Each memory request in the* j*th subproblem will be satisfied if the messages in* S'_j *are delivered.*

If each set S'_j has the properties described in Lemma 3.12 (as it will, with high probability), then we can satisfy all of the memory requests in $O(\lg \lg n)$ steps by routing the messages in $S = \bigcup_j S'_j$. These messages form a $27c_2 \lg \lg n/\epsilon$-relation, so we can use the routing algorithm in [17] to route the messages.

To prove Lemma 3.12 we use the following lemma and the following observation.

LEMMA 3.13. *With probability at least* $1-n^{-\alpha}$ *(for any constant* α*) every memory request in every subproblem is satisfied in at least* $\lg^2(n')/3$ *of the* $\lg^2(n')$ *copies of the* c_2*-collision access schedule routing procedure.*

Proof. Suppose that every subproblem is such that the corresponding hypergraph is s-good. (Lemma 3.9 shows that this is so with high probability, as long as s is chosen to be sufficiently large.) Consider a particular memory request in a particular subproblem. Lemma 3.10 shows that the probability that this request is satisfied in any given copy of the subproblem is at least $\frac{1}{2}$. A Chernoff bound shows that with probability at least $1-ne^{-\lg^2(n')/54}$ the request is satisfied in at least $\lg^2(n')/3$ copies. The lemma follows by summing the failure probabilities over particular memory requests. □

OBSERVATION 3.2. *If* x_1*,* x_2*, and* x_3 *are the three messages in a memory request that is satisfied in at least* ℓ *copies of the* c_2*-collision access schedule routing procedure, then there is a pair of messages from* $\{x_1, x_2, x_3\}$ *such that both of the messages in the pair are satisfied in at least* $\frac{\ell}{3}$ *copies of the procedure. Similarly, if* x_1 *and* x_2 *are the two messages in a memory request that is satisfied in at least* ℓ *copies of the* c_2*-collision access schedule routing procedure, then at least one of* x_1 *and* x_2 *is satisfied in at least* $\frac{\ell}{2}$ *copies of the procedure.*

Proof of Lemma 3.12. The fact that (with high probability) each memory request in the jth subproblem will be satisfied if the messages in S'_j are delivered follows from Lemma 3.13 and from Observation 3.2. To see that each processor is the destination of at most $27c_2$ messages in S'_j note that a message is a member of S'_j only if it is delivered in at least $\lg^2(n')/9$ copies of the c_2-collision access schedule routing procedure. However, we proved in the previous section that each destination will receive at most $3c_2$ messages in each copy of the procedure. Therefore, at most $27c_2$ messages that have the same destination will be included in S'_j. This completes the proof of Lemma 3.12. □

4. Construction and evaluation of the hash function. In the simulation algorithm we have assumed that a hash function h was chosen uniformly at random from the family $\overline{R}^{d,j}_{m,n}$ and is available to every processor for constant time evaluation. When concurrent-read is available in the simulating model, a hash function in use can be kept in the shared memory and be read as necessary in constant time. The exclusive-read nature of the OCPC model, together with the fact that the function $h \in \overline{R}^{d,j}_{m,n}$ is represented by a polynomial number of memory words, imply a more subtle situation. A straightforward implementation is to keep a copy of the function h at each processor. However, this implies polynomial overheads both in the time of preprocessing for distributing all copies and in the space dedicated for this function at each processor. In the remainder of this section we describe an efficient implemen-

tation in which the function requires only a total of linear space and its evaluation increases the simulation delay by at most a constant factor.

4.1. The family of hash functions. Our basic approach is to (i) replace the class $\overline{R}^{d,j}_{m,n}$ with a class whose functions h have similar properties but can be represented in $O(n^\epsilon)$ space, where $\frac{1}{2} \le \epsilon < 1$ (the modified class will still satisfy Properties 3.1–3.3); (ii) make $O(n^{1-\epsilon})$ copies of the selected function h; and (iii) make sure that at each simulation step the number of processors that need to read a component of h is bounded by $O(n^{1-\epsilon} \lg\lg n)$, an average of $O(\lg\lg n)$ per copy, thereby enabling the use of an efficient $\lg\lg n$-relation algorithm for the read operation. (A similar approach of making duplicates to reduce contention was used in [14], in implementing a perfect hash function on the QRQW PRAM.) To implement the approach sketched above we first modify the definition of $\overline{R}^{d,j}_{m,n}$ from section 3.1 as follows. To choose a random hash function $h : [1, \ldots, m] \to [1, \ldots, n]$ from the modified class, one first chooses a function f uniformly at random from $H^d_{m,\sqrt{n}}$ and integers $a_1, \ldots, a_{\sqrt{n}}$ uniformly at random from $[1, \ldots, n]$ as before. Similarly, one chooses the function r uniformly at random from $\overline{H}_{n^j,n}$ as before. However, we will use Siegel's space-efficient implementation of the class $\overline{H}_{n^j,n}$ from [35], which we will explain below, and we will make sure that the implementation satisfies an additional property (see Lemma 4.2). The function s will no longer be chosen uniformly at random from H^1_{m,n^j}. Instead, it will be chosen as follows. Let $t = j/\epsilon$ and let d be a sufficiently large constant. We will choose functions $s_1, s_2, \ldots, s_t$ uniformly at random from the class H^d_{m,n^ϵ}, and we will define $s(x)$ to be the tuple $\langle s_1(x), \ldots, s_t(x)\rangle$. Finally, we will define $h(x) = (r(s(x)) + a_{f(x)}) \bmod n$ as before. The following lemma shows that Property 3.1 still holds for the new family of hash functions.

LEMMA 4.1. *Let $\ell \ge 1$ be arbitrary and let d and j be large enough relative to ℓ. Let $S \subseteq [1, \ldots, m]$, $n \le |S| \le n^{11/10}$. If s is chosen randomly as described above, then* $\Pr[s$ *is* 1*-perfect on* $S]$ *is at least* $1 - n^{-\ell}$.

Proof. The probability that two given distinct points $x, y \in S$ will collide under s, i.e., that $s(x) = s(y)$, is at most $(2/n^\epsilon)^t$, since the s_i are $(2, d)$-universal. The probability that any pair of points from S will collide is therefore at most

$$\binom{|S|}{2}(2/n^\epsilon)^t \le n^{22/10-j}2^{j-1}.$$

The lemma follows by taking $j > \ell + 22/10$. ∎

4.2. The implementation of $\overline{H}_{n^j,n}$. We now describe Siegel's space-efficient implementation of the class $\overline{H}_{n^j,n}$ from [35].

Siegel defines a (p, ϵ, d, h)*-weak concentrator* H as a bipartite graph on the sets of vertices I (inputs) and O (outputs), where $|I| = p$, and $|O| = p^\epsilon$, that has outdegree d for each node in $|I|$ and that has, for any h inputs, edges matching them one-by-one with some h outputs.

A (p, ϵ, d, h)-weak concentrator H is used to construct a function F by storing d random numbers from $[0, \ldots, p-1]$ at each node of O. On input i, $F(i)$ is computed by evaluating a polynomial hash function of degree $d-1$ whose coefficients are determined by the numbers stored at the neighbors of i in O. Siegel showed that the family of hash functions F so defined is a $(1, h)$-universal family of hash functions mapping $[0, p-1] \mapsto [0, p-1]$.

Let H be a $(n^\epsilon, \epsilon, d, n^{\epsilon'})$-weak concentrator. Siegel showed that the Cartesian product $G = H^t$ is a $(n^j, \epsilon, d^t, n^{\epsilon'})$-weak concentrator. The graph G can therefore be

used to construct a $(1, n^{\epsilon'})$-universal family of hash functions mapping $[1, \ldots, n^j]$ to $[1, \ldots, n^j]$. One obtains a $(2, n^{\epsilon'})$-universal family of hash functions mapping $[1, \ldots, n^j]$ to $[1, \ldots, n]$ by taking the results modulo n.

In the following lemma, we observe that Siegel's implementation of $\overline{H}_{n^j,n}$ requires only a graph H with small out-degree. This property will be useful in our OCPC implementation.

LEMMA 4.2. *There exists a graph H that is $(n^{\epsilon}, \epsilon, d, n^{\epsilon'})$-weak concentrator, and which also has the property that every output of H has degree at most $2dn^{\epsilon-\epsilon^2}$.*

Proof. We use a probabilistic construction, as in [35] for finding an $(n^{\epsilon}, \epsilon, d, n^{\epsilon'})$-weak concentrator. Suppose that each input of H chooses its d (distinct) neighbors uniformly at random. Siegel proves that the probability that H is not a $(n^{\epsilon}, \epsilon, d, n^{\epsilon'})$-weak concentrator is at most $n^{-(\epsilon^2-\epsilon')}$. (As long as ϵ' is sufficiently small.) We can now use a Chernoff bound to show that the degree of each output of H is sufficiently small as required. □

4.3. Constructing the hash functions. The graph H from Lemma 4.2 can be constructed and built into the machine when the machine is built. Each of the n^{ϵ} inputs has d neighbors. A set of $n^{1-\epsilon}$ processors is selected and each processor in the set is given the name of these dn^{ϵ} neighbors.

A new hash function h from the modified family is constructed in $O(\lg n)$ steps as follows.

1. Select (appropriately at random) $s_1, \ldots, s_t$ and f and distribute to all processors.
2. Each of the $n^{j\epsilon}$ output nodes of $G = H^t$ chooses d^t values in $[0, \ldots, n^j - 1]$. A set of $n^{1-j\epsilon}$ processors is selected for each given output node and each processor in the set is given the d^t values associated with the output node.
3. The values $a_1, \ldots, a_{\sqrt{n}}$ are generated. $\sqrt{n}$ sets of $\sqrt{n}$ processors are selected and each processor in a set i is given the value of a_i.

Recall that a new function may need to be constructed (a "rehash" operation) when the selected one does not satisfy the required properties. (This occurs with polynomially small probability for each parallel step and with high probability after a polynomial number of steps.)

4.4. Evaluating the hash function. At each simulation step, the hash function is computed for all memory addresses in $O(\lg \lg n)$ time, as described next. Let S be the set of $3n \lg \lg n$ requests from $[1, \ldots, m]$. Recall from section 4.1 that $h(x) = (r(s(x)) + a_{f(x)}) \bmod n$.

Each processor executes the following steps for each request x.

1. Compute $s_1(x), \ldots, s_t(x)$.
2. Compute the names of the neighbors of $\langle s_1(x), \ldots, s_t(x) \rangle$ in G.
3. Read the values corresponding to the neighbors of $\langle s_1(x), \ldots, s_t(x) \rangle$ in G.
4. Apply r to $\langle s_1(x), \ldots, s_t(x) \rangle$.
5. Compute $f(x)$.
6. Read $a_{f(x)}$.
7. Compute $r(s(x)) + a_{f(x)}$.

The executions of steps 1, 4, 5, and 7 are in constant time. The following lemma of Dietzfelbinger [23] is central to the analysis of the other steps.

LEMMA 4.3. *Let $X_1, \ldots, X_n$ be 0–1-valued, d-independent, equidistributed random*

variables. Let $\mu = E(X_i)$. *Then, for* $n \geq d/(2\mu)$,

$$\Pr\left(\sum_{i=1}^{n}(X_i - \mu) \geq \lambda\right) \leq \frac{\alpha(n\mu)^{d/2}}{\lambda^d},$$

where α *is a constant that depends on* d *but not on* n.

CLAIM 4.4. *In step* 2, *with high probability, for every* y *in* $[1, \ldots, n^\epsilon]$ *(i.e., for every input of* H*) there are at most* $O(n^{1-\epsilon} \lg\lg n)$ *pairs* (i, x) *such that* $x \in S$ *and* $s_i(x) = y$.

Proof. Note that the set of values $s_i(x) : 1 \leq x \leq m$ is d-independent. Following Kruskal, Rudolph, and Snir [23] we use Lemma 4.3. Fix a y and an i, and let X_b be a 0–1 random variable, which is 1 if and only if s_i maps the bth member of S to y. Here μ is $1/n^\epsilon$. Let λ be $|S|/n^\epsilon$. Then the probability that s_i maps more than 2λ to y is $O(n^{-d/2(1-\epsilon)})$. Choose d large enough to sum over all i and y. □

We conclude that at most $O(n^{1-\epsilon} \lg\lg n)$ processors want to read the information about input y, and so we have a "target group $O(\lg\lg n)$ relation." The requests can be routed by using [17].

CLAIM 4.5. *In step* 3, *with high probability, for every output* y *of* G *there are at most* $O(n^{1-j\epsilon} \lg\lg n)$ *values* x *in* S *such that* $\langle s_1(x), \ldots, s_t(x)\rangle$ *is a neighbor of* y *in* G.

Proof. Fix $y = \langle y_1, \ldots, y_t\rangle$. Let L_i denote the neighbors of y_i in H. Note that $|L_i| \leq 2dn^{\epsilon-\epsilon^2}$. If $s(x)$ has a neighbor y in G, then $s_i(x)$ is in L_i for $1 \leq i \leq t$.

The probability of this event is at most $(2d/n^{\epsilon^2})^t$. Let X_b be a 0–1 random variable, which is 1 if and only if the bth member x of S has $s(x)$ mapped to y in G. Apply Lemma 4.3: μ is at most $(2d/n^{\epsilon^2})^t$ by Lemma 4.2; let λ be $|S|(2d/n^{\epsilon^2})^t$. The probability that there are more than λ such values x is at most $\alpha n^{-(d/2)(1-j\epsilon)}$. □

Given the claim, we have a "target group $O(\lg\lg n)$ relation." The requests can be routed using [17].

Step 6 remains to be analyzed. By Property 3.2, with probability at least $1 - n^{-\alpha}$ each group needs to be read by at most $6\sqrt{n}\lg\lg n$ of the requests, so we have a "target group $6\lg\lg n$ relation." The requests can be routed by using [17].

5. Conclusions. In this paper we have described a work-optimal algorithm which simulates an $n\lg\lg n$-processor EREW PRAM on an n-processor OCPC with $O(\lg\lg n)$ expected delay. The probability that the delay is longer than this is at most $n^{-\alpha}$ for any constant α.

It would be interesting to determine whether this is the fastest possible work-optimal simulation. It would also be interesting to discover how much delay is required in order to simulate a CRCW PRAM. We have recently derived an algorithm that simulates an n-processor CRCW PRAM step on an n-processor OCPC in time $O(\lg k + \lg\lg n)$ with high probability, where k is the maximum memory contention of the CRCW step.

The simulation algorithm assumes that k is known. This assumption can be removed by augmenting the OCPC model to include a single bus which can be used to synchronize all of the processors: each processor can broadcast a 1 bit, and every processor can determine whether or not any processor is broadcasting a 1 at any given time.

We note that the $\lg k$ term in the simulation algorithm is provably necessary, as implied by an $\Omega(\lg k)$ expected time lower bound for broadcasting the value of a bit

to k processors on a QRCW PRAM (and hence on an ERCW) by Gibbons, Matias, and Ramachandran (see [14]).

Evidently, the performance of the CRCW simulation depends on the maximum contention. A model that accounts for memory contention was recently proposed in [13]. In this model the run time of each step is a function of the memory contention encountered at this step. Thus, in the submodel of SIMD-QRQW log PRAM, a step in which the maximum memory contention is k is assumed to take $\lg k$ time units.

The CRCW simulation implies that an n-processor SIMD-QRQW log PRAM algorithm can be simulated on an n-processor OCPC, augmented with a bus, with delay $O(\lg \lg n)$ with high probability. We note that the SIMD-QRQW log PRAM is strictly stronger than the EREW PRAM.

REFERENCES

[1] H. Alt, T. Hagerup, K. Mehlhorn, and F.P. Preparata, *Deterministic simulation of idealized parallel computers on more realistic ones*, SIAM J. Comput., 16 (1987), pp. 808–835.

[2] R.J. Anderson and G.L. Miller, *Optical Communication for Pointer Based Algorithms*, Technical report CRI 88-14, Computer Science Department, University of Southern California, Los Angeles, CA, 1988.

[3] B. Bollobás, *Martingales, isoperimetric inequalities and random graphs*, in Combinatorics, Colloq. Math. Soc. János Bolyai 52, A. Hajnal, L. Lovász, and V. T. Sós, eds., North Holland, Amsterdam, 1988, pp. 113–139.

[4] J.L. Carter and M.N. Wegman, *Universal classes of hash functions*, J. Comput. Systems Sci., 18 (1979), pp. 143–154.

[5] B.S. Chlebus, K. Diks, T. Hagerup, and T. Radzik, *New simulations between CRCW PRAMs*, in Proc. 7th Foundations of Computation Theory, Lecture Notes in Comput. Sci. 380, Springer-Verlag, New York, 1989, pp. 95–104.

[6] M. Dietzfelbinger and F. Meyer auf der Heide, *How to distribute a dictionary in a complete network*, in Proc. 22nd ACM Symposium on Theory of Computing, 1990, pp. 117–127.

[7] M. Dietzfelbinger and F. Meyer auf der Heide, *Simple, efficient shared memory simulations*, in Proc. 5th ACM Symposium on Parallel Algorithms and Architectures, 1993, pp. 110–119.

[8] M.M. Eshaghian, *Parallel Computing with Optical Interconnects*, Ph.D. thesis, University of Southern California, Los Angeles, CA, 1988.

[9] M.M. Eshaghian, *Parallel algorithms for image processing on OMC*, IEEE Trans. Comput., 40 (1991), pp. 827–833.

[10] M.M. Eshaghian and V.K.P. Kumar, *Optical arrays for parallel processing*, in Proc. 2nd Annual Parallel Processing Symposium, 1988, pp. 58–71.

[11] A.V. Gerbessiotis and L.G. Valiant, *Direct bulk-synchronous parallel algorithms*, in Proc. 3rd Scandinavian Workshop on Algorithm Theory, 1992.

[12] M. Geréb-Graus and T. Tsantilas, *Efficient optical communication in parallel computers*, in Proc. 4th ACM Symposium on Parallel Algorithms and Architectures, 1992, pp. 41–48.

[13] P.B. Gibbons, Y. Matias, and V.L. Ramachandran, *The QRQW PRAM: Accounting for contention in parallel algorithms*, in Proc. 5th ACM–SIAM Symposium on Discrete Algorithms, 1994, pp. 638–648.

[14] P.B. Gibbons, Y. Matias, and V.L. Ramachandran, *Efficient low-contention parallel algorithms*, in Proc. 6th ACM Symposium on Parallel Algorithms and Architectures, 1994, pp. 236–247.

[15] J. Gil and Y. Matias, *Fast hashing on a PRAM—Designing by expectation*, in Proc. 2nd ACM–SIAM Symposium on Discrete Algorithms, 1991, pp. 271–280.

[16] J. Gil, Y. Matias, and U. Vishkin, *Towards a theory of nearly constant time parallel algorithms*, in Proc. 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 698–710.

[17] L.A. Goldberg, M. Jerrum, T. Leighton, and S. Rao, *Doubly logarithmic communication algorithms for optical communication parallel computers*, SIAM J. Comput., 26 (1997), pp. 1100–1119.

[18] L.A. Goldberg, M. Jerrum, and P.D. MacKenzie, *An $\Omega(\sqrt{\log \log n})$ lower bound for routing*

in optical networks, SIAM J. Comput., 27 (1998), pp. 1083–1098.
[19] J. JÁJÁ, *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, MA, 1992.
[20] R.M. KARP, M. LUBY, AND F. MEYER AUF DER HEIDE, *Efficient PRAM simulation on a distributed memory machine*, preprint, 1994. (A preliminary version of this paper appeared in Proc. 24th ACM Symposium on Theory of Computing, 1992, pp. 318–326.)
[21] A.R. KARLIN AND E. UPFAL, *Parallel hashing—An efficient implementation of shared memory*, in Proc. 18th ACM Symposium on Theory of Computing, 1986, pp. 160–168.
[22] R.M. KARP AND V. RAMACHANDRAN, *Parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 869–941.
[23] C.P. KRUSKAL, L. RUDOLPH, AND M. SNIR, *A complexity theory of efficient parallel algorithms*, Theoret. Comput. Sci., 71 (1990), pp. 95–132.
[24] F.T. LEIGHTON, *Methods for message routing in parallel machines*, in Proc. 24th ACM Symposium on Theory of Computing, 1992, pp. 77–96.
[25] Y. MATIAS, *Highly Parallel Randomized Algorithmics*, Ph.D. thesis, Tel Aviv University, Tel Aviv, Israel, 1992.
[26] Y. MATIAS AND U. VISHKIN, *Converting high probability into nearly-constant time—With applications to parallel hashing*, in Proc. 23rd ACM Symposium on Theory of Computing, 1991, pp. 307–316.
[27] F. MEYER AUF DER HEIDE, C. SCHEIDELER, AND V. STEMANN, *Exploiting storage redundancy to speed up randomized shared memory simulations*, in Proc. 12th Symposium on Theoretical Aspects of Computer Science (STACS), 1995, pp. 267–278.
[28] P.D. MACKENZIE, C.G. PLAXTON, AND R. RAJARAMAN, *On contention resolution protocols and associated probabilistic phenomena*, in Proc. 26th ACM Symposium on Theory of Computing, 1994, pp. 153–162.
[29] C. MCDIARMID, *On the method of bounded differences*, in Surveys in Combinatorics, London Math. Soc. Lecture Note Ser. 141, Cambridge University Press, Cambridge, UK, 1989, pp. 148–188.
[30] W.F. MCCOLL, *General purpose parallel computing,* in Lectures on Parallel Computation, Proc. 1991 ALCOM Spring School on Parallel Computation, A.M. Gibbons and P. Spirakis, eds., Cambridge University Press, Cambridge, UK, 1993, pp. 337–391.
[31] K. MEHLHORN AND U. VISHKIN, *Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories*, Acta Inform., 21 (1984), pp. 339–374.
[32] A.G. RANADE, *How to emulate shared memory*, J. Comput. Systems Sci., 42 (1991), pp. 307–326.
[33] S.B. RAO, *Properties of an Interconnection Architecture Based on Wavelength Division Multiplexing*, Technical report TR-92-009-3-0054-2, NEC Research Institute, Princeton, NJ, 1992.
[34] J.H. REIF, ED., *A Synthesis of Parallel Algorithms*, Morgan-Kaufmann, San Francisco, CA, 1993.
[35] A. SIEGEL, *On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications*, in Proc. 30th IEEE Symposium on Foundations of Computer Science, 1989, pp. 20–25.
[36] E. UPFAL, *Efficient schemes for parallel communications*, J. Assoc. Comput. Mach., 31 (1984), pp. 507–517.
[37] E. UPFAL, *A probabilistic relation between desirable and feasible models of parallel computation*, in Proc. 16th ACM Symposium on Theory of Computing, 1984, pp. 258–265.
[38] E. UPFAL AND A. WIGDERSON, *How to share memory in a distributed system*, J. Assoc. Comput. Mach., 34 (1987), pp. 116–127.
[39] L.G. VALIANT, *General purpose parallel architectures*, Handbook of Theoretical Computer Science, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, Chapter 18.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1848–1874

TIME-LAPSE SNAPSHOTS*

CYNTHIA DWORK†, MAURICE HERLIHY‡, SERGE PLOTKIN§, AND ORLI WAARTS¶

Abstract. A snapshot scan algorithm produces an "instantaneous" picture of a region of shared memory that may be updated by concurrent processes. Many complex shared memory algorithms can be greatly simplified by structuring them around the snapshot scan abstraction. Unfortunately, the substantial decrease in conceptual complexity quite often is counterbalanced by an increase in computational complexity.

In this paper, we introduce the notion of a *weak snapshot scan*, a slightly weaker primitive that has a more efficient implementation. We propose the following methodology for using this abstraction: first, design and verify an algorithm using the more powerful snapshot scan; second, replace the more powerful but less efficient snapshot with the weaker but more efficient snapshot, and show that the weaker abstraction nevertheless suffices to ensure the correctness of the enclosing algorithm.

We give two examples of algorithms whose performance is enhanced while retaining a simple modular structure: bounded concurrent timestamping and bounded randomized consensus. The resulting timestamping protocol dominates all other currently known timestamping protocols: it matches the speed of the fastest known bounded concurrent timestamping protocol while actually reducing the register size by a logarithmic factor. The resulting randomized consensus protocol matches the computational complexity of the best known protocol that uses only bounded values.

Key words. distributed computing, distributed algorithms, shared-memory algorithms, asynchronous PRAMS, atomic snapshots, timestamping, time-lapse snapshots

AMS subject classifications. 68Q22, 68Q25

PII. S0097539793243685

1. Introduction.

Synchronization algorithms for shared-memory multiprocessors are notoriously difficult to understand and to prove correct. Recently, however, researchers have identified several powerful abstractions that greatly simplify the conceptual complexity of many shared-memory algorithms. One of the most powerful of these is *atomic snapshot scan* (in this paper we sometimes omit the word "scan"). Informally, this is a procedure that makes an "instantaneous" copy of memory that is being updated by concurrent processes. More precisely, the problem is defined as follows. A set of n asynchronous processes share an n-element array A, where only process P writes $A[P]$.[1] An atomic snapshot is a read of all the elements in the array that appears to occur instantaneously. Formally, scans and updates are required to be *linearizable* [25], i.e., each operation appears to take effect instantaneously at some

*Received by the editors January 3, 1993; accepted for publication (in revised form) February 2, 1998; published electronically May 21, 1999.

http://www.siam.org/journals/sicomp/28-5/24368.html

†IBM Almaden Research Center, San Jose, CA 95120 (dwork@almaden.ibm.com).

‡Computer Science Department, Brown University, Providence, RI 02912 (herlihy@cs.brown.edu). Part of the research of this author was done while at the DEC Cambridge Research Lab.

§Department of Computer Science, Stanford University, Stanford, CA 94305 (plotkin@theory.stanford.edu). The research of this author was supported by NSF Research Initiation Award CCR-900-8226, U.S. Army Research Office grant DAAL-03-91-G-0102, ONR contract N00014–88–K–0166, and a grant from Mitsubishi Electric Laboratories.

¶Computer Science Division, University of California, Berkeley, CA 94720 (waarts@cs.berkeley.edu). Part of the research of this author was done at Stanford University and was supported in part by NSF grant CCR-8814921, U.S. Army Research Office grant DAAL-03-91-G-0102, ONR contract N00014-88-K-0166, and an IBM fellowship.

[1] One can also define multiwriter algorithms in which any process can write to any location.

point between the operation's invocation and response. Thus, there exists a *total* "linearization" order on the scans and updates, such that a scan operation returns for each process P the value of the last update operation to $A[P]$, where *last* is with respect to the linearization order. Note that this implies that if processes p and q update their values from u_p, u_q to v_p, v_q, respectively, then if some atomic snapshot scan returns values v_p, u_q, no other atomic snapshot scan can return values u_p, v_q.

Prior to our work, atomic snapshot scan algorithms had been constructed by Anderson [3] (bounded registers and exponential running time), Aspnes and Herlihy [6] (unbounded registers and $O(n^2)$ running time), and Afek et al. [2] (bounded registers and $O(n^2)$ running time). More recent results, derived after the time this work was first published and requiring only bounded registers, include randomized algorithms of Attiya, Herlihy, and Rachman [10] ($O(n \log^2 n)$ time) and Chandra and Dwork [14] ($O(n) + C_n$, where C_n is the running time of the best randomized consensus algorithm for n processes) and the fastest deterministic algorithm to date, by Attiya and Rachman [11] ($O(n \log n)$ steps per operation). Here running time is measured by the number of accesses to shared memory. Chandy and Lamport [15] considered a closely related problem in the message-passing model.

Unfortunately, the substantial decrease in conceptual complexity provided by atomic snapshot scan is often counterbalanced by an increase in computational complexity. In this paper, we introduce the notion of a *weak snapshot scan*, called a *time-lapse snapshot*. The time-lapse snapshot is a slightly weaker abstraction than the atomic snapshot scan in which the linearization order is allowed to be partial, rather than total. This weaker notion of correctness allows two scans to disagree on the order of two updates, but only if all four operations are concurrent with one another. That is, if processes p and q update their values from u_p, u_q to v_p, v_q, respectively, and if scans S, S' are done concurrently with each other and with these updates, it is possible that scan S returns values v_p, u_q and scan S' returns u_p, v_q. The advantage of using weak snapshot is that it can be implemented in $O(n)$ time. Thus, the cost of our time-lapse snapshot is asymptotically the same as the cost of a simple "collect" of the n values, but the primitive is much more powerful. Letting v be the maximum number of bits in any element in the array A, our implementation of the time-lapse snapshot requires registers of size only $O(n + v)$. In contrast, all atomic snapshot algorithms known to us require registers of size $O(nv)$. Since in some applications v can be as large as n, this reduction in register size can be significant.

Our results indicate that weak snapshot scan can sometimes alleviate the tradeoff between conceptual and computational complexity. We focus on two well-studied problems: bounded concurrent timestamping and randomized consensus. In particular, we consider algorithms for these problems based on an atomic snapshot. In both cases, we show that one can simply replace the atomic snapshot scan with a weak snapshot scan, thus retaining the algorithms' structure while improving their performance.

The weak snapshot algorithm presented here was influenced by work of Kirousis, Spirakis, and Tsigas [30], who designed a linear-time atomic snapshot algorithm for a *single* scanner. In this special case our algorithm solves the original atomic snapshot problem as well.

In the bounded *concurrent timestamping* problem processes repeatedly choose labels, or timestamps, reflecting the real-time order of events. More specifically, processes can repeatedly perform two types of operations, called **labeling** and **scan**. In a **labeling** operation a process assigns itself a new label. A **scan** operation returns

a set of current labels, one per process, and a total order on these labels, consistent with the real-time order of their corresponding **labeling** operations.[2]

Israeli and Li [26] were the first to investigate bounded *sequential* timestamp systems; Dolev and Shavit [19] were the first to explore the *concurrent* version of the problem. The Dolev–Shavit construction requires $O(n)$-sized registers and labels, $O(n)$ time for a **labeling** operation, and $O(n^2 \log n)$ time for a **scan**. In their algorithm each process is assigned a single multireader–single-writer register of $O(n)$ bits. Extending the Dolev–Shavit solution in a nontrivial way, Israeli and Pinhasov [28] obtained a bounded concurrent timestamp system that is linear in time and label size but uses registers of size $O(n^2)$. An alternative implementation of their algorithm uses *single*-reader–single-writer registers[3] of size $O(n)$ but requires $O(n^2)$ time to perform a **scan**. Independently, and slightly later, Dwork and Waarts [20] obtained the first linear time solution, using a different approach not based on any of the previous solutions, with a simpler proof of correctness. The drawback of their construction is that it requires registers and labels of size $O(n \log n)$. After this work was first published, the Dwork–Waarts approach was pursued by Haldar [23], who presented a timestamping system that is slightly more memory efficient than the construction of [20] (although not asymptotically) but that also requires registers and labels of size $O(n \log n)$.

Dolev and Shavit observed that the *conceptual* complexity of their concurrent timestamping algorithm can be reduced by using atomic snapshot scan. We show that, in addition, the algorithm's *computational* complexity can be reduced by simply replacing the atomic snapshot with the weak snapshot, making no other changes to the original algorithm. The resulting bounded concurrent timestamping algorithm is linear in both time and the size of registers and labels and is conceptually simpler than the Dolev–Shavit and Israeli–Pinhasov solutions. We do not need to prove directly that our timestamping is correct; rather, we reduce the correctness of our solution to the correctness of the Dolev–Shavit algorithm.

Independent of our work, Gawlick, Lynch, and Shavit [22] actually structured the Dolev–Shavit algorithm around the atomic snapshot abstraction. The resulting algorithm is indeed significantly more simple and modular than that of Dolev and Shavit. Introducing new proof techniques, Gawlick, Lynch, and Shavit prove from scratch that their algorithm is a bounded concurrent timestamping system. At the time of the Gawlick, Lynch, and Shavit work, the (then) best atomic snapshot algorithm was the one in [2]. Using this in the construction in [22] effectively slowed down the original Dolev–Shavit timestamping algorithm, yielding a timestamping algorithm that requires $O(n^2)$ accesses to registers of size $O(n^2)$ for both **labeling** and **scan** operations. More than a year after initial publication of our results, Attiya and Rachman developed the $O(n \log n)$ atomic snapshot [11] mentioned above. Plugging this atomic snapshot algorithm into the timestamping construction of Gawlick, Lynch, and Shavit yields a timestamping algorithm that requires $O(n \log n)$ accesses to registers of size $O(n^2)$. In comparison, our construction requires only $O(n)$ accesses to registers of size only $O(n)$. Thus, our timestamping construction dominates all others of which we are aware.

In the *randomized consensus* problem, each of n asynchronous processes starts with an input value taken from a two-element set and runs until it chooses a *de-*

[2] Observe that the scan required by the timestamping is not necessarily identical to the atomic snapshot scan. Unfortunately, the two operations have the same name in the literature.

[3] All other results mentioned are in terms of multireader–single-writer registers.

cision value and halts. The protocol must be *consistent*—no two processes choose different decision values; *valid*—the decision value is the input value of some process; and randomized *wait-free*—each process decides after an expected finite number of steps. The consensus problem lies at the heart of the more general problem of constructing highly concurrent data structures [24]. Consensus has no deterministic solution in the asynchronous shared-memory model with only atomic reads and writes [18, 36]. Nevertheless, it can be solved by *randomized* protocols in which each process is guaranteed to decide after a finite *expected* number of steps. Randomized consensus protocols that use unbounded registers have been obtained by Chor, Israeli, and Li [17] (against a "weak" adversary), by Abrahamson [1] (exponential running time), by Aspnes and Herlihy [7] (the first polynomial algorithm), by Saks, Shavit, and Woll [40] (optimized for the case where processes run in lock step), and by Bracha and Rachman [12] (running time $O(n^2 \log n)$). After the time of our work, Aspnes and Waarts [8] presented a randomized consensus protocol that uses unbounded registers and in which the expected number of operations per process is $O(n \log^2 n)$.

Protocols that use bounded registers have been proposed by Attiya, Dolev, and Shavit [9] (running time $O(n^3)$), by Aspnes [5] (running time $O(n^2(p^2+n))$, where p is the number of active processors), and by Bracha and Rachman [13] (running time $O(n(p^2+n))$, the best known). The bottleneck in Aspnes's algorithm is atomic snapshot. Replacing this atomic snapshot with our more efficient weak snapshot improves the running time by $\Omega(n)$ (from $O(n^2(p^2+n))$ to $O(n(p^2+n))$) and yields a protocol that matches the protocol of Bracha and Rachman [13]. Both our consensus algorithm and the one in [13] are based on Aspnes's algorithm. The crucial difference is that the solution of Bracha and Rachman is specific to consensus, whereas our algorithm is an immediate application of the primitive developed in this paper.

The remainder of this paper is organized as follows. Section 2 gives our model of computation and defines the weak snapshot primitive. Some properties of weak snapshots appear in section 3. The remaining sections describe the weak snapshot algorithm and its applications.

2. Model and definitions. A *concurrent system* consists of a collection of n asynchronous *processes* that communicate through an initialized shared memory. Each memory location, called a *register*, can be written by one "owner" process and read by any process. Reads and writes to shared registers are assumed to be *atomic*, that is, they can be viewed as occurring at a single instant of time. In order to be consistent with the literature on the discussed problems, our time complexity measure is expressed in terms of read and write operations on single-writer–multireader registers, in our case of size $O(n)$. Polynomial-time algorithms for implementing large single-writer–multireader atomic registers from small, weaker registers are well known [31, 32, 38, 27, 29, 33, 34, 35, 37, 42, 44].

We view an execution of a protocol as an interleaving of atomic reads and writes. Sometimes it is convenient to assign *times* to these atomic operations. These times are not accessible to the processes. A register can undergo at most one atomic operation at a time, and a process can perform at most one atomic operation at a time.

An algorithm is *wait-free* if there is an a priori upper bound on the number of steps a process might take when running the algorithm, regardless of how its steps are interleaved with those of other processes. All algorithms discussed in this paper are wait-free.

An *atomic snapshot memory* supports two kinds of abstract operations: **update** modifies a location in the shared array, and **scan** instantaneously reads (makes a copy

of) the entire array. Let U_i^k (S_i^k) denote the kth **update** (**scan**) of process i, and v_i^k the value written by i during U_i^k. The superscripts are omitted where this cannot cause confusion. An operation A *precedes* operation B, written as $A \longrightarrow B$, if B starts after A finishes. Operations unrelated by precedence are *concurrent*. Processes are *sequential*: each process starts a new operation only when its previous operation has finished, hence its operations are totally ordered by precedence.

Correctness of an atomic snapshot memory is defined as follows. There exists a total order "$\Longrightarrow$" on operations such that

- if $A \longrightarrow B$, then $A \Longrightarrow B$;
- if **scan** S_p returns $\bar{v} = \langle v_1, \ldots, v_n \rangle$, then v_q is the value written by the latest **update** U_q ordered before S_p by $\Longrightarrow$.

The order $\Longrightarrow$ is called the *linearization* order [25]. Intuitively, the first condition says that the linearization order respects the "real-time" precedence order, and the second says that each correct concurrent computation is equivalent to some sequential computation where the **scan** returns the last value written by each process.

We define a *time-lapse* (or *weak*) *snapshot* memory as follows: we impose the same two conditions, but we allow $\Longrightarrow$ to be a *partial order*[4] rather than a total order. We call this order a *partial linearization order*. Since for each q all the updates U_q are linearized by the $\longrightarrow$ relation, the "latest **update** U_q ordered before S_p" is well defined, even though $\Longrightarrow$ is only a partial order. If $A \Longrightarrow B$ we say that B *observes* A.

Each **scan** and **update** operation takes place during the interval of time beginning with the first atomic action of the operation and ending with the last atomic action. Intuitively, we require that scanning processes agree about **update**s that happened (in real time) before the **scan**s began, but they may disagree about concurrent **update**s. That is, if processes p and q update their values from u_p, u_q to v_p, v_q, respectively, and if **scan**s S, S' start after the two **update**s terminate, then the values each with returns for p, q are at least as recent as v_p, v_q, respectively; however, if **scan**s S, S' are done concurrently with each other and with these updates, then it is possible that **scan** S returns values v_p, u_q and **scan** S' returns u_p, v_q. Thus, in a system with only one scanner, atomic snapshots and weak snapshots are equivalent. Similarly, the two types of snapshots are equivalent if no two **update**s occur concurrently (overlap in real time). However, in general, our weaker notion of correctness allows two **scan**s S and S' to disagree on the order of two **update**s U and U', but only if all four operations are concurrent with one another.

The weak snapshot is still more powerful than a simple **collect** of values written by atomic writes. To see this, consider three processes a, b, c. Let a perform an **update** U_a, writing the value v_a. Let b then perform a **scan** S_b, followed by an **update** U_b resulting in v_b. Assume that process c performs a **scan** S_c concurrent with all three of these operations and that returns v_b for process b. If the **update**s were simple writes and the **scan** a simple collect, then the fact that S_c returns v_b places no constraint upon whether S_c returns v_a or an earlier value for a. However, if the operations are instead operations of the weak snapshot primitive, then we have $U_a \Longrightarrow S_b$ (because S_b returns v_a), $S_b \Longrightarrow U_b$ (because $S_b \longrightarrow U_b$), and $U_b \Longrightarrow S_c$ (because S_c returns v_b). By transitivity we get $U_a \Longrightarrow S_c$, whence it follows that S_c returns v_a or a later value for a.

The following assumption is convenient and easily implemented.

Assumption 2.1. If an **update** is observed by a **scan**, then this **update** termi-

[4] In this paper, all partial orders are irreflexive.

nates before the **scan** does.

Assumption 2.1 is useful in modifying the bounded concurrent timestamping algorithm of [19], since it allows us to reduce the correctness of the resulting system to the correctness of the original algorithm of [19], thereby avoiding a full proof of correctness from scratch. Intuitively, it allows us to argue that anything written by an **update** and seen by a **scan** in the modified algorithm could have been written by a simple atomic write in a *corresponding execution* of the original Dolev–Shavit algorithm and seen by a simple **collect** in that execution. We explain this further in section 5.

We close this section with a brief review of some properties of atomic registers. We use the notation W_x^i (respectively, R_x^i) to denote the ith atomic write (respectively, read) of the register by process x.

Regularity of an atomic register says that for any value v written by a write W_a^i and returned by a read R_b^k, W_a^i begins before R_b^k terminates, and there is no write W_a^j for $j > i$ such that $W_a^i \longrightarrow W_a^j \longrightarrow R_b$.

Monotonicity of an atomic register says that if $R_b \longrightarrow R_c$ and if R_b returns v written by a write W_a^i, then R_c returns v' written by a write $W_a^{i'}$ for some $i' \geq i$.

3. Properties of weak snapshots. The reader can easily verify that weak snapshots satisfy the following properties:

- *Regularity.* For any value v_a^i returned by S_b^j, U_a^i begins before S_b^j terminates, and there is no U_a^k such that $U_a^i \longrightarrow U_a^k \longrightarrow S_b^j$.
- *Monotonicity of* **scan**s. If S_a^i and S_b^j are two **scan**s satisfying $S_a^i \longrightarrow S_b^j$ (a and b could be the same process), and if S_a^i observes **update** U_c^k (formally, $U_c^k \Longrightarrow S_a^i$), then S_b^j observes U_c^k.
- *Monotonicity of* **update**s. If U_a^i and U_b^j are two **update** operations (possibly by the same process) such that $U_a^i \longrightarrow U_b^j$, and if S_c^k is a **scan** operation, possibly concurrent with both U_a^i and U_b^j, such that S_c^k observes U_b^j ($U_b^j \Longrightarrow S_c^k$), then S_c^k observes U_a^i.

Variants of the regularity and monotonicity properties are part of the definitions of several basic shared-memory abstractions, including timestamping systems [26, 19, 22] and the traceable-use abstraction [20], and follow from the definitions of several other abstractions, including atomic registers [32] and atomic snapshots [2, 6].

Roughly speaking, time-lapse snapshots satisfy all the properties of atomic snapshots except for the consistency property which states that if **scan**s S_a^i, S_b^j return $\bar{v} = \langle v_1, \ldots, v_n \rangle$ and $\bar{v}' = \langle v'_1, \ldots, v'_n \rangle$, respectively, then either $U_k \not\longrightarrow U'_k$ for every $k = 1, \ldots, n$, or vice versa.

Define the *span* of a value v_p^i to be the interval from the start of U_p^i to the end of U_p^{i+1}, if U_p^{i+1} exists and terminates, or the infinite interval with the same start time if no U_p^{i+1} ever terminates. Clearly, values written by successive **update**s have overlapping spans. The following lemma formalizes the intuition that a weak snapshot **scan** returns a possibly existing state of the system.

LEMMA 3.1. *If a weak snapshot* **scan** *S returns a set of values $\bar{v}$, then their spans have a nonempty intersection.*

Proof. By definition of time-lapse snapshot, for all $v_p^i \in \bar{v}$, U_p^i is the latest **update** ordered before S by $\Longrightarrow$. Let v_p^i in $\bar{v}$ be such that the span of v_p^i is the latest to start. If all the spans of the values in $\bar{v}$ are infinite, then clearly they intersect at all times after U_p^i begins.

Assuming at least one span is finite, let v_p^i be as above and let v_q^j be in $\bar{v}$ such that the span of v_q^j is the first to end. Then, it is enough to show that the spans of v_p^i and v_q^j intersect. Suppose not. By choice of q, U_q^{j+1} completes. Since by assumption, the spans of v_p^i and v_q^j do not intersect, it follows from the definition of a span that $U_q^{j+1} \longrightarrow U_p^i$. Thus $U_q^{j+1} \Longrightarrow U_p^i$, and hence it follows from the transitivity of the partial linearization order that $U_q^{j+1} \Longrightarrow S$, violating the requirement that each **scan** return the latest value written by the latest **update** ordered before it by $\Longrightarrow$. □

Let a **scan** S of a weak snapshot start at time t_s, end at time t_e, and return a set of values $\bar{v}$. By Lemma 3.1, there is a point t in which the spans of all these values intersect. We now argue that at least one such point is in the interval $[t_s, t_e]$. There may be more than one such point; however, the regularity property of weak snapshots implies that there is at least one such point t such that $t_s \leq t \leq t_e$. This is because the first clause in the definition of regularity implies that the span of v_a^i begins before t_e, while the second clause implies that the span of v_a^i ends at or after t_s. We will refer to the latest such point t by t_{scan} of S.

4. Weak snapshot. Intuitively, in order to be able to impose a partial order on the **scan**s and **update**s, we need to ensure that if a **scan** S_c does not return value v_a^j of process a because v_a^j is too new (i.e., S_c returns v_a^k for $k < j$), then S_c will not return a value v_b^i that was written by process b *after* b returns v_a^j in some **scan** S_b.

This intuition follows from the definition of the time-lapse snapshot: if S_b returns v_a^j and $S_b \longrightarrow U_b^i$ we have $U_a^j \Longrightarrow S_b \Longrightarrow U_b^i$, by definition of time-lapse snapshot. Moreover, if S_c returns v_b^i, then S_c must be ordered after b's **update** in the partial order; that is, $U_b^i \Longrightarrow S_c$. We therefore have by transitivity that $U_a^j \Longrightarrow S_c$, or in other words, that a's **update** U_a^i is ordered before the **scan** S_c, and hence by definition of time-lapse snapshots S_c must return v_a^k for $k \geq j$.

If each value returned by the **scan** is the value written by the latest **update** that terminated before a specific point in the **scan**, the above situation does not occur. This observation by Kirousis, Spirakis, and Tsigas [30] motivates our solution. Roughly speaking, in our solution, at the start of a **scan**, the scanner produces a new number, called *color*, for each other process. When a process wants to perform an **update**, it reads the colors produced for it (one color by each scanner) and tags its new value with these colors. This enables the scanner to distinguish older values from newer ones.

The next section describes a solution that uses an unbounded number of colors. Later we will show how to simulate this solution using only a bounded number of colors. The simulation uses a simplification of the **Traceable Use** abstraction defined by Dwork and Waarts in [20].

4.1. Unbounded weak snapshot. We follow the convention that shared registers appear in upper case and private variables in lower case. In order to simplify the presentation, we assume that all the private variables are persistent. If a variable is subscripted, the first subscript indicates the unique process that writes it, and the second, if present, indicates the process that uses it. When the code says "atomically write ... for all ...," we mean a single atomic write operation in which all indicated variables are written at once. In contrast, a statement of the form "for each $x \in X$ write ..." means that the executing process must perform $|X|$ atomic writes. Each process b has variables VALUE_b, which stores b's current value; $\text{PCOLOR}_b, \text{QCOLOR}_b$, each of which stores an n-element array of colors; and VASIDE_{bc}, for each $c \neq b$. In keeping with convention [2, 20, 32], we frequently refer to $\text{PCOLOR}_b[c]$ as PCOLOR_{bc}

1. For all $c \neq b$, read $qcolor_b[c] :=$ PCOLOR$_{cb}$.
2. For all $c \neq b$, if $qcolor_b[c] \neq$ QCOLOR$_{bc}$
   then $vaside_b[c] :=$ VALUE$_b$.
3. Atomically write:
   VALUE$_b :=$ new value.
   For all $c \neq b$, VASIDE$_{bc} := vaside_b[c]$.
   For all $c \neq b$, QCOLOR$_{bc} := qcolor_b[c]$.

FIG. 4.1. **update** *operation for process b.*

1. Call **Produce**.
2. For all $c \neq b$ atomically read:
   $value_b[c] :=$ VALUE$_c$,
   $qcolor_b[c] :=$ QCOLOR$_{cb}$,
   $vaside_b[c] :=$ VASIDE$_{cb}$.
3. For all $c \neq b$
   If $qcolor_b[c] \neq pcolor_b[c]$
      then $data_b[c] := value_b[c]$,
      else $data_b[c] := vaside_b[c]$.
4. Return $(data_b[1], \ldots,$ VALUE$_b, \ldots, data_b[n])$.

FIG. 4.2. **scan** *operation for process b.*

(analogously for QCOLOR$_b[c]$). In this section, we assume that all these variables are stored in a single register. Section 4.4 describes how to eliminate this assumption. The code for the **update** and **scan** operations appears in Figures 4.1 and 4.2, respectively; the code for the **Produce** operation, called by the **scan**, appears in Figure 4.3.

At the start of a **scan**, the scanner b produces a new color for each updater c and stores it in PCOLOR$_{bc}$. (Colors produced *by* different processes or colors produced *for* different processes are considered different even if they have the same value.) It then reads VALUE$_c$, VASIDE$_{cb}$, and QCOLOR$_{cb}$ in a single atomic step. If QCOLOR$_{cb}$ is equal to the color produced by b for c (and stored in PCOLOR$_{bc}$), then b returns VASIDE$_{cb}$ as the value for c; otherwise b returns VALUE$_c$.

The updater b first reads PCOLOR$_{cb}$ and then writes its new VALUE$_b$ atomically with QCOLOR$_{bc} :=$ PCOLOR$_{cb}$ for all c. At the same time b updates VASIDE$_{bc}$ for all c that it detects as having started to execute a concurrent **scan**.

The intuition behind the use of the VASIDE variable can be best described if we consider an example where we have a "fast" updater b and a "slow" scanner c, where c executes a single **scan** while b executes many **update**s. In this case, the updater will update VALUE$_b$ each time but will update VASIDE$_{bc}$ only once, when it will detect that c is scanning concurrently. Intuitively, VASIDE$_{bc}$ allows the scanner c to return a value for process b that was written by b during an **update** that started no later than the end of the color producing step of the current **scan**. Therefore, such a value can depend only on values that are not more recent than the values returned by the **scan**.

We superscript the values of variables to indicate the execution of **update** or **scan** in which they are written. For example PCOLOR$^{\ell}_{bc}$ is the value of PCOLOR$_{bc}$ written during **scan** S^{ℓ}_b.

Next, we construct an explicit partial linearization order $\Longrightarrow$ as follows. Define $U^j_q \Rightarrow S^i_p$ to hold if S^i_p returns the value originally written by U^j_q. (Note that S^i_p may read this value from VASIDE$^k_{qp}$, where $k > j$). Define $\Longrightarrow$ to be the transitive closure of $\longrightarrow \cup \Rightarrow$. It follows from Lemmas 4.1 and 4.4 that the **scan** and **update** procedures

1. For all $c \neq b$ $pcolor_b[c] := \text{PCOLOR}_{bc} + 1$.
2. Atomically write for all $c \neq b$ $\text{PCOLOR}_{bc} := pcolor_b[c]$.

FIG. 4.3. *Unbounded* **Produce** *operation for process b.*

yield a weak snapshot memory.

LEMMA 4.1. *The relation* $\Longrightarrow$ *is a partial order.*

Proof. It suffices to check that $\Longrightarrow$ is acyclic. Suppose there exists a cycle $A_0, \ldots, A_k$, where adjacent operations are related by $\longrightarrow$ or $\Rightarrow$, and the cycle length is minimal. Because $\longrightarrow$ is acyclic and transitive, some of these operations must be related only by $\Rightarrow$. Since $\Rightarrow$ goes only from **update** to **scan** operations, there are no adjacent $\Rightarrow$ edges. Thus, if the cycle is of length 1, then we must have $A_0 \longrightarrow A_0$, which is a contradiction. Hence the cycle must be of length > 1. Since the cycle is minimal, and $\longrightarrow$ is transitive, there are no adjacent $\longrightarrow$ edges and therefore for each consecutive pair A_i and A_{i+1}, if $A_i \Rightarrow A_{i+1}$, then $A_i \not\longrightarrow A_{i+1}$. Therefore the edges of the cycle must alternate between $\Rightarrow$ and $\longrightarrow$. It follows that k is odd. Without loss of generality, assume $A_0 \Rightarrow A_1$.

We argue by induction that, for $\ell \geq 0$, we have that A_0 starts before $A_{2\ell+2}$ starts. Throughout the proof all subscripts are taken modulo $k+1$. (In particular, when $k = 1$ we have that A_2 is precisely A_0.) For the base case ($\ell = 0$), since by assumption $A_0 \Rightarrow A_1$, it follows that A_0 starts before A_1 finishes (otherwise A_1 would be returning a value written by an operation that had not even started and therefore might never start, contradicting the regularity of atomic registers). Since by construction $A_1 \longrightarrow A_2$ (alternating edges property), we have that A_1 finishes before A_2 starts, and the base case follows.

Assume the result for ℓ. We have $A_{2\ell+2} \Rightarrow A_{2\ell+3}$ (alternating edges), and hence again $A_{2\ell+2}$ starts before $A_{2\ell+3}$ finishes. By the inductive hypothesis, A_0 starts before $A_{2\ell+2}$ starts and hence A_0 starts before $A_{2\ell+3}$ finishes. To finish the argument, note that $A_{2\ell+3} \longrightarrow A_{2\ell+4}$ (alternating edges), which implies that A_0 starts before $A_{2\ell+4}$ starts, completing the induction.

Recall that the cycle has even length and that this length is at least 2. Thus, A_0 starts before A_{k+1} starts, but since all subscripts are modulo $k+1$ this says that A_0 starts before itself, which is a contradiction. ∎

To complete the proof that our implementation is a weak snapshot, we need only to show that for each process our weak **scan** returns the value written by the latest **update** ordered before that **scan**. We first show the two technical lemmas below.

LEMMA 4.2. *If a* **scan** *returns a value written by an* **update**, *then this* **update** *terminates before the* **scan** *reads* VALUE_q.

Proof. Recall that v_q^j denotes the value originally written by U_q^j. Assume **scan** S_p^i returns v_q^j. Recall that by design of the algorithm, there are two scenarios in which S_p^i returns v_q^j: either (a) in step 3 of S_p^i $qcolor_p^i[q] \neq pcolor_p^i[q]$ and v_q^j is the value p read from VALUE_q in step 2, or (b) $qcolor_p^i[q] = pcolor_p^i[q]$ and v_q^j is the value in VASIDE_{qp} that p read in step 2. Recall that v_q^j can be written into VASIDE_{qp} only by $U_q^{j'}$, $j' > j$ (by inspection of the code for an **update**).

Case (a) can occur only if q wrote v_q^j to VALUE_q before S_p^i reads that variable. Since (by inspection of the code) that write is the last atomic action in U_q^j, the claim follows. Case (b) can occur only if q wrote v_q^j to VASIDE_{qp} before S_p^i reads that variable. Since such a write must take place during $U_q^{j'}$, $j' > j$, it follows that the write must be after U_q^j terminates, and again the claim follows. ∎

LEMMA 4.3. *Let U_q^j be the last* **update** *by process q to be ordered before S_p^i by $\Longrightarrow$. U_q^j terminates before S_p^i reads VALUE_q. Moreover, in step 1 of U_q^j, q reads $\text{PCOLOR}_{pq}^{i'}$ for some $i' < i$.*

Proof. By definition of $\Longrightarrow$ there must be a sequence $A_0, \ldots, A_k$ where adjacent operations are related by $\longrightarrow$ or $\Rightarrow$ and where $A_0 = U_q^j$ and $A_k = S_p^i$. The proof proceeds by induction on the length k of a minimal such sequence.

For the base case, $k = 1$, observe that either $U_q^j \longrightarrow S_p^i$ or $U_q^j \Rightarrow S_p^i$. If $U_q^j \longrightarrow S_p^i$, then the first part of the claim is immediate, and the second part of the claim follows from the regularity of atomic registers.

We now consider the case in which $U_q^j \Rightarrow S_p^i$. By definition v_q^j is thus the value returned for q by S_p^i. Lemma 4.2 thus implies the first part of the claim.

To complete the proof of the base case, we will show by contradiction that U_q^j does not read PCOLOR_{pq}^i but instead reads $\text{PCOLOR}_{pq}^{i'}$ for some $i' < i$.

First, suppose for the sake of contradiction that U_q^j reads $\text{PCOLOR}_{pq}^{i'}$ for some $i' > i$ in step 1 of the **update**. Then U_q^j writes v_q^j only after S_p^i ends, so by the regularity of atomic registers S_p^i cannot return a value written by U_q^j; that is, $U_q^j \not\Rightarrow S_p^i$. It remains only to show that U_q^j does not read PCOLOR_{pq}^i.

Suppose otherwise; that is, suppose that, in step 1 of U_q^j, q reads PCOLOR_{pq}^i and thus $\text{QCOLOR}_{qp}^j = \text{PCOLOR}_{pq}^i = pcolor_p^i[q]$. It follows from step 3 of the **scan** operation that S_p^i could not have taken v_q^j from VALUE_q because $\text{QCOLOR}_{qp}^j = pcolor_p^i[q]$. So S_p^i must have taken v_q^j from VASIDE_{qp}. This implies that there is some $U_q^{j'}, j' > j$, that wrote v_q^j into VASIDE_{qp} and that terminated before S_p^i performed step 2. We show that there is no such $U_q^{j'}$. Since by assumption U_q^j reads PCOLOR_{pq}^i, the monotonicity of an atomic register implies that so does any later $U_q^{j'}$ that terminates before S_p^i's read from q in step 2, and hence it follows from the code of the **update** operation that any such later $U_q^{j'}$ sees $qcolor_q[p] = \text{QCOLOR}_{qp}$ and hence does not write v_q^j into VASIDE_{qp}. This completes the proof of the base case.

Assume the claim for k and suppose the minimal sequence from U_q^j to S_p^i is of length $k+1$. Then such a sequence has one of the following forms:

1. $U_q^j \longrightarrow U_z^l \Longrightarrow S_p^i$.
2. $U_q^j \longrightarrow S_g^m \Longrightarrow S_p^i$.
3. $U_q^j \Rightarrow S_g^m \Longrightarrow S_p^i$.

For case 1, without loss of generality we can assume that U_z^ℓ is the last **update** by process z to be ordered before S_p^i by $\Longrightarrow$. Then by the inductive hypothesis, U_z^ℓ reads $\text{PCOLOR}_{pz}^{i'}$ for some $i' < i$, and it terminates before S_p^i's read in step 2. Since $U_q^j \longrightarrow U_z^\ell$ we have, by the monotonicity of atomic registers, that U_q^j reads $\text{PCOLOR}_{pq}^{i''}$ for some $i'' \leq i' < i$, thereby establishing the second part of the claim. Now, p writes PCOLOR_{pz}^i during the **Produce** operation in step 1 of S_p^i. Since U_z^ℓ reads only an earlier value of PCOLOR_{pz}, it follows from the regularity of atomic registers that U_z^ℓ began before the **Produce** of S_p^i completed. Since $U_q^j \longrightarrow U_z^\ell$ we therefore have that U_q^j completed before this **Produce** is completed, and the claim follows.

For case 2, observe that since $\Rightarrow$ goes only from **update** to **scan** operations, we have that any sequence from S_g^m to S_p^i is either of the form (2.i) $S_g^m \longrightarrow U_z^l \Longrightarrow S_p^i$, or (2.ii) $S_g^m \longrightarrow S_p^i$. By transitivity of $\longrightarrow$, if (2.i) holds, then $U_q^j \longrightarrow U_z^l \Longrightarrow S_p^i$, and if (2.ii) holds, then $U_q^j \longrightarrow S_p^i$. This implies that a minimal sequence from U_q^j to S_p^i cannot be of the form of case 2.

For case 3, we have again that either (3.i) $S_g^m \longrightarrow U_z^l \Longrightarrow S_p^i$, or (3.ii) $S_g^m \longrightarrow S_p^i$. By Lemma 4.2, U_q^j must have completed before S_g^m has completed, and hence it follows analogously to the above that a minimal sequence from U_q^j to S_p^i cannot be of the form of case 3. □

Note that Lemma 4.3 implies that our implementation satisfies Assumption 2.1.

LEMMA 4.4. *For each process, our weak* **scan** *returns the value written by the latest* **update** *ordered before that* **scan** *by* $\Longrightarrow$.

Proof. Let U_q^j be the last **update** by process q to be ordered before S_p^i by $\Longrightarrow$. Since U_q^j is the last **update** of q ordered before S_p^i by $\Longrightarrow$, S_p^i could not have returned $v_q^{j'}$ for some $j' > j$. Therefore, it is enough to show that S_p^i does not return $v_q^{j'}$ for $j' < j$. Suppose otherwise.

From Lemma 4.3 it follows that U_q^j has completed before S_p^i performs its read from q in step 2. Thus before the time S_p^i performs this read, v_q^j is written into VALUE_q. It follows from the code for a **scan** operation that if, after v_q^j is written into VALUE_q, the **scan** S_p^i returns some $v_q^{j'}$ for $j' < j$, then this was the value read by S_p^i in VASIDE_{qp} and S_p^i reads some $\text{QCOLOR}_{qp}^{j''}$ that is the same as PCOLOR_{pq}^i.

Lemma 4.3 states that U_q^j reads $\text{PCOLOR}_{pq}^{i'}$ for some $i' < i$. Thus, when U_q^j reads PCOLOR_{pq} in step 1, PCOLOR_{pq} contains some $\text{PCOLOR}_{pq}^{i'}$ for $i' < i$. The monotonicity of atomic registers immediately implies that also when U_q^k, for $k \leq j$, reads PCOLOR_{pq} in step 1, it reads $\text{PCOLOR}_{pq}^{i''}$ for some $i'' \leq i' < i$. Thus, from step 1 of the **update** operation it follows that for each $k \leq j$, $\text{QCOLOR}_{qp}^k \neq \text{PCOLOR}_{pq}^i$. Thus, the j'' in the above paragraph must be $> j$.

Since, as reasoned above, $\text{QCOLOR}_{qp}^j \neq \text{PCOLOR}_{pq}^i$, there exists $j < j''' \leq j''$ such that $\text{QCOLOR}_{qp}^{j'''} \neq \text{QCOLOR}_{qp}^{j'''-1}$. But this implies that VASIDE_{qp} is updated with $v_q^{j'''-1}$ by **update** $U_q^{j'''}$, contradicting the assumption that S_p^i returns $v_q^{j'}$ read in VASIDE_{qp}, for some $j' < j$. □

4.2. Review of the Traceable Use abstraction. We use a simplified version of the **Traceable Use** abstraction of Dwork and Waarts [20] in order to convert the unbounded weak snapshot described in the previous section into a bounded one. Recall that in the unbounded solution, when process b produces a new color for process c, this new color was not previously produced by b for c. This feature implies that when b sees VALUE_c tagged by this new color it knows that this VALUE_c is too recent (was written after the **scan** began) and will not return it as the result of its **scan**. However, the same property will follow also if when b produces a new color for c, it will simply choose a color that is guaranteed not to tag a value of c unless b produces it for c again. To do this b must be able to detect which of the colors it produced for c may still tag c's values. This requirement can be easily satisfied by incorporating a simplified version of the **Traceable Use** abstraction.

In general, the goal of the **Traceable Use** abstraction is to enable the colors to be *traceable*, in that at any time it should be possible for a process to determine which of its colors might tag any current or future values, where by "future value" we mean a value that has been prepared but not yet written. That is, in the case of time-lapse snapshots, at any time a process p can determine, for each process q, which of p's colors currently appear, or may appear in the future, in PCOLOR_{pq} or QCOLOR_{qp}. Although we allow a color that is marked as "in use" not to be used at all, we require that the number of such colors be bounded.

Due to the asynchrony and concurrency, if revealing a color (i.e., writing a new

color) were done by a simple write, and obtaining a color (i.e., in order to tag a value) were done by a simple read, the colors would not have been traceable. Therefore, to achieve its goal, the **Traceable Use** requires the processes to communicate through three types of wait-free operations—**traceable-read**, **traceable-write**, and **garbage collection**:

- **traceable-read** allows the calling process to obtain the current color produced for it by another process—in other words, to *consume* a color. It takes two parameters: the name c of the process from which the color is being consumed (and to which the color belongs), and the name of the color (that is, the shared variable holding the color).[5] It returns the value of the consumed color.
- **traceable-write** allows a process to update a variable containing its colors—in other words, to *reveal* its new colors. It takes two parameters: the name of the variable and a new value for the variable.
- **garbage collection** allows a process to detect all of its colors that are currently in use. It takes a list of shared variables in which the garbage collector's colors reside (intuitively, the list of where to look for the collector's colors). It returns a list of colors that may currently be in use.

The processes will communicate through the **Traceable Use** abstraction as follows. For process b to obtain a color produced for it by another process, b invokes the **traceable-read** operation. To write a color that other processes may need to obtain from it through the **traceable-read** operation, the process will perform the **traceable-write** operation. Thus, if a variable written by b contains a color of $c \neq b$, this color must have been consumed earlier by b; on the other hand, colors of b appearing in variables owned by b (meaning variables residing in single-writer registers of which b is the writer) do not need to be consumed from anybody. When b writes a new color for itself in one of its variables without consuming it from anybody, we say that this color is *produced* at that time. Finally, to detect which of its colors are in use in the system, the process will perform **garbage collection**.

Each process has $n-1$ *pools* of available colors, one for each other process. When a color v is produced by a process b for process c, it is removed from the pool. The color v is not available for reuse by b (cannot be returned to the pool) before b invokes **garbage collection** and determines that v is not in use by the end of **garbage collection**.

Before we specify the properties of the **Traceable Use**, some notation is in order. First, it is important to distinguish between shared variables of an algorithm that uses the **Traceable Use** abstraction and auxiliary shared variables needed for the implementation of the abstraction. We call the first type of variables *principal* shared variables, and the second type *auxiliary*.[6] Only principal shared variables are obtained through the **traceable-read** operation, revealed through the **traceable-write** operation, and passed to the **garbage collection** procedure. For example, in the weak snapshot system PCOLOR_{pq} and QCOLOR_{pq}, for any p and q, are principal shared variables.

We now specify the situation in which a color is considered currently in use, and therefore not available for reuse. Let E be a finite prefix of an execution of a system in which communication is performed using the **Traceable Use** abstraction. Let E end

[5] For simplicity, the description here is slightly different from the corresponding one in [20].

[6] Sometimes the term *auxiliary variables* denotes variables introduced to facilitate a proof. We are simply using the adjective *auxiliary* in its English sense.

at time t. Then a color v belonging to b is *in use* at time t if there is some extension $\hat{E}$ of E, where b does not produce v in $\hat{E}$ after time t, and v appears as a color of b in some principal shared variable after time t of $\hat{E}$. (Recall that colors produced by b for different processes are considered different even if they have the same value; thus b can produce v only for a specific process, say c.)

Finally, for $1 \leq i \leq n$, let TW_i^k (respectively, TR_i^k) denote the kth **traceable-write** (respectively, **traceable-read**) operation performed by process i. X_i^k denotes a color written by i during TW_i^k. Then **Traceable Use** is required to have the following properties:

- *Regularity*: For any color X_p^a consumed by TR_i^k, TW_p^a begins before TR_i^k terminates, and there is no TW_p^b such that $TW_p^a \longrightarrow TW_p^b \longrightarrow TR_i^k$.
- *Monotonicity*: Let $TR_i^k, TR_j^{k'}$ (where i and j may be equal) be a pair of **traceable-read** operations returning the colors X_p^a, X_p^b, respectively. If $TR_i^k \longrightarrow TR_j^{k'}$, then $a \leq b$.
- *Detectability*: If, during **garbage collection**, a color v of process b was not determined by b to be in use, then v will not be in use before b again produces it.
- *Boundedness*: By taking the local pools to be sufficiently large, it is always possible to find some color not returned as in use by the **garbage collection** procedure.

The *regularity* and *monotonicity* properties of the **Traceable Use** guarantee the *regularity* and *monotonicity* properties of the bounded weak snapshot system. *Detectability* guarantees that a process will be able to safely recycle its colors. *Boundedness* guarantees that by taking the local pools to be sufficiently large, the producer will always find colors to recycle.

Dwork and Waarts [20] presented an implementation of the **Traceable Use** under four restrictions. Since the version we present here is a simplification of their abstraction, three of their restrictions (Uniqueness, Limited Indirection, and Limited Values) are already imposed by the simplification. The remaining restriction is conservation.

- *Conservation.* Let TR_b^k consume a color v from c, and let $TR_b^{k'}$ be the first **traceable-read** operation of b from c to follow TR_b^k. Denote by t' the time in which $TR_b^{k'}$ starts. Then if v appears in a principal shared variable of b at some time $t \geq t'$, it appears in this variable throughout the period $[t', t]$.

Very roughly, the intuition for this restriction is that in the implementation in [20], when b performs **garbage collection** it examines certain auxiliary variables as well as all the principal variables to see which colors are in use. Colors that are or will be in use by c must therefore be in one of those places at all times, and moreover should not be shifting back and forth between those places (because otherwise **garbage collection** could repeatedly miss seeing a color v, reading a register just after v has been moved to a different register and erased from the first one).

In the next section we prove that **Traceable Use** under these restrictions suffices for our weak snapshot algorithm.

4.3. Bounded time-lapse snapshots. For simplicity of exposition, we first present an algorithm that uses registers of size $O(nv)$, where v is the maximum number of bits in any process's value. In section 4.4 we show how to modify this algorithm so that registers of size $O(n + v)$ will suffice.

It is easy to see that the **Traceable Use** abstraction allows us to convert our unbounded solution to a bounded one. In the unbounded solution each time a process

1.a. For all $c \neq b$ $x[c] :=$ **garbage collection**($\text{PCOLOR}_{bc}, \text{QCOLOR}_{cb}$).
1.b. For all $c \neq b$, choose from your local pool for c $pcolor_b[c] \notin x[c]$.
2. **traceable-write** ($\text{PCOLOR}_b, pcolor_b$).

FIG. 4.4. *Bounded* **Produce** *operation for process b.*

performed a **scan** it produced a new color for each updater. In the bounded solution, the scanner will draw its colors from bounded pools, one pool for each updater. (Again, colors produced by different scanners or for different updaters are considered different.) The algorithm is exactly like the unbounded one, except that the processes communicate through the **Traceable Use** operations. Roughly speaking, to get a color produced for it by another process, the process uses the **traceable-read** operation; to write its new colors, it uses the **traceable-write** operation; and to find a color it can produce again, it performs **garbage collection**.

Using the **Traceable Use** abstraction, the process will be able to narrow the set of its colors suspected by it as colors that may tag values of another process, say c, to contain no more than six values (discussed below). We take the pools to be of size seven, thereby ensuring that whenever the process wants to perform a **scan**, it can find, for each updater, a color it can reuse.

More specifically, in the bounded solution, every process has $n-1$ pools of available colors, one pool for each other process. Each pool initially contains seven colors. When a scanner b wants to produce a new color for updater c, it performs **garbage collection** to determine which of its colors may currently be tagging values of c (and therefore cannot be reused at this time). Any color determined by **garbage collection** not to be in use can be recycled without causing confusion. In general, **garbage collection** ensures that when a process b produces v as its color for c for the kth time, there is not, and never will be, anything in the system that contains v from when it was produced by b for c for the $(k-1)$st time. For example, suppose b produces the color 5 at time t_1, later performs **garbage collection**, and then later, at time t_2, again produces the color 5. Thus, by definition, at time t_1, b performs an atomic write to PCOLOR_b, setting $\text{PCOLOR}_{bc} := 5^{t_1}$, where the t_1 in the superscript indicates that this is the value written at time t_1. Similarly, at time t_2, b performs an atomic write setting $\text{PCOLOR}_{bc} := pcolor_b[c] = 5^{t_2}$. Since this second write occurs as part of a **scan** operation, if b completes the **scan**, then at some time $t > t_2$, when performing step 2 of this **scan**, b sets $qcolor_b[c] := \text{QCOLOR}^i_{cb}$ and later, in step 3, tests $qcolor_b[c] = pcolor_b[c]$ $(= 5)$. Suppose that the two colors are equal, that is, that $\text{QCOLOR}^i_{cb} = 5$. Then the **garbage collection** procedure ensures that $\text{QCOLOR}^i_{cb} = 5^{t_2}$ and *not* 5^{t_1}, that is, the **update** operation U^i_c reads 5^{t_2}, the result of the more recent write of 5, in PCOLOR_{cb}.

In order to convert the unbounded solution to a bounded one, we replace the **Produce** operation shown in Figure 4.3 by the **Produce** operation shown in Figure 4.4. The meaning of the notation in step 2 of the new **Produce** operation is that all $n-1$ colors $pcolor_b[i]$, $i \neq b$, are written atomically to PCOLOR_{bi}.

Also, line 1 of the **update** operation shown in Figure 4.1 is replaced by the following:

1. For all $c \neq b$, $qcolor_b[c] :=$ **traceable-read**(c, PCOLOR_{cb}).

We refer to the modified **scan** and **update** as the bounded **scan** and bounded **update**, respectively.

We now prove that the resulting bounded construction is a time-lapse snapshot algorithm. The proof follows along the same lines as for the unbounded construction of section 4.1.

First, the partial linearization order $\Longrightarrow$ on the bounded **scan** and **update** operations is defined identically to the one defined in section 4.1 on the unbounded **scan** and **update** operations. It follows from Lemmas 4.5 and 4.8 that the bounded **scan** and **update** procedures yield a weak snapshot memory.

LEMMA 4.5. *The relation $\Longrightarrow$ is a partial order.*

Proof. The proof is analogous to the proof of Lemma 4.1, when "regularity of **Traceable Use**" is replacing "regularity of atomic registers." □

LEMMA 4.6. *If a bounded* **scan** *returns a value written by a bounded* **update**, *then this* **update** *terminates before the* **scan** *reads* VALUE_q.

Proof. The proof is identical to the proof of Lemma 4.2. □

LEMMA 4.7. *Let U_q^j be the last* **update** *by process q to be ordered before S_p^i by $\Longrightarrow$. U_q^j terminates before S_p^i reads VALUE_q. Moreover, in step 1 of U_q^j, q reads $\text{PCOLOR}_{pq}^{i'}$ for some $i' < i$.*

Proof. The proof is analogous to the proof of Lemma 4.3, when "regularity of **Traceable Use**" and "monotonicity of **Traceable Use**" are replacing "regularity of atomic registers" and "monotonicity of atomic registers," respectively, and in addition Lemma 4.6 is replacing Lemma 4.2. □

Note that Lemma 4.7 implies that our bounded construction satisfies Assumption 2.1.

LEMMA 4.8. *For each process, our bounded weak* **scan** *returns the value written by the latest bounded* **update** *ordered before that* **scan** *by $\Longrightarrow$.*

Proof. The proof is along the same lines of the proof of Lemma 4.4. The difference here is that in the bounded implementation, QCOLOR_{qp}^m may contain the same color as PCOLOR_{pq}^r even if U_q^m does not consume PCOLOR_{pq}^i but consumes some $\text{PCOLOR}_{pq}^{i'}$ for some $i' \neq i$.

Let U_q^j be the last **update** by process q to be ordered before S_p^i by $\Longrightarrow$. Since U_q^j is the last **update** of q ordered before S_p^i by $\Longrightarrow$, S_p^i could not have returned $v_q^{j'}$ for some $j' > j$. Therefore, it is enough to show that S_p^i does not return $v_q^{j'}$ for $j' < j$. Suppose otherwise.

From Lemma 4.7 it follows that U_q^j has completed before S_p^i performs its read from q in step 2. So before the time S_p^i performs this read, v_q^j is written into VALUE_q. It follows from the code for a **scan** operation that if, after v_q^j is written into VALUE_q, the **scan** S_p^i returns some $v_q^{j'}$ for $j' < j$, then this was the value read by S_p^i in VASIDE_{qp} and S_p^i reads some $\text{QCOLOR}_{qp}^{j''}$ that is the same as PCOLOR_{pq}^i.

Let QCOLOR_{qp}^k be the QCOLOR_{qp} read by S_p^i when S_p^i performs its **garbage collection** operation. Step 1.b of the bounded **Produce** operation implies that QCOLOR_{qp}^k contains a different color than the color contained in PCOLOR_{pq}^i.

Therefore, there exists $k < j''' \leq j''$ such that $\text{QCOLOR}_{qp}^{j'''} \neq \text{QCOLOR}_{qp}^{j'''-1}$. Step 2 of the **update** operation implies that VASIDE_{qp} is updated with $v_q^{j'''-1}$ by **update** $U_q^{j'''}$. Thus, if $j \leq k$, then we get a contradiction to the assumption that S_p^i returns $v_q^{j'}$ read in VASIDE_{qp}, for some $j' < j$. To complete the proof, assume $j \geq k$.

Lemma 4.7 states that U_q^j reads $\text{PCOLOR}_{pq}^{i'}$ for some $i' < i$. The detectability property of the **Traceable Use** implies that after the **garbage collection** done in step 1 of S_p^i, the color contained in PCOLOR_{pq}^i may appear in some $\text{QCOLOR}_{qp}^{k'}$ for $k' \geq k$ (and hence be in use) only if p produces it again and $U_q^{k'}$ consumes $\text{PCOLOR}_{pq}^{i''}$ for some $i'' \geq i$. Thus, QCOLOR_{qp}^j does not contain the same color as PCOLOR_{pq}^i.

Thus there exists $j < j''' \leq j''$ such that $\text{QCOLOR}_{qp}^{j'''} \neq \text{QCOLOR}_{qp}^{j'''-1}$. Step 2 of the **update** operation implies that VASIDE_{qp} is updated with $v_q^{j'''-1}$ by **update** $U_q^{j'''}$, contradicting again the assumption that S_p^i returns $v_q^{j'}$ read in VASIDE_{qp}, for some $j' < j$. □

Lemmas 4.5 and 4.8 immediately imply the following.

THEOREM 4.9. *The bounded construction is a time-lapse snapshot algorithm.*

Letting n be the number of processes in the system, the implementation of **Traceable Use** given in [20] requires $O(n)$ steps per **traceable-write** (that is, $O(n)$ reads and writes of shared variables) and $O(1)$ steps to consume one color using the **traceable-read** operation. Applied to our algorithm, the construction in [20] yields a cost of $O(k)$ for **garbage collection**, where k is the number of variables passed as parameters to the **garbage collection** procedure; moreover, when b invokes **garbage collection** to detect how many of its colors for c are in use, **garbage collection** detects at most six colors.[7]

Clearly, each **Produce** operation requires $O(n)$ invocations of **garbage collection**, each of which costs $O(1)$, and one invocation of **traceable-write**, and thus takes $O(n)$ steps. Each **scan** requires one invocation of **Produce** and $O(n)$ simple reads, and thus takes $O(n)$ steps. Each **update** requires one simple write and $O(n)$ invocations of **traceable-read**, and hence takes $O(n)$ steps.

4.4. Reducing the register size. The weak snapshot described above uses registers of size $O(nv)$ where v is the maximum number of bits in any variable VALUE_b. This is due to the fact that an updater b may set aside a different value for each scanner c in a variable VASIDE_{bc}, and all these values are kept in a single register. To reduce the size of the registers, each updater b stores VASIDE_{bc} in a separate register for each c. Only after this has been accomplished, b atomically updates VALUE_b and, for all $c \neq b$, QCOLOR_{bc}.

The modifications to the code are straightforward. Lines 2 and 3 of the code for the **scan** (Figure 4.2) are replaced by lines 2′ and 3′ below.

2′. For all $c \neq b$ atomically read
  $value_b[c] := \text{VALUE}_c$
  $qcolor_b[c] := \text{QCOLOR}_{cb}$.
3′. For all $c \neq b$
  If $qcolor_b[c] \neq pcolor_b[c]$
    then $data_b[c] := value_b[c]$
    else read $data_b[c] := \text{VASIDE}_{cb}$.

Lines 2 and 3 of the code for the **update** operation (Figure 4.1) are replaced by the following lines 2′ and 3′:

2′. For all $c \neq b$
  if $qcolor_b[c] \neq \text{QCOLOR}_{bc}$ then $vaside_b[c] := \text{VALUE}_b$
  $\text{VASIDE}_{bc} := vaside_b[c]$.
3′. Atomically write
  $\text{VALUE}_b :=$ new value
  For all $c \neq b$, $\text{QCOLOR}_{bc} := qcolor_b[c]$.

[7] This is because, in the notation of [20], when a process b wants to detect which of its colors for c may currently be in use, it needs only to read $Ai\text{-}PCOLOR_{bc}[b]$, for $i = 1, 2, 3$ (colors set aside by b for c when b detects that c is trying to consume b's color for c), $B\text{-}PCOLOR_{cb}[b]$ (color written by c for b when c is trying to consume b's color for it), and QCOLOR_{cb}; moreover, each **garbage collection** needs to read these variables only once because in our application colors have what [20] calls indirection of degree one (see section 8.3 of [20]).

Observe that the time complexity of the modified algorithm is the same as the original one.

THEOREM 4.10. *The modified algorithm is a time-lapse snapshot algorithm.*

Proof. Clearly, the only difference between the modified and the original algorithms is that the shared variables VASIDE_{qp} and VALUE_q are not written atomically together by the **update** and are not read atomically together by the **scan**. If in every execution of the modified algorithm whenever a scan S_p^i reads VALUE_q^k and $\text{VASIDE}_{qp}^{k'}$ (recall that VASIDE_{qp}^k is the value of VASIDE_{qp} written by U_q^k) either S_p^i returns VALUE_q^k or $\text{VASIDE}_{qp}^k = \text{VASIDE}_{qp}^{k'}$, then every execution of the modified algorithm has a corresponding execution of the original algorithm, in which corresponding **scan**s return the same sets of values, and the modified algorithm is correct because the original algorithm is correct.

Therefore, let us assume for the sake of contradiction that the **scan** S_p^i reads VALUE_q^k and $\text{VASIDE}_{qp}^{k'}$ and returns the latter, where the value of VASIDE_{qp}^k is different from the value of $\text{VASIDE}_{qp}^{k'}$. We now show that this cannot happen. Since VASIDE_{qp} is written before VALUE_q by an **update** and read after it by a **scan**, we have that $k' > k$. Since the **scan** S_p^i returns the value it read from VASIDE_{qp}, we have $\text{PCOLOR}_{pq}^i = \text{QCOLOR}_{qp}^k$. It follows from the detectability property of **Traceable Use** that, after GC_p^i, PCOLOR_{pq}^i is not in use it will not be used again until p produces it. So the chronology is: S_p^i performs **garbage collection**, denoted GC_p^i; S_p^i reads QCOLOR_{qp} during GC_p^i; S_p^i sets PCOLOR_{pq}^i to a value not currently in use; U_q^k writes $\text{QCOLOR}_{qp}^k = \text{PCOLOR}_{pq}^i$; S_p^i reads QCOLOR_{qp}^k. Thus, U_q^k consumes color PCOLOR_{pq}^i.

By monotonicity of **Traceable Use**, for all $k < k_1 \le k'$, $U_q^{k_1}$ consumes PCOLOR_{pq}^i and hence saw $qcolor_q[p] = \text{QCOLOR}_{qp}$ when performing step 2 of the **update** operation. Hence no such $U_q^{k_1}$ changed the value in VASIDE_{qp}, i.e., the value of VASIDE_{qp}^k is the same as the value of $\text{VASIDE}_{qp}^{k'}$. This is a contradiction. ∎

5. Applications. In this section, we explore two applications of the weak snapshot: bounded concurrent timestamping and randomized consensus. First we take the bounded concurrent timestamping protocol of Dolev and Shavit [19] and show that the labels can be stored in an abstract weak snapshot object, where each access to the labels is through either the weak snapshot **update** or the weak snapshot **scan**. The resulting protocol has running time, label size, and register size all $O(n)$.

We then take the elegant randomized consensus protocol of Aspnes [5], and show that replacing atomic snapshot with weak snapshot leads to an algorithm with an expected number $O(n(p^2+n))$ of operations, matching the fastest bounded algorithm known [12]. This is an improvement of $\Omega(n)$ over the original algorithm and an improvement of $\Omega(\log n)$ over what could be obtained by using the atomic snapshot of [11].

5.1. Efficient bounded concurrent timestamping. Our definition of a concurrent timestamping system is a slightly stronger version (due to Gawlick [21]) of the one given by Dolev and Shavit in [19]. In a *concurrent timestamping system*, processes repeatedly choose *labels*, or timestamps, reflecting the real-time order of events. More precisely, each process i has a label, denoted by ℓ_i. There are two kinds of operations: **labeling** generates a new timestamp for the calling process, and **scan** returns an indexed set of labels $\bar{\ell} = \langle \ell_1, \ldots, \ell_n \rangle$ and an irreflexive total order $\prec$ on the labels.

For $1 \le i \le n$, let L_i^k (S_i^k) denote the kth **labeling** (**scan**) operation performed

1. For all c, read $label_c := \text{LABEL}_c$.
2. $label_b := f(label_1, \ldots, label_n)$.
3. Atomically write $\text{LABEL}_b := label_b$.

FIG. 5.1. **labeling** *operation for process b (Dolev–Shavit* [21]*).*

by process i (process i need not keep track of k; this is simply a notational device allowing us to describe long-lived runs of the timestamping system). Analogously, ℓ_i^k denotes the label obtained by i during L_i^k. In order to handle initial conditions, we assume that each processor has an initial label denoted by ℓ_i^0. To avoid distinguishing between these initial labels and the labels assigned by **labeling** operations we say that label l_i^0 was assigned to process i by a fictitious initial **labeling** operation L_i^0 that took place just before the beginning of the execution. Moreover, we define the L_i^0's for all i as concurrent. Correctness is defined by the following properties:

- *Ordering.* There exists an irreflexive total order $\stackrel{ts}{\Longrightarrow}$ on the set of all **labeling** operations,[8] such that the following apply:
  - *Precedence.* For any pair of **labeling** operations L_p^a and L_q^b (where p and q may be equal), if $L_p^a \longrightarrow L_q^b$, then $L_p^a \stackrel{ts}{\Longrightarrow} L_q^b$.
  - *Consistency.* For any **scan** operation S_i^k returning $(\bar{\ell}, \prec)$, $\ell_p^a \prec \ell_q^b$ if and only if $L_p^a \stackrel{ts}{\Longrightarrow} L_q^b$.
- *Regularity.* For any label ℓ_p^a in $\bar{\ell}$ returned by S_i^k, L_p^a begins before S_i^k terminates, and there is no L_p^b such that $L_p^a \longrightarrow L_p^b \longrightarrow S_i^k$.
- *Monotonicity.* Let $S_i^k, S_j^{k'}$ (where i and j may be equal) be a pair of **scan** operations returning the vectors $\bar{\ell}, \bar{\ell}'$, respectively, which contain labels ℓ_p^a, ℓ_p^b, respectively. If $S_i^k \longrightarrow S_j^{k'}$, then $a \leq b$.
- *Extended regularity.* For any label l_p^a returned by S_i^k, if $S_i^k \longrightarrow L_q^b$, then $L_p^a \stackrel{ts}{\Longrightarrow} L_q^b$.

Dolev and Shavit describe a *bounded* concurrent timestamping system that uses atomic multireader registers of size $O(n)$ and whose **scan**[9] and **labeling** operations take time $O(n^2 \log n)$ and $O(n)$, respectively. Their **labeling** operation was simply the performance of a **collect** of all labels[10] and the writing of a new label based on the ones collected. A paraphrased version of their code appears in Figure 5.1, where the variables $label_c$ and LABEL_c are the local and global shared variables that contain the current labels of process c; and f denotes the function used by Dolev and Shavit to compute a new label. The exact details of this function are not relevant here, as will be seen. The Dolev–Shavit **scan** was more involved. A paraphrased version of their code appears in Figure 5.2, where g is a procedure for computing the order among the obtained labels. Again, the exact values of k and details about g are not relevant here.

Dolev and Shavit observed that the labels can be stored in an abstract atomic snapshot object, where each access to the labels is through either atomic snapshot **update** or **scan** operation. More specifically, they would replace the **collect** performed during the **labeling** operation by an atomic snapshot **scan**, replace the simple writing

[8] Observe that this order does not have to be consistent with the partial linearization order on the time-lapse snapshot **update** and **scan** operations.

[9] Note that this **scan** is different from our time-lapse snapshot **scan**.

[10] We say that a process *collects* a variable X if it reads X_c for every process c in some arbitrary order.

1. For $i = 1$ to k
   For all c, read $label_c^i := \text{LABEL}_c$.
2. Select for each c, $label_c := label_c^i$ for some i.
3. Order $(label_1, \ldots, label_n)$ using procedure g.
4. Return $(label_1, \ldots, label_n)$ and their order.

FIG. 5.2. *Timestamping* **scan** *operation for process b (Dolev–Shavit* [21]*).*

1. Perform a time-lapse **scan** to read for all c, $label_c := \text{LABEL}_c$.
2. $label_b := f(label_1, \ldots, label_n)$.
3. Perform a time-lapse **update** to write $\text{LABEL}_b := label_b$.

FIG. 5.3. *Our* **labeling** *operation for process b.*

of the new label with an atomic snapshot **update**, and replace their entire original **scan** with an atomic snapshot **scan**. As mentioned in the introduction, this approach was successfully pursued by Gawlick, Lynch, and Shavit [22].

However, as Dolev and Shavit note, this transformation has drawbacks. The size of the atomic registers in all known implementations of atomic snapshot memory is $O(nv)$, where v is the size of the value of each process, and hence the size of the atomic registers in the resulting timestamping system is $O(n^2)$ (because here v is a label, and their labels are of size n). Second, since both **update** and **scan** operations of the snapshot take $O(n \log n)$ steps [11],[11] so do both **labeling** and **scan** operations of the resulting timestamping system. Hence, while the number of steps performed by a timestamping **scan** in the resulting timestamping system improves, the running time of the **labeling** operation increases.

We show that when we modify the Dolev–Shavit timestamping system by replacing the **collect** performed during the **labeling** operation of [19] by a time-lapse snapshot **scan**, replacing their simple writing of the new label with a time-lapse snapshot **update**, and replacing their entire original **scan** with a time-lapse snapshot **scan**, we get a timestamping system with linear running time, register size, and label size. The code for the resulting **labeling** operation appears in Figure 5.3, where f is the same function appearing in Figure 5.1. The code for the resulting timestamping **scan** operation appears in Figure 5.4, where g is the same procedure as the one of Figure 5.2.

Next we prove that the resulting system is indeed a bounded concurrent timestamping system. The proof of correctness of the Dolev–Shavit algorithm is long and involved. Rather than prove from scratch the correctness of this new algorithm (a very involved process; see [19, 22]), we argue by reduction to the correctness of the original algorithm.

LEMMA 5.1. *Our modification of the Dolev–Shavit algorithm satisfies the* regularity *and* monotonicity *properties.*

Proof. Regularity follows directly from the regularity property of weak snapshots, and monotonicity follows directly from the monotonicity of **scans** property of weak snapshot. □

The following lemma will be used to show that our modification of the Dolev–

[11]The atomic snapshot algorithm in [11], obtained more than a year after the results of this paper were obtained, is the most efficient atomic snapshot algorithm using atomic single-writer–multireader registers currently known. At the time the results of the present paper were obtained, both **labeling** and **scan** operations of the best atomic snapshot algorithm [2] took $O(n^2)$ steps.

1. Perform a time-lapse **scan** to read for all c, $label_c := \text{LABEL}_c$.
2. Order $(label_1, \ldots, label_n)$ using procedure g.
3. Return $(label_1, \ldots, label_n)$ and their order.

FIG. 5.4. *Our timestamping* **scan** *operation for process b.*

Shavit algorithm satisfies the ordering property.

LEMMA 5.2. *For each execution E of our algorithm, there exists a corresponding execution of the Dolev–Shavit algorithm that produces the same sequence of* **labeling** *operations, with corresponding* **labeling** *operations producing the same labels in both executions.*

Proof. Consider an execution L_p of the **labeling** operation. In our algorithm, when a process performs a **labeling** operation it obtains the labels of the other processes using a time-lapse snapshot **scan** and then writes the new label using a time-lapse snapshot **update**, while in the original algorithm of Dolev and Shavit these labels are obtained using a simple **collect** (one atomic read from each other process), and are written using a simple atomic write. However, since when a process performs a **collect** it reads all labels once each in an arbitrary order, and since by definition each label returned by a weak snapshot **scan** was written by a **labeling** operation, say L_q^k, that by Assumption 2.1 terminated before the **scan** did and, by regularity of time-lapse **scan**, such that no later L_q terminated before the **scan** started, we have that the set of labels returned by the weak snapshot **scan** executed by L_p could also have been returned by a simple **collect** executed in the same time interval. The claim follows because the label written by the **labeling** operation in either algorithm depends only on the set of labels obtained during this operation. □

Next we show that the first part of the ordering property is satisfied.

LEMMA 5.3. *There exists an irreflexive total order on the* **labeling** *operations in the execution of our algorithm that is consistent with the precedence relation on the* **labeling** *operations.*

Proof. This is immediate from Lemma 5.2. The total order is simply the one guaranteed by the Dolev–Shavit algorithm for the corresponding execution. More specifically, given an execution the total order on the **labeling** operations is as follows: if one **labeling** operation reads (through a weak snapshot **scan**) the label produced by another **labeling** operation, then the first operation is ordered after the second. To get the total order, Dolev and Shavit take the transitive closure of this partial order and extend it to a total order by considering the values of the labels taken by the **labeling** operations. (The exact details of how these values are taken into consideration to determine the total order are not relevant here.) □

The following lemma shows that the second part of the ordering property also is satisfied.

LEMMA 5.4. *The order produced by a* **scan** *operation of our timestamping algorithm is consistent with the total order described in the proof of Lemma* 5.3.

Proof. A **scan** operation of our timestamping algorithm obtains a set of labels via an invocation of a weak snapshot **scan**. Consider a weak snapshot **scan**, say S, performed in an execution E of our algorithm, that returns a set of labels $\bar{\ell}$. To compute the order on these labels, our algorithm invokes the appropriate procedure in the Dolev–Shavit algorithm. Therefore, it remains to show that the order on these labels produced by this procedure is consistent with the total order on the **labeling** operations defined above.

We do this in two parts. In the first part we begin with **scan** S in an execution E

of our algorithm and construct a modified execution $E1$, also containing S. We then examine a corresponding execution $\hat{E}1$ of the Dolev–Shavit algorithm, containing a corresponding **scan** $\hat{S}$ and returning the same set of labels as does S. Since the two algorithms use the same procedure to order the labels returned by a **scan**, we have by the correctness of the Dolev–Shavit algorithm that the order determined by S is consistent with the total order on the **labeling** operations in $E1$. In the second part we show that the total order on the **labeling** operations in $E1$ is consistent with their order in E.

Define a modified execution $E1$ of our algorithm where we stop each process in E after it completes the **labeling** operation that generates its label in $\bar{\ell}$. Think of a **labeling** operation L_a as a pair of time-lapse snapshot operations (S_a, U_a), where $S_a \longrightarrow U_a$. We write $L_b \Longrightarrow' L_a$ (read, L_a *observes* L_b) if $U_b \Longrightarrow S_a$.

CLAIM 5.5. *If a* **labeling** *operation L_a appears in both $E1$ and E, and $L_b^i \Longrightarrow' L_a$ in E, then L_b^i is in $E1$.*

Proof. Since L_a is in $E1$ we have that $U_a \Longrightarrow S$ in E. By assumption $L_b^i \Longrightarrow' L_a$, and thus $U_b^i \Longrightarrow S_a$, and hence $U_b^i \Longrightarrow S_a \longrightarrow U_a \Longrightarrow S$, whence by transitivity $U_b^i \Longrightarrow S$. By definition of time-lapse snapshot, S returns $v_b^{i'}$ for $i' \geq i$. Hence L_b^i is in $E1$. □

It follows from Claim 5.5 that the set of current labels in the end of $E1$ is still $\bar{\ell}$. Now, consider an execution $\hat{E}1$ of the Dolev–Shavit algorithm that corresponds to our modified execution, and extend this corresponding execution by executing at the end a **scan**, say $\hat{S}$, of the Dolev–Shavit algorithm. Clearly, $\hat{S}$ returns the same labels as in $\bar{\ell}$. The order of the labels computed by the Dolev–Shavit **scan** $\hat{S}$ is consistent with the total order on the **labeling** operations in $\hat{E}1$ (by the correctness of the Dolev–Shavit algorithm). Since the order determined on $\bar{\ell}$ by S is exactly the same as their order determined by $\hat{S}$, and since the total order on the **labeling** operations in $E1$ was defined to be identical to their order in $\hat{E}1$, we have that the order on $\bar{\ell}$ determined by S is consistent with the total order on the **labeling** operations in $E1$.

To complete the proof of the ordering property we show that the total order on the **labeling** operations in $E1$ is consistent with their order in E. That is, we show that for any two **labeling** operations L_a and L_b that appear in both executions E and $E1$, L_a is ordered before L_b in the total order on **labeling** operations in E only if L_a is ordered before L_b in the total order on **labeling** operations in $E1$. Recall how the total order is determined (see the proof of Lemma 5.3). By Claim 5.5, $L_a \Longrightarrow' L_b$ in $E1$ if and only if $L_a \Longrightarrow' L_b$ in E (and vice versa). Thus, if L_a is ordered before L_b in E and after L_b in $E1$ it must be that the order between L_a and L_b in both executions is determined based on the values of the labels taken. Thus one of these operations, without loss of generality, L_a, results in a different value in E than in $E1$. However, for this to occur there must be **labeling** operation L_c such that $L_c \Longrightarrow' L_a$ in E but $L_c \not\Longrightarrow' L_a$ in $E1$, contradicting Claim 5.5. This completes the proof of the lemma. □

Finally we show the following.

LEMMA 5.6. *Our modification of the Dolev–Shavit algorithm satisfies the* extended regularity *property.*

Proof. Recall that our time-lapse snapshot algorithm satisfies Assumption 2.1: a value returned by a **scan** was written by an **update** operation that terminated before the **scan** does. Thus, since in our timestamping system the last operation performed by a **labeling** operation is writing a new label, and since this is done by a weak snapshot **update**, each label returned by a timestamping **scan** of our system is

assigned by a **labeling** operation, say L_1, that terminated before the **scan** does, and hence before the beginning of any **labeling** operation, say L_2, preceded by the **scan**. The precedence property of timestamping implies immediately that $L_1 \stackrel{ts}{\Longrightarrow} L_2$. □

Lemmas 5.1, 5.3, 5.4, and 5.6 immediately imply the following theorem.

THEOREM 5.7. *Our modification of the Dolev–Shavit algorithm yields a bounded concurrent timestamping system.*

5.2. Efficient randomized consensus. In a *randomized consensus protocol*, each of n asynchronous processes starts with a *preference* taken from a two-element set (typically $\{0, 1\}$), and runs until it chooses a *decision value* and halts. The protocol is correct if it is *consistent*—no two processes choose different decision values; *valid*—the decision value is some process's preference; and *randomized wait-free*—each process decides after a finite expected number of steps. When computing a protocol's expected number of steps, we assume that scheduling decisions are made by an *adversary* with unlimited resources and complete knowledge of the processes' protocols, their internal states, and the state of the shared memory. The adversary cannot, however, predict future coin flips.

Celebrated work of Rabin [39] reduces randomized consensus to the construction of a *global coin*: a source of randomness visible to all participants in the protocol. Virtually all randomized consensus algorithms rely on some form of a global coin; indeed, the construction of the coin is invariably the bulk of the effort (see [16] for a survey of results on this problem in the message-passing model).

The starting point for our algorithm is the randomized consensus protocol of Aspnes [5], and in particular Aspnes's *robust weak shared coin* protocol, which guarantees that all participating processes agree on the outcome of the coin flip, and an adversary scheduler has only a slight influence on the outcome. Roughly speaking, in this protocol the n processes collectively undertake a one-dimensional random walk centered at the origin with absorbing barriers at $\pm 2n$. The shared coin is implemented by a shared counter. Each process alternates between reading the counter's position and updating it. Eventually the counter reaches one of the absorbing barriers, determining the decision value. While the counter is near the middle of the region, each process flips an unbiased local coin to determine the direction in which to move the counter. If a process observes that the counter is within n of one of the barriers, however, the process moves the counter deterministically toward that barrier.

More specifically, Figure 5.5 shows pseudocode for each process's behavior in the randomized consensus protocol. The protocol uses three shared counters: the first two maintain a total of the number of participating processes that started with inputs 0 and 1, respectively, and the last is used as the counter for the robust weak shared coin protocol. All of the counters start with an initial value of 0.

The protocol is optimized for the case where few processes participate. A process is defined to be *active* if it takes at least one step before some process decides on a value, and p denotes the total number of active processes in a given execution. The protocol uses counters a_0 and a_1 to keep track of the number of active processes by having each process increment one or the other of these counters as it starts the protocol.

Aspnes implements each of the three shared counters as an n-element array of atomic single-writer–multireader registers, one per process. All three counters are read by a single atomic snapshot scan operation that reads the arrays implementing them; i.e., all three n-elements arrays are read in one atomic snapshot scan operation that returns a value for each of the three counters. To increment or decrement counter

```
Shared data:
counter a_0 with range [0, n] and initial value 0
counter a_1 with range [0, n] and initial value 0
counter c with range [−4n, 4n] and initial value 0

FUNCTION Consensus(input)
increment a_input
repeat
        read(a_0, a_1, c)
        if c ≤ −2n then decide 0
        else if c ≥ 2n then decide 1 fi
        else if c ≤ −(a_0 + a_1) or a_1 = 0 then decrement(counter) fi
        else if c ≥ (a_0 + a_1) or a_0 = 0 then increment(counter) fi
        else
                if coin() = 0 then decrement(counter)
                else increment(counter)
        fi
end
```

FIG. 5.5. *Randomized consensus protocol (Aspnes* [5]*).*

c, a process updates, through an atomic snapshot update operation that operates on all three counters, its own field in the n-element array that implements counter c. Careful use of modular arithmetic ensures that all values in the counter implementing the robust weak shared coin remain bounded.

The expected running time of this consensus protocol, expressed in primitive reads and writes, is $O(n^2(p^2 + n))$, where p is the number of processes that actually participate in the protocol.

Since the three counters are updated and read through atomic snapshot operations, each of these counters is *linearizable* [25]: There exists a total order "$\overset{c}{\Longrightarrow}$" on increment, decrement, and read operations on each counter such that

- if $A \longrightarrow B$, then $A \overset{c}{\Longrightarrow} B$;
- each *Read* operation returns, for each counter, the sum of all increments and decrements on the counter, ordered before the *Read* by $\overset{c}{\Longrightarrow}$.

We replace the linearizable counter with a different data abstraction: by analogy with the definition of weak snapshot, a *weak counter* imposes the same two restrictions but allows $\overset{c}{\Longrightarrow}$ to be a partial order instead of a total order. Informally, concurrent *Read* operations may disagree about concurrent increment and decrement operations, but no others. We can construct a weak counter implementation from Aspnes's linearizable counter implementation simply by replacing the atomic snapshot scan (update) with a weak snapshot scan (update). We now argue that the consensus protocol remains correct if we replace the linearizable counter with a more efficient weak counter.

First notice that counters a_0 and a_1 allow the protocol to guarantee validity, since the random walk is invoked only if both have nonzero values. These counters are also used to minimize the range of the random walk, as follows. Define the *true position* of the random walk at any instant to be the value the random walk counter would assume if all operations in progress were run to completion without starting any new operations.

For reads, increments, and decrements of the counter, let R_p^i denote the ith read operation of process p, let I_q^j denote the jth increment operation by q, and let D_q^j denote the jth decrement operation by q.

Recall the definition of t_{scan} from section 3 (the latest point during the interval of the **scan** in which the spans of all values returned by the **scan** intersect). Our discussion there implies that a **scan** observes all **update**s that terminate before its t_{scan}, and does not observe any **update** that starts after its t_{scan}.

Aspnes's proof of the expected running time of the protocol hinges on the following lemma, which holds even if we replace the atomic snapshot scan by a weak snapshot scan.

LEMMA 5.8. *Let τ be the true position of the random walk at t_{scan} of R_p. If R_p returns values c, a_0, and a_1 for the random walk counter and the two active counters, then $c - (a_0 + a_1 - 1) \leq \tau \leq c + (a_0 + a_1 - 1)$.*

Proof. A process q affects the random walk's true position only if it has started to increment or decrement the random walk counter by time t_{scan}. Any q that has started to modify the random walk counter by the t_{scan} of R_p has already finished incrementing the appropriate active counter before that time, so R_p observes that increment. Thus, R_p fails to observe at most $(a_0 + a_1 - 1)$ increments or decrements active at its t_{scan}, and the result follows. □

Aspnes [5] shows that Lemma 5.8 implies the following theorem.

THEOREM 5.9 (Lemma 13 in [5]). *Let n be the total number of processes and let p be the number of processes that take at least one step before some process decides on a value. Then the worst-case expected running time of the consensus protocol is $O(p^2+n)$ counter operations (i.e., scan or update operations on the shared counters).*

A detailed proof appears in [5]. Roughly speaking, since p is both an upper bound on the distance between the value c read by a process from the random walk counter and the true position τ, and on the value of $a_0 + a_1$, we have that if $|\tau| \geq 2p$, then $|c| \geq a_0+a_1$ and the true position will move away from 0 thereafter. Thus, the random walk has two absorbing barriers at $2p, -2p$, and hence takes expected $O(p^2)$ counter operations; to this value must be added $O(n)$ counter operations until termination.

Theorem 5.9 implies the running time of our modification of Aspnes's algorithm. Recall that in our modification of this algorithm each counter operation is implemented by a time-lapse **scan** or **update**, and hence each such operation takes $O(n)$ atomic steps. Thus, Theorem 5.9 implies that the worst-case expected running time of our modification of Aspnes's algorithm is $O(n(p^2 + n))$ atomic read and writes operations.

The proof that the modified consensus protocol is consistent depends on the following lemma, which is analogous to a similar lemma in [5] and which, roughly speaking, ensures that if any one process decides that the outcome of the coin is a given value, say v, then all other processes will see values that cause them to push the counter toward v (see the code in Figure 5.5).

LEMMA 5.10. *If R_p^i returns value $v \geq 2n$, then all reads whose t_{scan} is not smaller than the t_{scan} of R_p^i will return values $\geq n+1$. (The symmetric claim holds when $v \leq -2n$.)*

Proof. Suppose not. Pick an earliest (with respect to t_{scan}) R_q^j that violates the hypothesis. Denote the t_{scan}'s of R_p^i, R_q^j by t_p, t_q, respectively. Clearly, any difference in the values returned by R_q^j and R_p^i must be caused by **update**s observed by exactly one of them (not necessarily the same one for each such **update**). It follows from the definition of t_{scan} that each **update** that completed before time t_p was observed by

R_p^i; i.e., all these **update**s are ordered before it by $\Longrightarrow$. Moreover, since $t_q \geq t_p$, it follows that each such **update** was also observed by R_q^j. Thus, only **update**s that completed after t_p may be seen by exactly one of R_p^i, R_q^j.

Consider the set S containing all **update**s that completed at or after t_p excluding, for each $z \neq p$, the first such **update**. Since each of the **update**s in S started after t_p, none of them was observed by R_p^i. Let $S1$ denote the subset of S that contains all **update**s in S that were observed by R_q^j. To complete the proof it is enough to show that each **update** in $S1$ was an increment. This immediately gives a contradiction to our assumption that R_q^j returns a value $< n+1$.

Since processes alternate between reading and modifying the counter, any **update** U_z^k in $S1$ must follow a read R_z^k that started at or after t_p and hence the t_{scan} of this read (t_z) is not smaller than t_p. In addition, any such R_z^k completes before t_q (because, since U_z^k is observed by R_q^j, it must have started at or before t_q), and hence $t_p \leq t_z < t_q$. Since R_q^j is the first to violate the claim, we have that any such R_z^k returns a value $\geq n+1$. Any counter modification that follows such a read (R_z^k) must be an increment (see step 5), and we are done. □

6. Conclusions. We have defined the weak snapshot scan primitive and constructed an efficient implementation of it. We have given two examples of algorithms designed using the strong primitive of atomic snapshot scan for which it was possible to simply replace the expensive atomic snapshot with the much less expensive weak snapshot scan. Indeed, it seems that in many cases atomic snapshot scan simply can be replaced by weak snapshot scan. Our bounded construction relied on the **Traceable Use** abstraction of Dwork and Waarts [20]. Alternatively, we could have used the weaker primitives of Vitányi and Awerbuch [44], Tromp [43], Kirousis, Spirakis, and Tsigas [30], or Singh [41]. We also mentioned the implementation for the **Traceable Use** provided by [20]; another implementation for the **Traceable Use** was recently provided by Haldar [23].

In a similar spirit to the weak snapshot, one can define a weak concurrent timestamping system, which, roughly speaking, satisfies the properties of the standard timestamping system except that the ordering $\stackrel{ts}{\Longrightarrow}$ on **labeling** operations and the $\prec$ orders on labels are *partial* rather than total. Such a timestamping system is interesting for two reasons: it is conceptually simple and it can replace standard timestamping in at least one situation: Abrahamson's randomized consensus algorithm [1].

In conclusion, we can generalize our approach as follows. Consider a concurrent object with the following sequential specification:[12]

- *Mutator* operations modify the object's state but do not return any values. Mutator operations executed by different processes commute: applying them in either order leaves the object in the same state.
- *Observer* operations return some function of the object's state but do not modify the object.

A concurrent implementation of such an object is *linearizable* if the precedence order on operations can be extended to a total order $\Longrightarrow$ such that the value returned by each observer is the result of applying all the mutator operations ordered before it by $\Longrightarrow$. This kind of object has a straightforward wait-free linearizable implementation using atomic snapshot scan [6].[13] A *weakly linearizable* implementation is one that

[12] This definition is similar to Anderson's notion of a *pseudo read-modify-write* operation [4]. Anderson, however, requires that all mutators commute, not just those applied by different processes.

[13] This implementation uses unbounded space.

permits $\Longrightarrow$ to be a partial order instead of a total order. This paper's contribution is to observe that (1) weakly linearizable objects can be implemented more efficiently than any algorithm known for their fully linearizable counterparts, and (2) there are certain important applications where one can replace linearizable objects with weakly linearizable objects, preserving the application's modular structure while enhancing performance.

Acknowledgments. We would like to thank Jim Aspnes, Hagit Attiya, and Nir Shavit for helpful discussions.

REFERENCES

[1] K. Abrahamson, *On achieving consensus using a shared memory*, in Proc. 7th ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, ACM, New York, 1988, pp. 291–302.

[2] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *Atomic snapshots of shared memory*, J. ACM, 40 (1993), pp. 872–890.

[3] J. Anderson, *Composite registers*, Distrib. Comput., 6 (1987), pp. 141–154.

[4] J. Anderson and B. Groselj, *Beyond atomic registers: Bounded wait-free implementations of non-trivial objects*, Sci. Comput. Programming, 19 (1992), pp. 197–237.

[5] J. Aspnes, *Time- and space-efficient randomized consensus*, J. Algorithms, 14 (1993), pp. 414–431.

[6] J. Aspnes and M. P. Herlihy, *Wait-free data structures in the asynchronous PRAM model*, in Proc. 2nd Annual Symposium on Parallel Algorithms and Architectures, Crete, Greece, 1990, pp. 340–349.

[7] J. Aspnes and M. P. Herlihy, *Fast randomized consensus using shared memory*, J. Algorithms, 11 (1990), pp. 441–461.

[8] J. Aspnes and O. Waarts, *Randomized consensus in expected $O(N \log^2 N)$ operations*, SIAM J. Comput., 25 (1996), pp. 1024–1044.

[9] H. Attiya, D. Dolev, and N. Shavit, *Bounded polynomial randomized consensus*, in Proc. 8th ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, ACM, New York, 1989, pp. 281–294.

[10] H. Attiya, M. Herlihy, and O. Rachman, *Efficient atomic snapshots using lattice agreement*, in Proc. 6th International Workshop on Distributed Algorithms, Lecture Notes in Comput. Sci. 647, Springer-Verlag, New York, 1992, pp. 35–53.

[11] H. Attiya and O. Rachman, *Atomic snapshots in $O(n \log n)$ operations*, in Proc. 12th ACM Symposium on Principles of Distributed Computing, Ithaca, NY, ACM, New York, 1993, pp. 29–39.

[12] G. Bracha and O. Rachman, *Randomized consensus in expected $O(n^2 \log n)$ operations*, in Proc. 5th International Workshop on Distributed Algorithms, Delphi, Greece, 1991, Lecture Notes in Comput. Sci. 579, Springer-Verlag, Berlin, 1992, pp. 143–150.

[13] G. Bracha and O. Rachman, *Approximated Counters and Randomized Consensus*, Technical report 662, Computer Science Department, Israel Institute of Technology, Haifa, Israel, 1990.

[14] T. D. Chandra and C. Dwork, *Using Consensus to Solve Atomic Snapshots*, manuscript, August 1992.

[15] K. M. Chandy and L. Lamport, *Distributed snapshots: Determining global states of distributed systems*, ACM Trans. Comput. Systems, 3 (1985), pp. 63–75.

[16] B. Chor and C. Dwork, *Randomization in Byzantine agreement*, Adv. Comput. Res., 4 (1989), pp. 443–497.

[17] B. Chor, A. Israeli, and M. Li, *Wait–free consensus using asynchronous hardware*, SIAM J. Comput., 23 (1994), pp. 701–712.

[18] D. Dolev, C. Dwork, and L. Stockmeyer, *On the minimal synchronism needed for distributed consensus*, J. ACM, 34 (1987), pp. 77–97.

[19] D. Dolev and N. Shavit, *Bounded concurrent time-stamp systems are constructible!*, SIAM J. Comput.26 (1997), pp. 418–455

[20] C. Dwork and O. Waarts, *Simple and efficient bounded concurrent timestamping or bounded concurrent timestamp systems are comprehensible!*, in Proc. 24th ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 655–666.

[21] R. Gawlick, *Concurrent Timestamping Made Simple*, M.Sc. thesis, MIT, Cambridge, MA,

1992.

[22] R. Gawlick, N. Lynch, and N. Shavit, *Concurrent timestamping made simple*, in Proc. Israel Symposium on Theory of Computing and Systems, Haifa, Israel, Lecture Notes in Comput. Sci. 601, Springer-Verlag, Berlin, 1992, pp. 171–183.

[23] S. Haldar, *Efficient Bounded Timestamping Using Traceable Use Abstraction—Is Writer's Guessing Better than Reader's Telling?*, Technical report RUU-CS-93-28, Department of Computer Science, Utrecht University, Utrecht, The Netherlands, 1993.

[24] M. P. Herlihy, *Wait-free synchronization*, ACM Trans. Programming Languages and Systems, 13 (1991), pp. 124–149.

[25] M. P. Herlihy and J. M. Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Trans. Programming Languages and Systems, 12 (1990), pp. 463–492.

[26] A. Israeli and M. Li, *Bounded time stamps*, Distrib. Comput., 6 (1993), pp. 205–209.

[27] A. Israeli, M. Li, and P. M. B. Vitányi, *Simple Multireader Registers using Timestamp Systems*, Technical report CS-R8758, CWI, Amstedam, 1987.

[28] A. Israeli and M. Pinhasov, *A concurrent time-stamp scheme which is linear in time and space*, in Proc. 6th International Workshop on Distributed Algorithms, Lecture Notes in Comput. Sci. 647, Springer-Verlag, Berlin, New York, 1992, pp. 95–109.

[29] L. Kirousis, E. Kranakis, and P. M. B. Vitányi, *Atomic multireader register*, in Proc. 2nd International Workshop on Distributed Algorithms, Berlin, 1987, Lecture Notes in Comput. Sci. 312, Springer-Verlag, Berlin, New York, pp. 278–296.

[30] L. M. Kirousis, P. Spirakis, and P. Tsigas, *Reading many variables in one atomic operation: Solutions with linear or sublinear complexity*, IEEE Trans. on Parallel and Distributed Systems, 5 (1994), pp. 688–696.

[31] L. Lamport, *Concurrent reading and writing*, Commun. ACM, 20 (1977), pp. 806–811.

[32] L. Lamport, *On Interprocess communication, part* II*: Algorithms*, Distrib. Comput., 1 (1986), pp. 86–101.

[33] M. Li, J. Tromp, and P. M. B. Vitányi, *How to Share Concurrent Wait-free Variables*, JACM, 43 (1996), pp. 723–746.

[34] M. Li and P. M. B. Vitányi, *A Very Simple Construction for Atomic Multiwriter Register*, Technical report TR-8701, Aiken Computation Laboratory, Harvard University, Cambridge, MA, 1987.

[35] M. Li and P. M. B. Vitányi, *How to share concurrent asynchronous wait-free variables*, in Proc. ICALP (1989), Lecture Notes in Comput. Sci. 372, Springer-Verlag, Berlin, New York, 1989, pp. 488–505.

[36] M. C. Loui and H. H. Abu-Amara, *Memory requirements for agreement among unreliable asynchronous processes*, Adv. Comput. Res., 4 (1987), pp. 163–183.

[37] R. Newman-Wolfe, *A protocol for wait-free atomic multi-reader shared variables*, in Proc. 6th ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, ACM, New York, 1987, pp. 232–248.

[38] G. Peterson, *Concurrent reading while writing*, ACM Trans. Programming Languages and Systems, 5 (1983), pp. 46–55.

[39] M. Rabin, *Randomized Byzantine generals*, in Proc. 24th IEEE Symposium on Foundations of Computer Science, Tuscon, AZ, IEEE Press, Piscataway, NJ, 1983, pp. 403–409.

[40] M. Saks, N. Shavit, and H. Woll, *Optimal time randomized consensus—making resilient algorithms fast in practice*, in Proc. 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, ACM, New York, 1991, pp. 351–362.

[41] A. Singh, *Towards an understanding of unbounded variables in asynchronous systems*, Inform. Process. Lett., 42 (1992), pp. 7–17.

[42] A. Singh, J. Anderson, and M. Gauda, *The elusive atomic register revisited*, in Proc. 6th ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, ACM, New York, 1987, pp. 206–221.

[43] J. Tromp, *How to construct an atomic variable*, in Proc. 3rd International Workshop on Distributed Algorithms, Lecture Notes in Comput. Sci. 392, Springer-Verlag, New York, 1989, pp. 292–302.

[44] P. M. B. Vitányi and B. Awerbuch, *Atomic shared register access by asynchronous hardware*, in Proc. 27th IEEE Symposium on Foundations of Computer Science, Toronto, Ontario, IEEE Press, Piscataway, NJ, 1986, pp. 233–243.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1875–1905

THE CONSTRUCTION OF HUFFMAN CODES IS A SUBMODULAR ("CONVEX") OPTIMIZATION PROBLEM OVER A LATTICE OF BINARY TREES*

D. STOTT PARKER† AND PRASAD RAM‡

Abstract. We show that the space of all binary Huffman codes for a finite alphabet defines a *lattice*, ordered by the imbalance of the code trees. Representing code trees as path-length sequences, we show that the imbalance ordering is closely related to a majorization ordering on real-valued sequences that correspond to discrete probability density functions. Furthermore, this tree imbalance is a partial ordering that is consistent with the total orderings given by either the external path length (sum of tree path lengths) or the entropy determined by the tree structure. On the imbalance lattice, we show the weighted path-length of a tree (the usual objective function for Huffman coding) is a *submodular* function, as is the corresponding function on the majorization lattice. Submodular functions are discrete analogues of convex functions. These results give perspective on Huffman coding and suggest new approaches to coding as optimization over a lattice.

Key words. Huffman coding, adaptive coding, prefix codes, enumeration of trees, lattices, combinatorial optimization, convexity, submodular functions, entropy, tree imbalance, Schur convex functions, majorization, Moebius inversion, combinatorial inequalities, Fortuin–Kasteleyn–Ginibre (FKG) inequality, quadrangle inequality, Monge matrices, dynamic programming, greedy algorithms.

AMS subject classifications. 94A15, 94A24, 94A29, 94A45, 90C25, 90C27, 90C39, 90C48, 52A41, 68Q20, 68R05, 05A05, 05A20, 05C05, 05C30, 06A07, 26B25, 26D15

PII. S0097539796311077

1. Introduction.

The Huffman algorithm has been used heavily to produce efficient binary codes for almost half a century now. It has inspired a large literature with diverse theoretical and practical contributions. A comprehensive, very recent survey is [1]. Although the algorithm is quite elegant, it is tricky to prove correct and to reason about. While there may be little hope of improving on the $O(n \log n)$ complexity of the Huffman algorithm itself,[1] there is still room for improvement in our understanding of the algorithm.

There is also plenty of room for improvement in our understanding of variants of Huffman coding. Although the Huffman algorithm is remarkably robust in general and has widespread use, it is far from optimal in many real applications. Huffman coding is optimal only when the symbols to be coded are random and occur with fixed probabilities. Time-varying dependencies are not captured by the Huffman coding model, and optimal encoding of finite messages is not captured either.

Our motivation came from analysis of dynamic Huffman coding, a specific extension of Huffman coding in which the code used evolves over time. Recently, dynamic coding algorithms have been studied heavily. Our initial idea was to define "rebalancing" operations on code trees and to use these dynamically ("on the fly") in producing

*Received by the editors October 25, 1996; accepted for publication (in revised form) September 8, 1997; published electronically May 21, 1999.
http://www.siam.org/journals/sicomp/28-5/31107.html
†Computer Science Department, University of California, Los Angeles, CA 90095-1596 (stott@cs.ucla.edu).
‡Xerox Corporation, El Segundo, CA 90245 (Prasad.Ram@usa.xerox.com).
[1]The algorithm is closely related to sorting, in the sense that the sorted sequence of a sequence of integer values $\langle x_1 \cdots x_n \rangle$ is obtainable directly from the optimal code tree for the values $\langle 2^{x_1} \cdots 2^{x_n} \rangle$ (e.g., [26, p. 335]).

better codes, in situations where the distribution of symbols to be coded varies over time and/or is not accurately predictable in advance.

This paper reconstructs Huffman coding as an optimization over the space of binary trees. A natural representation for this space is sequences of ascending path-lengths, since this captures what is significant in producing optimal codes.

We show that the set of path-length sequences representing binary trees forms a lattice, which we call the *imbalance lattice*. This lattice orders trees by their imbalance and gives an organization for them that is useful in optimization. Our belief is that having a better mathematical (and not purely procedural) understanding of coding will ultimately pay off in improved algorithms.

The imbalance lattice and its imbalance ordering on trees depend on *majorization* in an essential way. Majorization is an important ordering on sequences that has many applications in pure and applied mathematics [27]. We have related it to greedy algorithms directly [33]. Earlier majorization was recognized as an important property of the internal node weights produced by the Huffman algorithm [13, 32], and in this work we go further to clarify its pervasive role.

By viewing the space of trees as a lattice, a variety of new theorems and algorithms become possible. For example, the objective functions commonly used in evaluating codes are *submodular* on this lattice. Submodular functions are closely related to convex functions (as we explain later; see Theorem 4.5) and are often easy to optimize [6, 9, 23, 24, 25]. Huffman coding gives a significant example of the importance of submodularity in algorithms.

2. Ordered sequences, rooted binary trees, and Huffman codes.

2.1. Ordered sequences. By a *sequence* we mean an ordered collection of nonnegative real values such as

$$\mathbf{x} \quad = \quad \langle\, x_1\ x_2\ \cdots\ x_n\,\rangle.$$

Repetition of values in the sequence is permitted: the values x_j need not be distinct. The *length* of this sequence is n, and for simplicity we also refer to the set of such sequences with the vector notation ${\Re_+}^n$.

We introduce several useful operators on sequences:

ascending sort	$\text{sort}\uparrow(\mathbf{x})$	$=$	$\langle\ \mathbf{x}$ put in ascending order $\rangle$,
descending sort	$\text{sort}\downarrow(\mathbf{x})$	$=$	$\langle\ \mathbf{x}$ put in descending order $\rangle$,
sequence exponential	$2^{-\mathbf{x}}$	$=$	$\langle\ 2^{-x_1}\ \cdots\ 2^{-x_n}\ \rangle$,
sequence logarithm	$-\log_2(\mathbf{x})$	$=$	$\langle\ -\log_2(x_1)\ \cdots\ -\log_2(x_n)\ \rangle$.

A *density sequence* is a nonnegative real-valued sequence whose entries sum to 1.

A *distribution sequence* is an ascending nonnegative sequence whose final entry is 1.

For simplicity, throughout this paper many sequences are implicitly sorted:

$\boldsymbol{\ell}$, $\mathbf{s}$, $\mathbf{t}$, $\mathbf{u}$	denote ascending sequences of positive integer values whose sequence exponentials $2^{-\ell}$, $2^{-\mathbf{s}}$, $2^{-\mathbf{t}}$ are density sequences.
$\mathbf{w}$	denotes a descending sequence of positive real values.
$\mathbf{v}$	denotes an ascending distribution sequence.
$\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$	denote descending density sequences.

Note since $\boldsymbol{\ell}$ is ascending, $2^{-\boldsymbol{\ell}}$ is descending; and since $\mathbf{x}$ is descending, $-\log_2(\mathbf{x})$ is ascending.

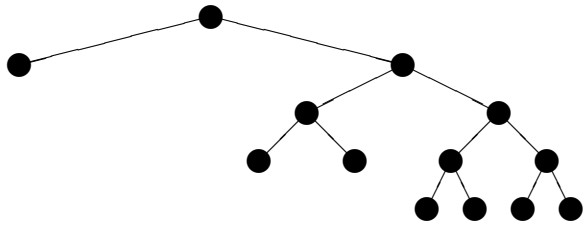


FIG. 2.1. *A binary tree having path-length sequence* $\langle\ 1\ 3\ 3\ 4\ 4\ 4\ 4\ \rangle$.

We also allow sequences to be operated upon as vectors. Thus, if $\mathbf{x}$ is a sequence (vector) of length n and A is an $n \times n$ matrix, then $A\,\mathbf{x}$ is a sequence (vector). Treating sequences as vectors allows us to define several useful operators using matrix algebra.

2.2. Rooted binary trees and path-length sequences. Rooted binary trees here are binary trees with a root node, in which every node is either a leaf node or an internal node having one parent and two children. The order of the leaves is insignificant, so a given tree is determined (up to permutation of the leaves) by the lengths of the paths from the root node to each leaf node (the distance of the leaf from the root). Thus we can represent equivalence classes of the rooted binary trees with n leaves by sequences of n nonnegative integers, which give the path-length of each leaf. For example, the path-length sequence

$$\langle\ 1\ 3\ 3\ 4\ 4\ 4\ 4\ \rangle$$

represents a binary tree with $n = 7$ leaves, of which one has path-length 1, two have path-length 3, and four have path-length 4; it is shown in Figure 2.1.

Path-length sequences obey what we call the *Kraft equality*, a special case of the Kraft inequality of noiseless coding theory (see, e.g., [10, p. 45]).

THEOREM 2.1. *For all* $n \geq 1$, $\langle\ \ell_1\ \cdots\ \ell_n\ \rangle$ *is the sequence of path-lengths in a rooted binary tree iff*

$$\sum_{i=1}^{n} 2^{-\ell_i} \quad = \quad 1.$$

Thus $\boldsymbol{\ell}$ *is a path-length sequence iff* $2^{-\boldsymbol{\ell}}$ *is a density sequence.*

Proof. The theorem is easily proven by induction on n. For the basis, with $n = 1$ we must have $\ell_1 = 0$. The induction step follows by noticing that the two principal subtrees of any binary tree must have sequences $\langle \ell'_1\ \cdots\ \ell'_p \rangle$ and $\langle \ell''_1\ \cdots\ \ell''_q \rangle$ satisfying the equality and that their composition has the sequence $\langle\ (\ell'_1+1)\ \cdots\ (\ell'_p+1)(\ell''_1+1)\ \cdots\ (\ell''_q+1)\ \rangle$, which again satisfies the equality. □

Henceforth we assume that tree path-length sequences are in ascending sorted order. Table 2.1 shows a lexicographic tabulation of all possible sequences for $1 \leq n \leq 7$, along with T_n, the total number of inequivalent sequences of length n. T_n is enumerated as sequence M0710 (A002572) in [39]. An upper bound on T_n can be obtained from the Catalan number C_n, which computes the number of unordered binary trees: for $n \geq 3$, $T_n \leq \frac{1}{2}C_n \leq 2^{n-3}$. Gilbert [12], using the notation $g(N)$ for T_N, points out that T_n is well approximated for $n \leq 30$ by

$$T_n \quad \simeq \quad 0.148\ (1.791)^n.$$

TABLE 2.1
Path-length sequences for small values of n.

n	1	2	3	4	5	6	7	8
T_n	1	1	1	2	3	5	9	16
	⟨ 0 ⟩	⟨ 1 1 ⟩	⟨ 1 2 2 ⟩	⟨ 1 2 3 3 ⟩ ⟨ 2 2 2 2 ⟩	⟨ 1 2 3 4 4 ⟩ ⟨ 1 3 3 3 3 ⟩ ⟨ 2 2 2 3 3 ⟩	⟨ 1 2 3 4 5 5 ⟩ ⟨ 1 2 4 4 4 4 ⟩ ⟨ 1 3 3 3 4 4 ⟩ ⟨ 2 2 2 3 4 4 ⟩ ⟨ 2 2 3 3 3 3 ⟩	⟨ 1 2 3 4 5 6 6 ⟩ ⟨ 1 2 3 5 5 5 5 ⟩ ⟨ 1 2 4 4 4 5 5 ⟩ ⟨ 1 3 3 3 4 5 5 ⟩ ⟨ 1 3 3 4 4 4 4 ⟩ ⟨ 2 2 2 3 4 5 5 ⟩ ⟨ 2 2 2 4 4 4 4 ⟩ ⟨ 2 2 3 3 3 4 4 ⟩ ⟨ 2 3 3 3 3 3 3 ⟩	⋮

TABLE 2.2
Path-length sequences ℓ and their weighted path-length $g_{\mathbf{w}}(\ell)$ for $\mathbf{w} = \langle$ 189 95 73 71 28 23 21 $\rangle$.

ℓ	$g_{\mathbf{w}}(\ell)$
⟨ 1 2 3 4 5 6 6 ⟩	1286
⟨ 1 2 3 5 5 5 5 ⟩	1313
⟨ 1 2 4 4 4 5 5 ⟩	1287
⟨ 1 3 3 3 4 5 5 ⟩	1238
⟨ 1 3 3 4 4 4 4 ⟩	1265
⟨ 2 2 2 3 4 5 5 ⟩	1259
⟨ 2 2 2 4 4 4 4 ⟩	1286
⟨ 2 2 3 3 3 4 4 ⟩	1260
⟨ 2 3 3 3 3 3 3 ⟩	1311

2.3. Huffman codes are optimal path-length sequences. A Huffman code for a given positive weight sequence

$$w_1 \;\geq\; w_2 \;\geq\; \cdots \;\geq\; w_n$$

consists of a binary tree, i.e., a path-length sequence $\boldsymbol{\ell} = \langle\, \ell_1\, \ell_2\, \cdots\, \ell_n \,\rangle$, which we evidently want to be in ascending order,

$$\ell_1 \;\leq\; \ell_2 \;\leq\; \cdots \;\leq\; \ell_n,$$

so that the weighted path-length

$$g_{\mathbf{w}}(\boldsymbol{\ell}) \quad = \quad \sum_{i=1}^{n} w_i\, \ell_i$$

is minimal. Beyond the Kraft equality of Theorem 2.1, it is difficult to characterize what it is that makes $\boldsymbol{\ell}$ optimal. For example, Table 2.2 shows all feasible codes and costs for the weight sequence $\mathbf{w} = \langle$ 189 95 73 71 28 23 21 $\rangle$, with $n = 7$.

Huffman's breakthrough [18] was to identify an efficient algorithm that finds an optimal tree, avoiding a search over the exponentially large space of trees. The algorithm repeatedly combines the two tree leaves with least weight, whose sum becomes the weight of a new leaf. The Huffman (optimal) tree in Table 2.2 has path lengths $\boldsymbol{\ell} = \langle$ 1 3 3 3 4 5 5 $\rangle$ and total weighted path-length 1238. The Huffman algorithm reflects a divide-and-conquer structure that has interesting properties on the space of trees, but because of its procedural nature does little to characterize optimal trees.

3. The imbalance lattice of binary trees. The optimality of a Huffman code is determined by the match between the balance (or imbalance) between the code tree and the weights of the symbols to be coded. In this section we show ternary balancing exchanges give an imbalance ordering on binary trees that defines a lattice.

The idea of using lattices in coding dates back at least to Shannon in 1950 [38]. However, we have not found the lattice characterization of tree imbalance elsewhere. Following considerable work in the early 1980s on enumeration of trees, Pallo classified trees by their *rotational* structure (e.g., [30, 31]) and showed that they then form a lattice. Our work differs from Pallo's in that we classify trees by their *path-length* (imbalance) structure.

3.1. Important properties of tree path-length sequences.

THEOREM 3.1. *Every path-length sequence* $\boldsymbol{\ell}$ *has the form*

$$\boldsymbol{\ell} \quad = \quad \langle \ \cdots \ (q-j) \ \overbrace{q \ \cdots \ q}^{2k} \ \rangle,$$

a sequence including $2k$ *copies of its largest value* q *(where* $j, k > 0$*). Also,* j *is at most the largest exponent of* 2 *in* $2k$*, and therefore* $j \leq \log_2(2k)$.

Proof. $\boldsymbol{\ell}$ must include $2k$ copies of its largest value q since otherwise $\left(2^q \cdot \sum_{i=1}^{n} 2^{-\ell_i}\right)$ is odd, contradicting the Kraft equality. Using this argument again on the shorter path-length sequence obtained by replacing the $2k$ copies of q with k copies of $(q-1)$, the Kraft equality requires not only that $j > 0$ but also that j be at most the number of times that 2 divides $2k$. ∎

THEOREM 3.2. *Except for the sequence* $\langle\ 1\ 2\ 3\ \ldots\ (n-2)\ (n-1)\ (n-1)\ \rangle$*, any path-length sequence contains at least three identical values.*

Proof. The proof is by induction on the length n of the sequence. For the basis, when $n = 3$ the only sequence is $\langle\ 1\ 2\ 2\ \rangle$, satisfying the theorem. For the induction step, suppose $n > 3$, and to the contrary of the theorem that there is a sequence does not have three identical values. Let q be the smallest value in the sequence appearing twice. We may assume $q < (n-1)$, since otherwise the sequence is $\langle\ 1\ 2\ 3\ \ldots\ (n-2)\ (n-1)\ (n-1)\ \rangle$. Construct the sequence of length $n-1$ that results from replacing the two values q with one value $(q-1)$. In this new sequence, q does not appear at all (since there were only two before), and $(q-1)$ appears at most twice. Therefore, by induction, since this sequence does not have three identical values it is $\langle\ 1\ 2\ 3\ \ldots\ (n-3)\ (n-2)\ (n-2)\ \rangle$. But since $q < (n-1)$ and q does not appear in the new sequence, this gives a contradiction. ∎

3.2. Ternary exchanges determine tree imbalance. The insight that inspired us to write this paper is that it is possible to generate all binary tree path-length sequences using ternary exchanges. Given any path-length sequence

$$\langle \cdots \qquad p \qquad \cdots\cdots\cdots \quad (q+1) \quad (q+1) \quad \cdots \rangle,$$

then the revision

$$\langle \cdots \quad (p+1) \quad (p+1) \quad \cdots\cdots\cdots \qquad q \qquad \cdots \rangle$$

is a path-length sequence also, because

$$2^{-p} + 2^{-(q+1)} + 2^{-(q+1)} \quad = \quad 2^{-p} + 2^{-q} \quad = \quad 2^{-(p+1)} + 2^{-(p+1)} + 2^{-q}.$$

Moreover, if the initial sequence is sorted in ascending order (so $p \leq q$) and we replace the rightmost p and leftmost two $(q+1)$s, then the resulting sequence is still sorted.

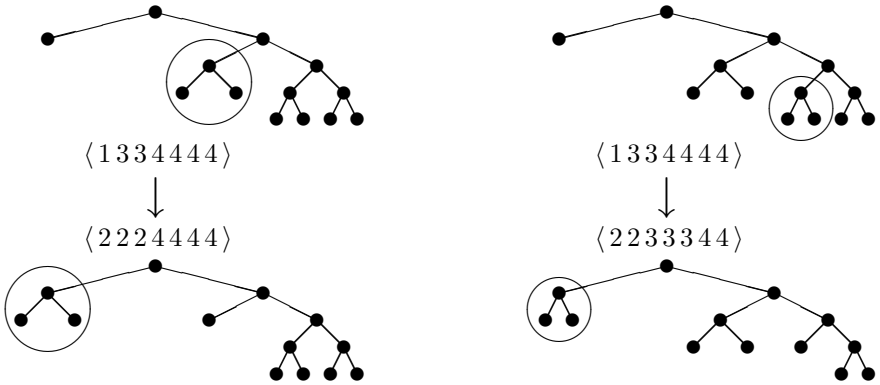


FIG. 3.1. *Balancing exchanges:* $\langle 1\,3\,3\,4\,4\,4\,4\rangle \rightarrow \langle 2\,2\,2\,4\,4\,4\,4\rangle$ *and* $\langle 1\,3\,3\,4\,4\,4\,4\rangle \rightarrow \langle 2\,2\,3\,3\,3\,4\,4\rangle$.

(When $p = q$ the two sequences are identical.) Dually, this exchange can be applied in reverse; with sorted sequences, if we replace the leftmost two $(p+1)$s and the rightmost $(q-1)$, the result will still be sorted in ascending order.

The net effect of this exchange is to transfer two leaves dangling from level q to level p. The two examples in Figure 3.1 show this pictorially.

DEFINITION 3.3. *Let p, q be integers such that $1 \le p < q < n$. A* balancing exchange *is a ternary exchange of the form*

$$\begin{array}{lccccccc} \langle\cdots & p & & \cdots\cdots & & (q+1) & (q+1) & \cdots\rangle \\ & \downarrow & \searrow & & \cdots & & \searrow & \downarrow \\ \langle\cdots & (p+1) & (p+1) & & & \cdots\cdots & q & \cdots\rangle. \end{array}$$

It is called a minimal balancing exchange *if $(p+1) = q$. An* imbalancing exchange *is of the reverse form,*

$$\begin{array}{lccccccc} \langle\cdots & (p+1) & (p+1) & & \cdots\cdots & & q & \cdots\rangle \\ & \downarrow & \swarrow & & \cdots & \swarrow & \downarrow & \\ \langle\cdots & p & & \cdots\cdots & & (q+1) & (q+1) & \cdots\rangle. \end{array}$$

Finally, we can define partial orders as the reflexive transitive closures of these relations among sequences. Given two sequences $\mathbf{s}$ *and* $\mathbf{t}$*, we say that* $\mathbf{s}$ *is at least as balanced as* $\mathbf{t}$*,*

$$\mathbf{s} \quad \trianglelefteq \quad \mathbf{t},$$

if there are sequences $\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_m$ $(m \ge 1)$ where $\mathbf{t} = \boldsymbol{\ell}_1$, $\boldsymbol{\ell}_m = \mathbf{s}$, and for each i, $1 \le i < m$, there is a balancing exchange from $\boldsymbol{\ell}_i$ to $\boldsymbol{\ell}_{i+1}$.

Minimal balancing exchanges, in which $(p+1) = q$, are particularly significant. The balancing exchange $\langle\, 1\,3\,3\,4\,4\,4\,4\,\rangle \rightarrow \langle\, 2\,2\,2\,4\,4\,4\,4\,\rangle$ in Figure 3.1 gives an example. Minimal balancing exchanges are *ternary exchanges of consecutive length values*, so any tree path-length sequence of the form $\langle\cdots\ (q-1)\ \cdots\ (q+1)\ (q+1)\ \cdots\rangle$ determines the more balanced tree path-length sequence $\langle\cdots\ q\ \cdots\ q\ q\ \cdots\rangle$ and vice versa.

THEOREM 3.4. *If two path-length sequences differ, they differ in at least three values. Also, if they differ in exactly three values, there is a ternary exchange between the sequences.*

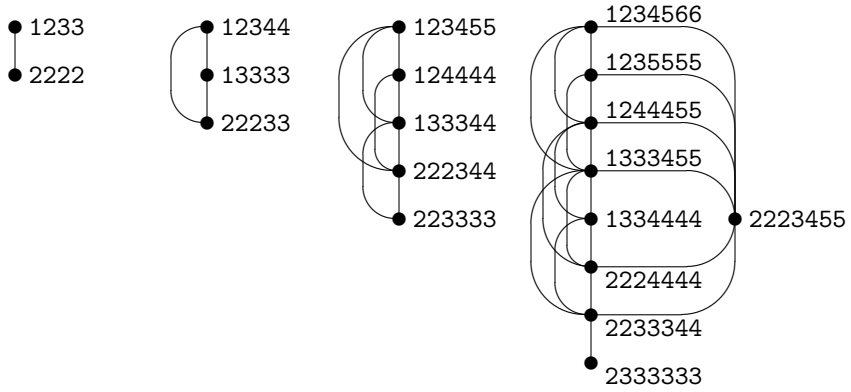


FIG. 3.2. *The path-length imbalance ordering for $n = 4,\ 5,\ 6,\ 7$; edges denote ternary exchanges.*

Proof. Direct consequence of the Kraft equality. The equality shows that two path-length sequences cannot differ in one value. Similarly, there cannot be sequences $\mathbf{s}$ and $\mathbf{t}$ differing in two values, since if the differences were the disjoint sequences of positive integers $\langle\ s_i\ s_j\ \rangle$ and $\langle\ t_i\ t_j\ \rangle$, then the Kraft equality would imply $2^{-s_i} + 2^{-s_j} = 2^{-t_i} + 2^{-t_j}$, which is false under the disjointness condition. Finally, sequences differing in three integer values $\langle\ s_i\ s_j\ s_k\ \rangle$ and $\langle\ t_i\ t_j\ t_k\ \rangle$ must satisfy $2^{-s_i} + 2^{-s_j} + 2^{-s_k} = 2^{-t_i} + 2^{-t_j} + 2^{-t_k}$, and a case analysis shows that this is solved only by $\langle\ s_i\ s_j\ s_k\ \rangle = \langle\ p\ (q{+}1)\ (q{+}1)\ \rangle$ and $\langle\ t_i\ t_j\ t_k\ \rangle = \langle\ (p{+}1)\ (p{+}1)\ q\ \rangle$, corresponding to a ternary exchange. ∎

THEOREM 3.5. *The path-length imbalance ordering is a partial order.*

Proof. It is reflexive and transitive by construction. Also the imbalance ordering is antisymmetric, since $\mathbf{s} \trianglelefteq \mathbf{t}$ and $\mathbf{t} \trianglelefteq \mathbf{s}$ together imply $\mathbf{s} = \mathbf{t}$. Otherwise there would be a sequence of balancing exchanges that transform $\mathbf{t}$ to $\mathbf{s}$ and ultimately back to $\mathbf{t}$; this is not possible, since each balancing exchange reduces by at least one the sum of the values in the sequence. ∎

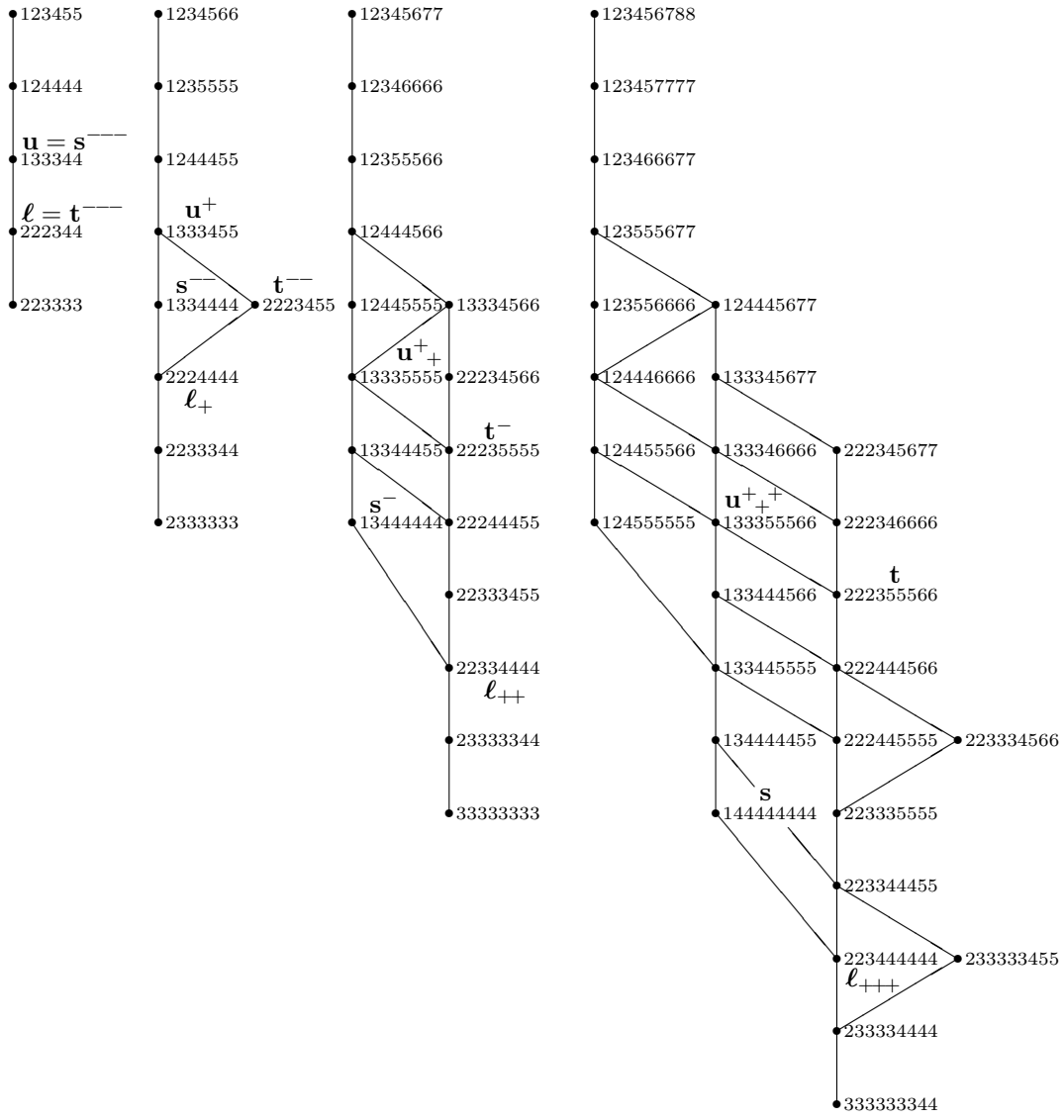
The imbalance partial order is straightforward to derive for small values of n. In Figure 3.2, it is displayed for $n = 4, 5, 6, 7$. The most imbalanced sequence appears at the top of the partial order, and an edge from a sequence $\mathbf{s}$ down to another $\mathbf{t}$ means that a balancing exchange is possible from $\mathbf{s}$ to $\mathbf{t}$. It is evident from Figure 3.2 that the minimal exchanges define the bulk of the ordering. In order to provide a deeper appreciation for its structure, Figure 3.3 presents the ordering for $n = 6, 7, 8, 9$. Figures 3.2 and 3.3 suggest a number of results about the imbalance ordering.

THEOREM 3.6. *A sequence is on level k of the imbalance partial order (counting from 0, the topmost and least balanced level)* iff *k minimal balancing exchanges are needed to derive it from the least balanced sequence $\langle\ 1\ 2\ 3\ \cdots\ (n{-}2)\ (n{-}1)\ (n{-}1)\ \rangle$. In this situation the sum of the values in the sequence is*

$$\frac{(n+2)(n-1)}{2} \quad - \quad k.$$

Thus the level of a sequence in the partial order is determined by the sum of its path-length values.

Proof. By induction on k. For the basis $k = 0$, the sum of the path-lengths in the least balanced sequence is $(\sum_{i=1}^{n-1} i) + (n{-}1) = (n+2)(n-1)/2$. For the induction step, consider a sequence whose sum of values is $\frac{(n+2)(n-1)}{2} - k$ with $k > 0$. By



s	t	s ∨ t	s ∧ t
⟨ 1 4 4 4 4 4 4 4 4 4 ⟩	⟨ 2 2 2 3 5 5 5 6 6 ⟩	⟨ 1 3 3 3 5 5 5 6 6 ⟩	⟨ 2 2 3 4 4 4 4 4 4 ⟩
⟨ 1 4 4 4 4 4 4 4 4 ⟩	⟨ 2 2 3 3 3 4 5 6 6 ⟩	⟨ 1 3 3 4 4 4 5 6 6 ⟩	⟨ 2 2 3 4 4 4 4 4 4 ⟩
⟨ 1 3 4 4 4 4 4 4 5 5 ⟩	⟨ 2 2 3 3 3 3 4 5 6 6 ⟩	⟨ 1 3 3 4 4 4 5 6 6 ⟩	⟨ 2 2 3 3 4 4 4 5 5 ⟩
⟨ 1 2 4 5 5 5 5 5 5 ⟩	⟨ 2 2 3 3 3 3 4 5 6 6 ⟩	⟨ 1 2 4 4 5 5 5 6 6 ⟩	⟨ 2 2 3 3 3 3 5 5 5 5 ⟩
⟨ 1 2 4 5 5 5 5 5 5 ⟩	⟨ 2 2 2 3 4 5 6 7 7 ⟩	⟨ 1 2 4 4 4 5 6 7 7 ⟩	⟨ 2 2 2 4 4 5 5 5 5 ⟩
⟨ 1 2 4 4 5 5 5 6 6 ⟩	⟨ 2 2 2 3 4 5 6 7 7 ⟩	⟨ 1 2 4 4 4 5 6 7 7 ⟩	⟨ 2 2 2 3 5 5 5 6 6 ⟩

FIG. 3.3. *The imbalance lattice, showing path-length sequences ordered by imbalance. The sequence* ⟨ 1 2 3 4 ⋯ ⟩ *is maximally imbalanced. The graphs display the (transitively reduced) path-length imbalance ordering for* $n = 6, 7, 8, 9$. *For clarity, only a minimal subset of the imbalance ordering is drawn; orderings in the transitive closure of the minimal set are omitted. The imbalance ordering is also a lattice, with well-defined upper bounds* $\mathbf{s} \vee \mathbf{t}$ *and lower bounds* $\mathbf{s} \wedge \mathbf{t}$ *for every pair of trees* $\mathbf{s}$ *and* $\mathbf{t}$. *Some trees are marked to clarify certain notions (contractions, lower expansions, and upper expansions), and their use in derivation of the first entry in the table of representative upper and lower bounds for* $n = 9$.

Theorem 3.2, this sequence must contain at least three identical values $\langle\, q\, q\, q \,\rangle$. Thus there is a minimal balancing exchange to this sequence from another that contains $\langle\, (q-1)\, (q+1)\, (q+1) \,\rangle$. This sequence is at level $k-1$ by construction, and by induction it has the stated sum. □

Theorem 3.6 shows the significance of the level of a sequence in the imbalance partial order.

DEFINITION 3.7. *The* level of balance *of a path-length sequence* $\mathbf{s}$ *is*

$$\frac{(n+2)(n-1)}{2} \quad - \quad (\textit{sum of the path-length values in } \mathbf{s}).$$

3.3. Contractions and expansions of path-length sequences.

DEFINITION 3.8. *Let* $\boldsymbol{\ell} = \langle\, \ell_1 \cdots \ell_n \,\rangle$ *be a tree path-length sequence of length* n. *The* contraction $\boldsymbol{\ell}^-$ *of* $\boldsymbol{\ell}$ *is the sequence of length* $(n-1)$ *defined by*

$$\boldsymbol{\ell}^- \quad = \quad \text{sort}\uparrow (\, \langle\, \ell_1 \ \cdots \ \ell_{n-2} \ \ (\ell_{n-1} - 1) \,\rangle \,).$$

The position i expansion of $\boldsymbol{\ell}$ *is the sequence of length* $(n+1)$ *defined by*

$$\text{sort}\uparrow (\, \langle\, \ell_1 \ \cdots \ \ell_{i-1} \ (\ell_i+1) \ (\ell_i+1) \ \ell_{i+1} \ \cdots \ \ell_n \,\rangle \,).$$

As permitted by Theorem 3.1, *if we write*

$$\boldsymbol{\ell} \quad = \quad \langle \ \cdots \ (q-j) \ \overbrace{q \ \cdots \ q}^{2k} \ \rangle$$

with j, $k > 0$, *then* $2k$ *is the* suffix length *of* $\boldsymbol{\ell}$, *and* j *is the* suffix increment *of* $\boldsymbol{\ell}$. *The* contraction $\boldsymbol{\ell}^-$ *is then*

$$\boldsymbol{\ell}^- \quad = \quad \langle \ \cdots \ (q-j) \ \ (q-1) \ \overbrace{q \ \cdots \ q}^{2k-2} \ \rangle.$$

The lower expansion $\boldsymbol{\ell}_+$ *is the position* $n - 2k$ *expansion of* $\boldsymbol{\ell}$:

$$\boldsymbol{\ell}_+ \quad = \quad \langle \ \cdots \ (q-j+1) \ \ (q-j+1) \ \overbrace{q \ \cdots \ q}^{2k} \ \rangle.$$

The upper expansion $\boldsymbol{\ell}^+$ *is the position* n *expansion of* $\boldsymbol{\ell}$:

$$\boldsymbol{\ell}^+ \quad = \quad \langle \ \cdots \ (q-j) \ \overbrace{q \ \cdots \ q}^{2k-1} \ \ (q+1) \ \ (q+1) \ \rangle.$$

Note the definition for $\boldsymbol{\ell}_+$ assumes $2k < n$. When $2k = n$, requiring n to be a power of 2 and $\boldsymbol{\ell} = \langle\, q \ \cdots \ q \,\rangle$, where $q = \log_2(n)$, the formula above does not define $\boldsymbol{\ell}_+$. In this very special case we define $\boldsymbol{\ell}_+ = \boldsymbol{\ell}^+$ rather than leave $\boldsymbol{\ell}_+$ undefined.

These definitions will be used heavily throughout the rest of the paper. Figure 3.4 and Table 3.1 give examples for $n = 7$. Figure 3.3 also gives examples illustrating the relationships these definitions produce among the imbalance orderings for successive values of n.

THEOREM 3.9. *If* $\boldsymbol{\ell}$ *is a path-length sequence,* $\boldsymbol{\ell}_+ \trianglelefteq \boldsymbol{\ell}^+$ *and* $(\boldsymbol{\ell}_+)^- \trianglelefteq (\boldsymbol{\ell}^+)^- = \boldsymbol{\ell}$. *Furthermore, either* $\boldsymbol{\ell} = (\boldsymbol{\ell}^-)_+$, *or* $\boldsymbol{\ell} = (\boldsymbol{\ell}^-)^+$. *Thus* $(\boldsymbol{\ell}^-)_+ \trianglelefteq \boldsymbol{\ell} \trianglelefteq (\boldsymbol{\ell}^-)^+$.

TABLE 3.1

Path-length sequences of length 7*, with their contractions and expansions. Note that all contractions have length* 6*, and expansions length* 8*. Emboldened digits reflect changes from* $\boldsymbol{\ell}$*.*

path-length sequence	*suffix length*	*suffix incr.*	*contraction*	*lower expansion*	*upper expansion*
$\boldsymbol{\ell}$	$2k$	j	$\boldsymbol{\ell}^-$	$\boldsymbol{\ell}_+$	$\boldsymbol{\ell}^+$
⟨ 1 2 3 4 5 6 6 ⟩	2	1	⟨ 1 2 3 4 5 **5** ⟩	⟨ 1 2 3 4 **6 6** 6 6 ⟩	⟨ 1 2 3 4 5 6 **7 7** ⟩
⟨ 1 2 4 4 4 5 5 ⟩	2	1	⟨ 1 2 4 4 4 **4** ⟩	⟨ 1 2 4 4 **5 5** 5 5 ⟩	⟨ 1 2 4 4 4 5 **6 6** ⟩
⟨ 1 2 3 5 5 5 5 ⟩	4	2	⟨ 1 2 3 **4** 5 5 ⟩	⟨ 1 2 **4 4** 5 5 5 5 ⟩	⟨ 1 2 3 5 5 5 **6 6** ⟩
⟨ 1 3 3 3 4 5 5 ⟩	2	1	⟨ 1 3 3 3 4 **4** ⟩	⟨ 1 3 3 3 **5 5** 5 5 ⟩	⟨ 1 3 3 3 4 5 **6 6** ⟩
⟨ 1 3 3 4 4 4 4 ⟩	4	1	⟨ 1 3 3 **3** 4 4 ⟩	⟨ 1 3 **4 4** 4 4 4 4 ⟩	⟨ 1 3 3 4 4 4 **5 5** ⟩
⟨ 2 2 2 3 4 5 5 ⟩	2	1	⟨ 2 2 2 3 4 **4** ⟩	⟨ 2 2 2 3 **5 5** 5 5 ⟩	⟨ 2 2 2 3 4 5 **6 6** ⟩
⟨ 2 2 2 4 4 4 4 ⟩	4	2	⟨ 2 2 2 **3** 4 4 ⟩	⟨ 2 2 **3 3** 4 4 4 4 ⟩	⟨ 2 2 2 4 4 4 **5 5** ⟩
⟨ 2 2 3 3 3 4 4 ⟩	2	1	⟨ 2 2 3 3 3 **3** ⟩	⟨ 2 2 3 3 **4 4** 4 4 ⟩	⟨ 2 2 3 3 3 4 **5 5** ⟩
⟨ 2 3 3 3 3 3 3 ⟩	6	1	⟨ 2 **2** 3 3 3 3 ⟩	⟨ **3 3** 3 3 3 3 3 3 ⟩	⟨ 2 3 3 3 3 3 **4 4** ⟩

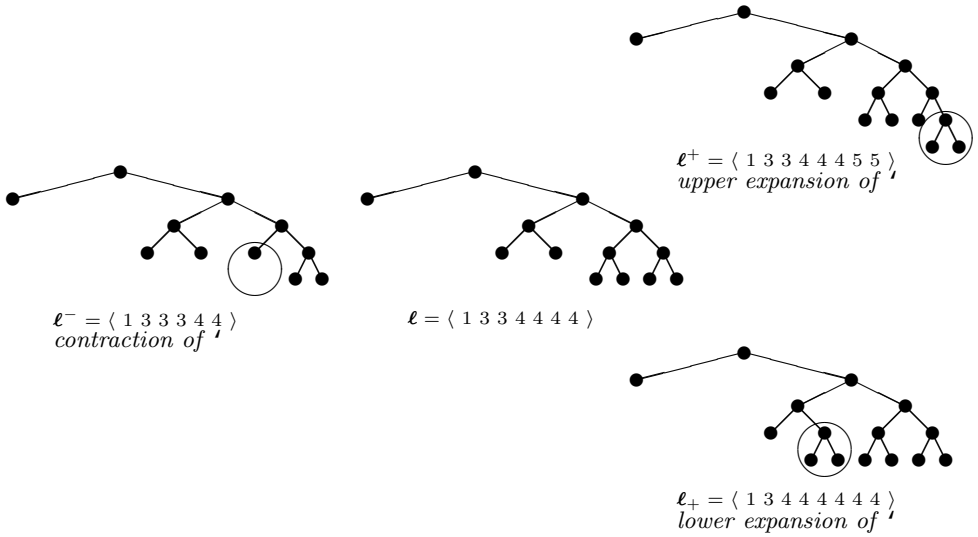


FIG. 3.4. *The contraction and expansions of the path-length sequence* $\boldsymbol{\ell} = \langle\ 1\ 3\ 3\ 4\ 4\ 4\ 4\ \rangle$.

Proof. $\boldsymbol{\ell}_+$ and $\boldsymbol{\ell}^+$ differ by a ternary exchange, so $\boldsymbol{\ell}_+ \trianglelefteq \boldsymbol{\ell}^+$. From Theorem 3.1 we can assume

$$\boldsymbol{\ell} \;=\; \langle \cdots (q{-}j)\, \overbrace{q \cdots q}^{2k} \rangle,$$

and thus $(\boldsymbol{\ell}^+)^- = \boldsymbol{\ell}$. Furthermore $(\boldsymbol{\ell}_+)^- = \boldsymbol{\ell}$ if $j = 1$ and

$$(\boldsymbol{\ell}_+)^- \;=\; \langle \cdots (q{-}j{+}1)(q{-}j{+}1)(q{-}1)\, \overbrace{q \cdots q}^{2k-2} \rangle \trianglelefteq \boldsymbol{\ell}$$

if $j > 1$. Finally $(\boldsymbol{\ell}^-)^+ = \boldsymbol{\ell}$ if $k = 1$ (necessitating $j = 1$) and $(\boldsymbol{\ell}^-)_+ = \boldsymbol{\ell}$ if $k > 1$. Consequently $\boldsymbol{\ell} \in \{\, (\boldsymbol{\ell}^-)_+,\ (\boldsymbol{\ell}^-)^+ \,\}$. □

THEOREM 3.10. *If* $\mathbf{s} \trianglelefteq \mathbf{t}$, *then* $\mathbf{s}^- \trianglelefteq \mathbf{t}^-$, $\mathbf{s}_+ \trianglelefteq \mathbf{t}_+$, *and* $\mathbf{s}^+ \trianglelefteq \mathbf{t}^+$.

Proof. Recall that if $\mathbf{s} \trianglelefteq \mathbf{t}$, then there are sequences $\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_m$ $(m \geq 1)$ such that $\mathbf{t} = \boldsymbol{\ell}_1$, $\boldsymbol{\ell}_m = \mathbf{s}$ and for each i, $1 \leq i < m$, there is a balancing exchange from

$\boldsymbol{\ell}_i$ to $\boldsymbol{\ell}_{i+1}$. Our approach here is very simple: to prove $\mathbf{s}^- \trianglelefteq \mathbf{t}^-$ we convert the derivation $\mathbf{t} = \boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_m = \mathbf{s}$ directly to the derivation $\mathbf{t}^- = \boldsymbol{\ell}_1{}^-, \ldots, \boldsymbol{\ell}_m{}^- = \mathbf{s}^-$. For this it is sufficient to show that either each step from $(\boldsymbol{\ell}_i)^-$ to $(\boldsymbol{\ell}_{i+1})^-$ is a balancing exchange, or $(\boldsymbol{\ell}_i)^- = (\boldsymbol{\ell}_{i+1})^-$. The former must hold if $\boldsymbol{\ell}_i$ and $\boldsymbol{\ell}_{i+1}$ agree in the final two positions. If they disagree,

$$\begin{array}{rcccccccccc} \boldsymbol{\ell}_i & = & \langle & \cdots & & p & & a & \cdots & b & q \quad q & \rangle, \\ \boldsymbol{\ell}_{i+1} & = & \langle & \cdots & (p{+}1) & (p{+}1) & & a & \cdots & b & (q{-}1) & \rangle \end{array}$$

because they define a balancing exchange, and by Theorem 3.1 necessarily $b = (q{-}1)$. Then

$$\begin{array}{rcccccccccc} (\boldsymbol{\ell}_i)^- & = & \mathrm{sort}{\uparrow}(\langle & \cdots & & p & & a & \cdots & (q{-}1) \quad (q{-}1) & \rangle), \\ (\boldsymbol{\ell}_{i+1})^- & = & \mathrm{sort}{\uparrow}(\langle & \cdots & (p{+}1) & (p{+}1) & & a & \cdots & (q{-}2) & \rangle). \end{array}$$

If $(p{+}1) = b = (q{-}1)$, then $p = (q{-}2)$ and the two contractions are equal. If not, they still differ by a balancing exchange. Proving $\mathbf{s}_+ \trianglelefteq \mathbf{t}_+$ is similar, where $(\boldsymbol{\ell}_i)_+ = (\boldsymbol{\ell}_{i+1})_+$ iff $\boldsymbol{\ell}_i = \langle \cdots (q{-}j)\ q\ q \cdots q \rangle$, $\boldsymbol{\ell}_{i+1} = \langle \cdots (q{-}j{+}1)\ (q{-}j{+}1)\ (q{-}1) \cdots q \rangle$ and $j \geq 2$. Proving $\mathbf{s}^+ \trianglelefteq \mathbf{t}^+$ is also similar, but easier, since then it is never the case that $(\boldsymbol{\ell}_i)^+ = (\boldsymbol{\ell}_{i+1})^+$. ∎

3.4. The vector lattice and distribution lattice. Recall [5] that a *lattice* is an algebra $\langle \mathcal{S}, \sqsubseteq, \sqcap, \sqcup \rangle$ in which $\mathcal{S}$ is a set, $\sqsubseteq$ is a partial ordering on $\mathcal{S}$, and for all $a, b \in \mathcal{S}$, there is a unique *greatest lower bound (glb)* $a \sqcap b$ and *least upper bound (lub)* $a \sqcup b$. The lattice is called *distributive* if these operators satisfy the distributive law:

$$\text{for all } a, b, c \text{ in } \mathcal{S}, \quad a \sqcap (b \sqcup c) \quad = \quad (a \sqcap b) \sqcup (a \sqcap c).$$

Optionally the lattice can have a *greatest element* $\top$ and *least element* $\bot$.

DEFINITION 3.11. *Let* $\leq_{vec}$ *be the element-wise ordering on vectors (sequences) in* $\Re^n$. *Then*

$$\mathbf{x} \leq_{vec} \mathbf{y} \quad \textit{iff} \quad x_i \leq y_i \ \textit{for } 1 \leq i \leq n.$$

Also define vector element-wise minima and maxima as

$$\begin{array}{rcccccc} \mathbf{x} & \mathbf{min}_{vec} & \mathbf{y} & = & \langle & \min(x_1, y_1) \cdots \min(x_n, y_n) & \rangle, \\ \mathbf{x} & \mathbf{max}_{vec} & \mathbf{y} & = & \langle & \max(x_1, y_1) \cdots \max(x_n, y_n) & \rangle. \end{array}$$

THEOREM 3.12. *The nonnegative vectors* $\langle {\Re_+}^n, \leq_{vec}, \mathbf{min}_{vec}, \mathbf{max}_{vec} \rangle$ *form a distributive lattice called the* vector lattice.

The set $\mathcal{P}$ *of distribution sequences of length* n *(ascending nonnegative vectors* $\mathbf{v}$ *with* $v_n = 1$*) also form a distributive lattice,* $\langle \mathcal{P}, \leq_{vec}, \mathbf{min}_{vec}, \mathbf{max}_{vec} \rangle$, *called the* distribution lattice, *with least element* $\bot = \langle 0\,0 \cdots 0\,1 \rangle$ *and greatest element* $\top = \langle 1\,1 \cdots 1\,1 \rangle$.

Proof. The one-dimensional algebra $\langle \Re_+, \leq, \min, \max \rangle$ is a distributive lattice. The vector properties required here follow from this. ∎

3.5. The majorization lattice and density lattice. We reproduce basic majorization concepts developed in [34]. Majorization as defined here is an extension of the classical majorization of Muirhead and Hardy, Littlewood, and Pólya [16], which is useful in the study of inequalities. Marshall and Olkin [27] provide a very good account of the classical theory and its applications. The classical theory defines

a majorization ordering on descendingly ordered (or sometimes ascendingly ordered) multisets, and although quite beautiful it is also quite complex. We have transplanted the theory to rely only on linear algebra and convexity. Thus the definitions in this section are ours, and the results vary from those in [27].

DEFINITION 3.13. *The* zeta matrix $\int = (\zeta_{ij})$ *is defined by*

$$\zeta_{ij} \;=\; 1 \text{ if } i \geq j, \quad 0 \text{ otherwise.}$$

The Möbius matrix $\boldsymbol{\partial} = (\mu_{ij})$ *is defined by*

$$\mu_{ij} \;=\; 1 \text{ if } i = j, \quad -1 \text{ if } j = i-1, \quad 0 \text{ otherwise.}$$

The Möbius matrix is the inverse of the zeta matrix. For example, when $n = 5$:

$$\int \;=\; \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad \boldsymbol{\partial} \;=\; \int^{-1} \;=\; \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

The Möbius matrix is also significant in that it corresponds directly to the concept of pairwise exchange (of adjacent elements in a sequence). The theory of Möbius inversion [36] gives a generalized notion of differential on partially ordered domains (although here we consider only totally ordered sequences). We can think of $\int$ as an integral operator (which transforms a sequence to its left-to-right "integral"), with $\boldsymbol{\partial}$ as its inverse differential operator.

THEOREM 3.14. *If* $\mathbf{x}$ *and* $\mathbf{y}$ *are density sequences (so* $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i = 1$*), then*

$$\mathbf{x} \;\preceq\; \mathbf{y} \qquad \textit{iff} \qquad (\textstyle\int \mathbf{x}) \;\leq_{vec}\; (\int \mathbf{y}).$$

If $\mathbf{v}$ *and* $\mathbf{v}'$ *are distribution sequences, then* $\boldsymbol{\partial}\,\mathbf{v}$ *and* $\boldsymbol{\partial}\,\mathbf{v}'$ *are density sequences and*

$$\mathbf{v} \;\leq_{vec}\; \mathbf{v}' \qquad \textit{iff} \qquad (\boldsymbol{\partial}\,\mathbf{v}) \;\preceq\; (\boldsymbol{\partial}\,\mathbf{v}').$$

Proof. $(\int \mathbf{x}) \leq_{vec} (\int \mathbf{y})$ is equivalent to

$$x_1 \;\leq\; y_1, \qquad x_1 + x_2 \;\leq\; y_1 + y_2, \qquad \ldots, \qquad x_1 + x_2 + \cdots + x_n \;\leq\; y_1 + y_2 + \cdots + y_n.$$

Note that $\mathbf{x}$ is a density sequence iff $(\int \mathbf{x})$ is a distribution sequence. The second statement then follows since the Möbius and zeta transformations are inverses of one another. □

This isomorphism between $\leq_{vec}$ and $\preceq$ implies that majorization defines a lattice.

DEFINITION 3.15. *Majorization lub and glb operators are definable by*

$$\begin{aligned} \mathbf{x} \;\sqcup\; \mathbf{y} \;&=\; \boldsymbol{\partial}\,((\textstyle\int \mathbf{x}) \;\mathbf{max}_{vec}\; (\int \mathbf{y})), \\ \mathbf{x} \;\sqcap\; \mathbf{y} \;&=\; \boldsymbol{\partial}\,((\textstyle\int \mathbf{x}) \;\mathbf{min}_{vec}\; (\int \mathbf{y})). \end{aligned}$$

THEOREM 3.16. *The nonnegative reals ordered by majorization forms a distributive lattice* $\langle \Re_+{}^n, \preceq, \sqcap, \sqcup \rangle$ *called the* majorization lattice.

The set $\mathcal{D}$ *of density sequences of length* n *(nonnegative* $\mathbf{x}$ *with* $\sum_{i=1}^{n} x_i = 1$*) forms a distributive lattice* $\langle \mathcal{D}, \preceq, \sqcap, \sqcup, \bot, \top \rangle$ *called the* density lattice*, with least element* $\bot = \langle 0\,0 \cdots 0\,1 \rangle$ *and greatest element* $\top = \langle 1\,0 \cdots 0\,0 \rangle$.

Proof. The transformation $\mathbf{x} \mapsto \int\mathbf{x}$ defines a lattice isomorphism between the vector and majorization lattices and between the distribution and density lattices. Here $\mathbf{x} \sqcap \mathbf{y}$ and $\mathbf{x} \sqcup \mathbf{y}$ are defined just so as to be the majorization glb and lub:

$$\begin{array}{l|l}
\mathbf{z} \preceq \mathbf{x},\quad \mathbf{z} \preceq \mathbf{y} & \mathbf{x} \preceq \mathbf{z},\quad \mathbf{y} \preceq \mathbf{z} \\
\Leftrightarrow\quad (\textstyle\int\mathbf{z}) \leq_{vec} (\int\mathbf{x}),\ (\int\mathbf{z}) \leq_{vec} (\int\mathbf{y}) & \Leftrightarrow\quad (\textstyle\int\mathbf{x}) \leq_{vec} (\int\mathbf{z}),\ (\int\mathbf{y}) \leq_{vec} (\int\mathbf{z}) \\
\Leftrightarrow\quad (\textstyle\int\mathbf{z}) \leq_{vec} ((\int\mathbf{x})\ \mathbf{min}_{vec}\ (\int\mathbf{y})) & \Leftrightarrow\quad ((\textstyle\int\mathbf{x})\ \mathbf{max}_{vec}\ (\int\mathbf{y})) \leq_{vec} (\int\mathbf{z}) \\
\Leftrightarrow\quad \mathbf{z} \preceq \boldsymbol{\partial}((\textstyle\int\mathbf{x})\ \mathbf{min}_{vec}\ (\int\mathbf{y})) & \Leftrightarrow\quad \boldsymbol{\partial}((\textstyle\int\mathbf{x})\ \mathbf{max}_{vec}\ (\int\mathbf{y})) \preceq \mathbf{z} \\
\Leftrightarrow\quad \mathbf{z} \preceq \mathbf{x} \sqcap \mathbf{y}. & \Leftrightarrow\quad \mathbf{x} \sqcup \mathbf{y} \preceq \mathbf{z}.
\end{array}$$

Thus the majorization algebra also forms a distributive lattice. □

Even when $\mathbf{x}$ and $\mathbf{y}$ are in descending order, the sequences $(\mathbf{x} \sqcap \mathbf{y})$ and $(\mathbf{x} \sqcup \mathbf{y})$ defined here are not necessarily in descending order:

$$\mathbf{x} = \langle\, 2^{-2}\ 2^{-2}\ 2^{-3}\ 2^{-4}\ 2^{-4}\ 2^{-4}\ 2^{-4}\ 2^{-4}\ 2^{-4} \,\rangle$$

and

$$\mathbf{y} = \langle\, 2^{-2}\ 2^{-3}\ 2^{-3}\ 2^{-3}\ 2^{-3}\ 2^{-3}\ 2^{-4}\ 2^{-5}\ 2^{-5} \,\rangle$$

yield the least upper bound

$$\mathbf{x} \sqcup \mathbf{y} = \boldsymbol{\partial}(\textstyle\int\mathbf{x}\ \mathbf{max}_{vec}\ \int\mathbf{y}) = \langle\, 2^{-2}\ 2^{-2}\ 2^{-3}\ \boxed{2^{-4}}\ 2^{-4}\ \boxed{2^{-3}}\ 2^{-4}\ 2^{-5}\ 2^{-5} \,\rangle.$$

See Figure 3.6.

3.6. The imbalance lattice: A discrete cousin of the majorization lattice. Since every pair of sequences in Figures 3.2 and 3.3 has a unique glb and lub, the imbalance ordering is not only a partial order but also a *lattice*. In this section we prove this by showing that every pair of sequences $\mathbf{s}$, $\mathbf{t}$ has a glb $\mathbf{s} \wedge \mathbf{t}$ and lub $\mathbf{s} \vee \mathbf{t}$. We also relate the imbalance lattice directly to the majorization lattice, as illustrated in Figures 3.5–3.7.

THEOREM 3.17. *On tree path-length sequences, the imbalance ordering is isomorphic to the majorization ordering. Specifically, whenever* $\mathbf{s}$ *and* $\mathbf{t}$ *are tree path-length sequences, then*

$$\mathbf{s} \trianglelefteq \mathbf{t} \quad \textit{iff} \quad 2^{-\mathbf{s}} \preceq 2^{-\mathbf{t}}.$$

Proof. We show first that balancing exchanges cause a reduction in the majorization ordering. Let $\mathbf{s}$ be the result of a balancing exchange on $\mathbf{t}$ (so $\mathbf{s} \trianglelefteq \mathbf{t}$). Then the following holds:

$$\begin{array}{rcccccccccccc}
\mathbf{t} = & (& \cdot & \cdot & \cdot & p & u & \cdot & \cdot & v & (q+1) & (q+1) & \cdot & \cdot\ ) \\
\mathbf{s} = & (& \cdot & \cdot & \cdot & (p+1) & (p+1) & u & \cdot & \cdot & v & q & \cdot & \cdot\ ) \\
2^{-\mathbf{t}} = & (& \cdot & \cdot & \cdot & 2^{-p} & 2^{-u} & \cdot & \cdot & 2^{-v} & 2^{-(q+1)} & 2^{-(q+1)} & \cdot & \cdot\ ) \\
2^{-\mathbf{s}} = & (& \cdot & \cdot & \cdot & 2^{-(p+1)} & 2^{-(p+1)} & 2^{-u} & \cdot & \cdot & 2^{-v} & 2^{-q} & \cdot & \cdot\ ) \\
\textstyle\int 2^{-\mathbf{t}} = & (& \cdot & \cdot & S & S+2^{-p} & S+2^{-p}+2^{-u} & \cdot & \cdot & T & T+2^{-(q+1)} & T+2^{-q} & \cdot & 1\ ) \\
\textstyle\int 2^{-\mathbf{s}} = & (& \cdot & \cdot & S & S+2^{-(p+1)} & S+2^{-p} & \cdot & \cdot & T-2^{-v} & T & T+2^{-q} & \cdot & 1\ ) \\
\hline
 & (& 0 & \cdot & 0 & +2^{-(p+1)} & +2^{-u} & \cdot & \cdot & +2^{-v} & +2^{-(q+1)} & 0 & \cdot & 0\ )
\end{array}$$

Thus $\int 2^{-\mathbf{s}}$ and $\int 2^{-\mathbf{t}}$ differ only in the values appearing between p and q, and each element in $\int 2^{-\mathbf{t}} - \int 2^{-\mathbf{s}}$ is nonnegative, so $2^{-\mathbf{s}} \preceq 2^{-\mathbf{t}}$.

$$
\begin{array}{rcl}
\mathbf{s} & = & (\ 1 \quad 2 \quad 4 \quad 4 \quad 5 \quad 5 \quad 5 \quad 6 \quad 6\ ) \\
2^{-\mathbf{s}} & = & (\ 2^{-1} \quad 2^{-2} \quad 2^{-4} \quad 2^{-4} \quad 2^{-5} \quad 2^{-5} \quad 2^{-5} \quad 2^{-6} \quad 2^{-6}\ ) \\
\int 2^{-\mathbf{s}} & = & 2^{-7}\ (\ 64 \quad 96 \quad 104 \quad 112 \quad 116 \quad 120 \quad 124 \quad 126 \quad 128\ ) \\
\\
\mathbf{t} & = & (\ 2 \quad 2 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 7\ ) \\
2^{-\mathbf{t}} & = & (\ 2^{-2} \quad 2^{-2} \quad 2^{-2} \quad 2^{-3} \quad 2^{-4} \quad 2^{-5} \quad 2^{-6} \quad 2^{-7} \quad 2^{-7}\ ) \\
\int 2^{-\mathbf{t}} & = & 2^{-7}\ (\ 32 \quad 64 \quad 96 \quad 112 \quad 120 \quad 124 \quad 126 \quad 127 \quad 128\ ) \\
\\
\int 2^{-\mathbf{s}}\ \mathbf{max}_{vec} \int 2^{-\mathbf{t}} & = & 2^{-7}\ (\ 64 \quad 96 \quad 104 \quad 112 \quad 120 \quad 124 \quad 126 \quad 127 \quad 128\ ) \\
\boldsymbol{\partial}(\int 2^{-\mathbf{s}}\ \mathbf{max}_{vec} \int 2^{-\mathbf{t}}) & = & 2^{-7}\ (\ 64 \quad 32 \quad 8 \quad 8 \quad 8 \quad 4 \quad 2 \quad 1 \quad 1\ ) \\
= \ 2^{-\mathbf{s}} \sqcup 2^{-\mathbf{t}} & = & (\ 2^{-1} \quad 2^{-2} \quad 2^{-4} \quad 2^{-4} \quad 2^{-4} \quad 2^{-5} \quad 2^{-6} \quad 2^{-7} \quad 2^{-7}\ ) \\
\mathbf{s} \vee \mathbf{t} & = & (\ 1 \quad 2 \quad 4 \quad 4 \quad 4 \quad 5 \quad 6 \quad 7 \quad 7\ ) \\
\\
\int 2^{-\mathbf{s}}\ \mathbf{min}_{vec} \int 2^{-\mathbf{t}} & = & 2^{-7}\ (\ 32 \quad 64 \quad 96 \quad 112 \quad 116 \quad 120 \quad 124 \quad 126 \quad 128\ ) \\
\boldsymbol{\partial}(\int 2^{-\mathbf{s}}\ \mathbf{min}_{vec} \int 2^{-\mathbf{t}}) & = & 2^{-7}\ (\ 32 \quad 32 \quad 32 \quad 16 \quad 4 \quad 4 \quad 4 \quad 2 \quad 2\ ) \\
= \ 2^{-\mathbf{s}} \sqcap 2^{-\mathbf{t}} & = & (\ 2^{-2} \quad 2^{-2} \quad 2^{-2} \quad 2^{-3} \quad 2^{-5} \quad 2^{-5} \quad 2^{-5} \quad 2^{-6} \quad 2^{-6}\ ) \\
\mathbf{s} \wedge \mathbf{t} & = & (\ 2 \quad 2 \quad 2 \quad 3 \quad 5 \quad 5 \quad 5 \quad 6 \quad 6\ )
\end{array}
$$

FIG. 3.5. *Related points in the majorization and imbalance lattices, showing their connection.*

$$
\begin{array}{rcl}
\mathbf{s} & = & (\ 2 \quad 2 \quad 3 \quad 4 \quad 4 \quad 4 \quad 4 \quad 4 \quad 4\ ) \\
2^{-\mathbf{s}} & = & (\ 2^{-2} \quad 2^{-2} \quad 2^{-3} \quad 2^{-4} \quad 2^{-4} \quad 2^{-4} \quad 2^{-4} \quad 2^{-4} \quad 2^{-4}\ ) \\
\int 2^{-\mathbf{s}} & = & 2^{-7}\ (\ 32 \quad 64 \quad 80 \quad 88 \quad 96 \quad 104 \quad 112 \quad 120 \quad 128\ ) \\
\\
\mathbf{t} & = & (\ 2 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 4 \quad 5 \quad 5\ ) \\
2^{-\mathbf{t}} & = & (\ 2^{-2} \quad 2^{-3} \quad 2^{-3} \quad 2^{-3} \quad 2^{-3} \quad 2^{-3} \quad 2^{-4} \quad 2^{-5} \quad 2^{-5}\ ) \\
\int 2^{-\mathbf{t}} & = & 2^{-7}\ (\ 32 \quad 48 \quad 64 \quad 80 \quad 96 \quad 112 \quad 120 \quad 124 \quad 128\ ) \\
\\
\int 2^{-\mathbf{s}}\ \mathbf{max}_{vec} \int 2^{-\mathbf{t}} & = & 2^{-7}\ (\ 32 \quad 64 \quad 80 \quad 88 \quad 96 \quad 112 \quad 120 \quad 124 \quad 128\ ) \\
\boldsymbol{\partial}(\int 2^{-\mathbf{s}}\ \mathbf{max}_{vec} \int 2^{-\mathbf{t}}) & = & 2^{-7}\ (\ 32 \quad 32 \quad 16 \quad 8 \quad 8 \quad 16 \quad 8 \quad 4 \quad 4\ ) \\
= \ 2^{-\mathbf{s}} \sqcup 2^{-\mathbf{t}} & = & (\ 2^{-2} \quad 2^{-2} \quad 2^{-3} \quad \boxed{2^{-4}} \quad 2^{-4} \quad \boxed{2^{-3}} \quad 2^{-4} \quad 2^{-5} \quad 2^{-5}\ ) \\
\mathbf{s} \vee \mathbf{t} & = & (\ 2 \quad 2 \quad 3 \quad \boxed{3} \quad 4 \quad \boxed{4} \quad 4 \quad 5 \quad 5\ )
\end{array}
$$

FIG. 3.6. *Results of* $(2^{-\mathbf{s}} \sqcap 2^{-\mathbf{t}})$ *and* $(2^{-\mathbf{s}} \sqcup 2^{-\mathbf{t}})$ *are not necessarily in descending order.*

$$
\begin{array}{rcl}
\mathbf{s} & = & (\ 1 \quad 2 \quad 4 \quad 5 \quad 5 \quad 5 \quad 5 \quad 5 \quad 5\ ) \\
2^{-\mathbf{s}} & = & (\ 2^{-1} \quad 2^{-2} \quad 2^{-4} \quad 2^{-5} \quad 2^{-5} \quad 2^{-5} \quad 2^{-5} \quad 2^{-5} \quad 2^{-5}\ ) \\
\int 2^{-\mathbf{s}} & = & 2^{-7}\ (\ 64 \quad 96 \quad 104 \quad 108 \quad 112 \quad 116 \quad 120 \quad 124 \quad 128\ ) \\
\\
\mathbf{t} & = & (\ 2 \quad 2 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 7\ ) \\
2^{-\mathbf{t}} & = & (\ 2^{-2} \quad 2^{-2} \quad 2^{-2} \quad 2^{-3} \quad 2^{-4} \quad 2^{-5} \quad 2^{-6} \quad 2^{-7} \quad 2^{-7}\ ) \\
\int 2^{-\mathbf{t}} & = & 2^{-7}\ (\ 32 \quad 64 \quad 96 \quad 112 \quad 120 \quad 124 \quad 126 \quad 127 \quad 128\ ) \\
\\
\int 2^{-\mathbf{s}}\ \mathbf{min}_{vec} \int 2^{-\mathbf{t}} & = & 2^{-7}\ (\ 32 \quad 64 \quad 96 \quad 108 \quad 112 \quad 116 \quad 120 \quad 124 \quad 128\ ) \\
\boldsymbol{\partial}(\int 2^{-\mathbf{s}}\ \mathbf{min}_{vec} \int 2^{-\mathbf{t}}) & = & 2^{-7}\ (\ 32 \quad 32 \quad 32 \quad 12 \quad 4 \quad 4 \quad 4 \quad 4 \quad 4\ ) \\
= \ 2^{-\mathbf{s}} \sqcap 2^{-\mathbf{t}} & = & (\ 2^{-2} \quad 2^{-2} \quad 2^{-2} \quad \boxed{2^{-\alpha}} \quad \boxed{2^{-5}} \quad 2^{-5} \quad 2^{-5} \quad 2^{-5} \quad 2^{-5}\ ) \\
 & & -\alpha \ = \ (-7 + \log_2(12)) \ \approx \ -3.4150375 \\
 & \succeq & (\ 2^{-2} \quad 2^{-2} \quad 2^{-2} \quad \boxed{2^{-4}} \quad \boxed{2^{-4}} \quad 2^{-5} \quad 2^{-5} \quad 2^{-5} \quad 2^{-5}\ ) \\
\\
\mathbf{s} \wedge \mathbf{t} & = & (\ 2 \quad 2 \quad 2 \quad \boxed{4} \quad \boxed{4} \quad 5 \quad 5 \quad 5 \quad 5\ )
\end{array}
$$

$2^{-(\mathbf{s} \wedge \mathbf{t})} \preceq (2^{-\mathbf{s}} \sqcap 2^{-\mathbf{t}})$; the two differ where indicated. Nonintegral exponents occur for $n \geq 9$.

FIG. 3.7. *The imbalance lattice is not simply conjugate to a sublattice of the majorization lattice.*

The proof of the converse, that $2^{-\mathbf{s}} \preceq 2^{-\mathbf{t}}$ implies $\mathbf{s} \trianglelefteq \mathbf{t}$ for tree path-length

sequences $\mathbf{s}$, $\mathbf{t}$, can proceed by assuming a counterexample for which the difference in the levels of balance of

$$m \quad = \quad (\text{level of balance of } \mathbf{s}) \; - \; (\text{level of balance of } \mathbf{t})$$

is minimal. Since $2^{-\mathbf{s}} \preceq 2^{-\mathbf{t}}$ let a, b, c, d be the rightmost aligned pairwise-differing values among the two sorted sequences such that $\mathbf{s} = \langle \cdots \; a \; \cdots \; b \; \cdots \rangle$ and $\mathbf{t} = \langle \cdots \; c \; \cdots \; d \; \cdots \rangle$, where $c < a$, $b < d$ because of the majorization inequality, $a \leq b$ and $c \leq d$ because the sequences are ascending, $c \neq d$ since $c < a \leq b < d$, and finally $2^{-a} + \cdots + 2^{-b} = 2^{-c} + \cdots + 2^{-d}$, which is always possible by the Kraft equality. Because $b < d$ necessarily $\mathbf{t} = \langle \cdots \; c \; \cdots \; d \; d \; \cdots \rangle$, since otherwise we reach a contradiction (multiplying both sides of the equality by 2^d makes the left side even but the right side odd). Thus, if we define the result $\mathbf{t}' = \langle \cdots \; (c+1) \; (c+1) \; \cdots \; (d-1) \; \cdots \rangle$ of a balancing exchange on $\mathbf{t} = \langle \cdots \; c \; \cdots \; d \; d \; \cdots \rangle$, then the level difference between $\mathbf{s}$ and $\mathbf{t}'$ is at most $(m-1)$, and $2^{-\mathbf{t}'} \preceq 2^{-\mathbf{t}}$. Furthermore we claim $2^{-\mathbf{s}} \preceq 2^{-\mathbf{t}'}$, using the following schematic:

$$\begin{array}{rcccccccccc}
\mathbf{t} & = & (\cdot & \cdot & \cdot & c & u & \cdot & d & d & \cdot \; \cdot) \\
\mathbf{t}' & = & (\cdot & \cdot & \cdot & (c+1) & \cdot & \cdot & & (d-1) & \cdot \; \cdot) \\
\mathbf{s} & = & (\cdot & \cdot & \cdot & a & w & \cdot & \cdot & b & \cdot \; \cdot) \\
 & & & & & & & & & & \\
2^{-\mathbf{t}} & = & (\cdot & \cdot & \cdot & 2^{-c} & 2^{-u} & \cdot & 2^{-d} & 2^{-d} & \cdot \; \cdot) \\
2^{-\mathbf{t}'} & = & (\cdot & \cdot & \cdot & 2^{-(c+1)} & 2^{-(c+1)} & \cdot & \cdot & 2^{-(d-1)} & \cdot \; \cdot) \\
2^{-\mathbf{s}} & = & (\cdot & \cdot & \cdot & 2^{-a} & 2^{-w} & \cdot & \cdot & 2^{-b} & \cdot \; \cdot) \\
 & & & & & & & & & & \\
\int 2^{-\mathbf{t}} - \int 2^{-\mathbf{s}} & = & (0 & \cdot & 0 & +(2^{-c} - 2^{-a}) & +S_1 & \cdot & +S_k & 0 & \cdot \; 0) \\
\int 2^{-\mathbf{t}} - \int 2^{-\mathbf{t}'} & = & (0 & \cdot & 0 & +2^{-(c+1)} & +2^{-u} & \cdot & +2^{-d} & 0 & \cdot \; 0) \\
\hline
\int 2^{-\mathbf{t}'} - \int 2^{-\mathbf{s}} & = & (0 & \cdot & 0 & +(2^{-(c+1)} - 2^{-a}) & +(S_1 - 2^{-u}) & \cdot & +(S_k - 2^{-d}) & 0 & \cdot \; 0)
\end{array}$$

Because $2^{-\mathbf{s}} \preceq 2^{-\mathbf{t}}$, the running totals $S_1, \ldots, S_k$ are nonnegative. Also, $(2^{-(c+1)} - 2^{-a}) \geq 0$ since $c < a$. Furthermore $c \leq (a-1) \leq w$, implying $S_1 - 2^{-u} = (2^{-c} - 2^{-a}) - 2^{w} \geq 2^{-(a-1)} - 2^{-w} \geq 0$. Finally $S_k + 2^{-d} - 2^{-b} = 0$, so $b \leq (d-1)$ implies $S_k - 2^{-d} = (2^{-b} - 2^{-d}) - 2^{-d} = 2^{-b} - 2^{-(d-1)} \geq 0$. Thus $\int 2^{-\mathbf{s}} \leq_{vec} \int 2^{-\mathbf{t}'}$ (i.e., $2^{-\mathbf{s}} \preceq 2^{-\mathbf{t}'}$), contradicting the assumed minimality of m, and existence of a counterexample. ∎

THEOREM 3.18. *The imbalance ordering on binary trees determines a bona fide lattice in which, for all* $\mathbf{s}$ *and* $\mathbf{t}$*, the glb* $\mathbf{s} \wedge \mathbf{t}$ *and lub* $\mathbf{s} \vee \mathbf{t}$ *are defined with the following recursive algorithms, where the expansion used is chosen from among the lower and upper expansions:*

$$\mathbf{s} \wedge \mathbf{t} \quad = \quad \begin{cases} \mathbf{s} & \text{if} \quad \mathbf{s} \trianglelefteq \mathbf{t}, \\ \mathbf{t} & \text{if} \quad \mathbf{t} \trianglelefteq \mathbf{s}, \\ \textit{the greatest expansion of } \; \mathbf{s}^- \wedge \mathbf{t}^- & \\ \textit{that is also a lower bound for } \mathbf{s} \textit{ and } \mathbf{t} & \textit{otherwise}; \end{cases}$$

$$\mathbf{s} \vee \mathbf{t} \quad = \quad \begin{cases} \mathbf{t} & \text{if} \quad \mathbf{s} \trianglelefteq \mathbf{t}, \\ \mathbf{s} & \text{if} \quad \mathbf{t} \trianglelefteq \mathbf{s}, \\ \textit{the least expansion of } \; \mathbf{s}^- \vee \mathbf{t}^- & \\ \textit{that is also an upper bound for } \mathbf{s} \textit{ and } \mathbf{t} & \textit{otherwise}. \end{cases}$$

Proof. We must show that, whenever $\mathbf{s}$ and $\mathbf{t}$ are tree path-length sequences of length n, there are unique path-length sequences $\mathbf{s} \wedge \mathbf{t}$ and $\mathbf{s} \vee \mathbf{t}$ such that the following hold:

- $\mathbf{s} \wedge \mathbf{t} \trianglelefteq \mathbf{s}, \; \mathbf{t}$; also, if $\boldsymbol{\ell}$ is any path-length sequence, then $\boldsymbol{\ell} \trianglelefteq \mathbf{s}, \; \mathbf{t}$ iff $\boldsymbol{\ell} \trianglelefteq \mathbf{s} \wedge \mathbf{t}$.
- $\mathbf{s}, \; \mathbf{t} \trianglelefteq \mathbf{s} \vee \mathbf{t}$; also, if $\boldsymbol{\ell}$ is any path-length sequence, then $\mathbf{s}, \; \mathbf{t} \trianglelefteq \boldsymbol{\ell}$ iff $\mathbf{s} \vee \mathbf{t} \trianglelefteq \boldsymbol{\ell}$.

TABLE 3.2

Elaboration of the first example of representative bounds in Figure 3.3*, showing how* $\mathbf{s} \wedge \mathbf{t}$ *and* $\mathbf{s} \vee \mathbf{t}$ *can be derived with their recursive algorithms.*

n	$\mathbf{s}$	$\mathbf{t}$	$\mathbf{s} \wedge \mathbf{t}$	$\mathbf{s} \vee \mathbf{t}$
9	$\langle\ 1\ 4\ 4\ 4\ 4\ 4\ 4\ 4\ 4\ \rangle$	$\langle\ 2\ 2\ 2\ 3\ 5\ 5\ 5\ 6\ 6\ \rangle$	$\langle\ 2\ 2\ 3\ 4\ 4\ 4\ 4\ 4\ 4\ \rangle$	$\langle\ 1\ 3\ 3\ 3\ 5\ 5\ 5\ 6\ 6\ \rangle$
	$\uparrow$ lower expansion	$\uparrow$ upper expansion	$\uparrow$ lower expansion	$\uparrow$ upper expansion
8	$\langle\ 1\ 3\ 4\ 4\ 4\ 4\ 4\ 4\ \rangle$	$\langle\ 2\ 2\ 2\ 3\ 5\ 5\ 5\ 5\ \rangle$	$\langle\ 2\ 2\ 3\ 3\ 4\ 4\ 4\ 4\ \rangle$	$\langle\ 1\ 3\ 3\ 3\ 5\ 5\ 5\ 5\ \rangle$
	$\uparrow$ lower expansion	$\uparrow$ lower expansion	$\uparrow$ lower expansion	$\uparrow$ lower expansion
7	$\langle\ 1\ 3\ 3\ 4\ 4\ 4\ 4\ 4\ \rangle$	$\langle\ 2\ 2\ 2\ 3\ 4\ 5\ 5\ \rangle$	$\langle\ 2\ 2\ 2\ 4\ 4\ 4\ 4\ \rangle$	$\langle\ 1\ 3\ 3\ 3\ 4\ 5\ 5\ \rangle$
	$\uparrow$ lower expansion	$\uparrow$ upper expansion	$\uparrow$ lower expansion	$\uparrow$ upper expansion
6	$\langle\ 1\ 3\ 3\ 3\ 4\ 4\ \rangle$	$\langle\ 2\ 2\ 2\ 3\ 4\ 4\ \rangle$	$\langle\ 2\ 2\ 2\ 3\ 4\ 4\ \rangle$	$\langle\ 1\ 3\ 3\ 3\ 4\ 4\ \rangle$

This can be done by induction on n. We consider only the glb here, the proof for the lub being similar. The theorem holds trivially for $n \leq 6$, since then the trees are totally ordered. Assume that it holds for sequences of size $n-1$ or less.

First, $\mathbf{s}$ and $\mathbf{t}$ must have a common lower bound: The glb $\mathbf{s}^- \wedge \mathbf{t}^-$ exists by induction, and (Theorems 3.9 and 3.10) lower expansion gives a lower bound

$$(\mathbf{s}^- \wedge \mathbf{t}^-)_+ \ \trianglelefteq\ (\mathbf{s}^-)_+ \ \trianglelefteq\ \mathbf{s}, \qquad (\mathbf{s}^- \wedge \mathbf{t}^-)_+ \ \trianglelefteq\ (\mathbf{t}^-)_+ \ \trianglelefteq\ \mathbf{t}.$$

Second, if $\mathbf{s}$ and $\mathbf{t}$ have two greatest lower bounds $\boldsymbol{\ell}$ and $\boldsymbol{\ell}'$, then they must be equal: From $\boldsymbol{\ell} \trianglelefteq \mathbf{s},\ \mathbf{t}$ and $\boldsymbol{\ell}' \trianglelefteq \mathbf{s},\ \mathbf{t}$ we infer $\boldsymbol{\ell}^- \trianglelefteq \mathbf{s}^- \wedge \mathbf{t}^-$ and $\boldsymbol{\ell}'^- \trianglelefteq \mathbf{s}^- \wedge \mathbf{t}^-$ by Theorem 3.10. Since furthermore $\boldsymbol{\ell}$ and $\boldsymbol{\ell}'$ are greatest lower bounds, $\mathbf{s}^- \wedge \mathbf{t}^- \trianglelefteq \boldsymbol{\ell}^-$ and $\mathbf{s}^- \wedge \mathbf{t}^- \trianglelefteq \boldsymbol{\ell}'^-$. Thus $\boldsymbol{\ell}^- = \boldsymbol{\ell}'^-$. By Theorem 3.9, the only way $\boldsymbol{\ell} \neq \boldsymbol{\ell}'$ can arise is that

$$\boldsymbol{\ell} = (\boldsymbol{\ell}^-)_+, \quad \boldsymbol{\ell}' = (\boldsymbol{\ell}^-)^+ \qquad \text{or} \qquad \boldsymbol{\ell} = (\boldsymbol{\ell}^-)^+, \quad \boldsymbol{\ell}' = (\boldsymbol{\ell}^-)_+$$

so $\boldsymbol{\ell} \trianglelefteq \boldsymbol{\ell}'$ or $\boldsymbol{\ell}' \trianglelefteq \boldsymbol{\ell}$, contradicting their both being greatest lower bounds. Thus $\boldsymbol{\ell} = \boldsymbol{\ell}'$.

Third, the algorithm produces a glb that is as good as any other lower bound: Assuming this for $(\mathbf{s}^- \wedge \mathbf{t}^-)$ by induction, there can be no lower bound $\boldsymbol{\ell} \neq (\mathbf{s} \wedge \mathbf{t})$ such that $(\mathbf{s}^- \wedge \mathbf{t}^-)^+ \trianglelefteq \boldsymbol{\ell}$, since otherwise $(\mathbf{s}^- \wedge \mathbf{t}^-) \trianglelefteq \boldsymbol{\ell}^- \trianglelefteq \mathbf{s}^-,\ \mathbf{t}^-$, contradicting our assumption. □

The table of nontrivial examples in Figure 3.3 gives an appreciation for glbs and lubs. The first example (which is illustrated in the figure) is expanded in Table 3.2. Note that the final pairs of entries in $\mathbf{s}$ and $\mathbf{t}$ are the same as the final pairs of entries in $\mathbf{s} \wedge \mathbf{t}$ and $\mathbf{s} \vee \mathbf{t}$ and that the suffix lengths of $\mathbf{s}$ and $\mathbf{t}$ are never shorter than those of $\mathbf{s} \wedge \mathbf{t}$ and $\mathbf{s} \vee \mathbf{t}$.

THEOREM 3.19. *If* $\mathbf{s}$ *and* $\mathbf{t}$ *are path-length sequences of length* n*, then*

$$\mathbf{s} \vee \mathbf{t} \ = \ \begin{cases} (\mathbf{s}^- \vee \mathbf{t}^-)^+ & \text{if} \quad \mathbf{s} = (\mathbf{s}^-)^+ \quad \text{and} \quad \mathbf{t} = (\mathbf{t}^-)^+, \\ (\mathbf{s}^- \vee \mathbf{t}^-)_+ & \text{if} \quad \mathbf{s} = (\mathbf{s}^-)_+ \quad \text{and} \quad \mathbf{t} = (\mathbf{t}^-)_+; \end{cases}$$

$$\mathbf{s} \wedge \mathbf{t} \ = \ \begin{cases} (\mathbf{s}^- \wedge \mathbf{t}^-)^+ & \text{if} \quad \mathbf{s} = (\mathbf{s}^-)^+ \quad \text{and} \quad \mathbf{t} = (\mathbf{t}^-)^+, \\ (\mathbf{s}^- \wedge \mathbf{t}^-)_+ & \text{if} \quad \mathbf{s} = (\mathbf{s}^-)_+ \quad \text{and} \quad \mathbf{t} = (\mathbf{t}^-)_+. \end{cases}$$

Otherwise, if either $\mathbf{s} = (\mathbf{s}^-)^+$ *and* $\mathbf{t} = (\mathbf{t}^-)_+$*, or* $\mathbf{s} = (\mathbf{s}^-)_+$ *and* $\mathbf{t} = (\mathbf{t}^-)^+$*, then either* $\mathbf{s} \wedge \mathbf{t} = (\mathbf{s}^- \wedge \mathbf{t}^-)^+$ *and* $\mathbf{s} \vee \mathbf{t} = (\mathbf{s}^- \vee \mathbf{t}^-)_+$*, or* $\mathbf{s} \wedge \mathbf{t} = (\mathbf{s}^- \wedge \mathbf{t}^-)_+$ *and* $\mathbf{s} \vee \mathbf{t} = (\mathbf{s}^- \vee \mathbf{t}^-)^+$.

Furthermore, if the final pairs of entries of $\mathbf{s}$ *and* $\mathbf{t}$ *are* $\langle p\ p \rangle$ *and* $\langle q\ q \rangle$*, where* $p \leq q$*, then the final pairs of entries of* $\mathbf{s} \wedge \mathbf{t}$ *and* $\mathbf{s} \vee \mathbf{t}$ *are, respectively,* $\langle p\ p \rangle$ *and* $\langle q\ q \rangle$.

Also, the suffix lengths of $\mathbf{s}$ *and* $\mathbf{t}$ *are at least as long as those of* $(\mathbf{s} \wedge \mathbf{t})$ *and* $(\mathbf{s} \vee \mathbf{t})$.

Proof. These properties follow by induction on n. For the basis, they all hold trivially when $n \leq 6$, since then the imbalance lattice is a total order and $\{\, \mathbf{s},\ \mathbf{t} \,\} = \{\, \mathbf{s} \vee \mathbf{t},\ \mathbf{s} \wedge \mathbf{t} \,\}$, and the final two entries of any path-length sequence are a pair by Theorem 3.1. For the induction step, we can write

$$\begin{array}{rclcrcl} \mathbf{s} & = & \langle\ \cdots\ (p{-}i)\ \overbrace{p\ \cdot\ \cdot\ \cdot\ p}^{2h}\ \rangle, & \quad & \mathbf{t} & = & \langle\ \cdots\ (q{-}j)\ \overbrace{q\ \cdot\ \cdot\ \cdot\ q}^{2k}\ \rangle, \\ \mathbf{s}^- & = & \langle\ \cdots\ (p{-}i)\ (p{-}1)\ \overbrace{p\ \cdots\ p}^{2h-2}\ \rangle, & \quad & \mathbf{t}^- & = & \langle\ \cdots\ (q{-}j)\ (q{-}1)\ \overbrace{q\ \cdots\ q}^{2k-2}\ \rangle, \end{array}$$

where $i, j, h, k > 0$, and we assume with no loss of generality that $p \leq q$. There are four cases to consider, depending on the suffix lengths $2h$ of $\mathbf{s}$ and $2k$ of $\mathbf{t}$. In the first, $h = 1$ and $k = 1$ (i.e., $\mathbf{s} = (\mathbf{s}^-)^+$ and $\mathbf{t} = (\mathbf{t}^-)^+$). Then $i = 1$ and $j = 1$ by Theorem 3.1. By induction $(\mathbf{s}^- \wedge \mathbf{t}^-)$ and $(\mathbf{s}^- \vee \mathbf{t}^-)$ have respective final pairs $\langle (p{-}1)\ (p{-}1) \rangle$ and $\langle (q{-}1)\ (q{-}1) \rangle$ and have suffix lengths not exceeding those of $\mathbf{s}^-$ and $\mathbf{t}^-$. Now, by Theorem 3.10 $(\mathbf{s}^- \wedge \mathbf{t}^-)^+ \trianglelefteq (\mathbf{s}^-)^+$ and $(\mathbf{s}^- \wedge \mathbf{t}^-)^+ \trianglelefteq (\mathbf{t}^-)^+$. Because $(\mathbf{s}^-)^+ = \mathbf{s}$ and $(\mathbf{t}^-)^+ = \mathbf{t}$, the recursive algorithm in Theorem 3.18 will find $(\mathbf{s}^- \wedge \mathbf{t}^-)^+ = \mathbf{s} \wedge \mathbf{t}$. Thus the final pair of $\mathbf{s} \wedge \mathbf{t}$ will be $\langle p\ p \rangle$, and it will have suffix length 2. Similarly $\mathbf{s} \vee \mathbf{t} = (\mathbf{s}^- \vee \mathbf{t}^-)^+$ because $\mathbf{s} \vee \mathbf{t} \in \{\, (\mathbf{s}^- \vee \mathbf{t}^-)_+, (\mathbf{s}^- \vee \mathbf{t}^-)^+ \,\}$, and choosing $(\mathbf{s}^- \vee \mathbf{t}^-)_+$ gives a contradiction: if $\mathbf{s} = (\mathbf{s}^-)^+ \trianglelefteq (\mathbf{s}^- \vee \mathbf{t}^-)_+$ and $\mathbf{t} = (\mathbf{t}^-)^+ \trianglelefteq (\mathbf{s}^- \vee \mathbf{t}^-)_+$, then (because of Theorem 3.10) $\mathbf{s}^- = ((\mathbf{s}^-)^+)^- \trianglelefteq ((\mathbf{s}^- \vee \mathbf{t}^-)_+)^- \neq ((\mathbf{s}^- \vee \mathbf{t}^-)^+)^- = \mathbf{s}^- \vee \mathbf{t}^-$ and correspondingly $\mathbf{t}^- = ((\mathbf{t}^-)^+)^- \trianglelefteq ((\mathbf{s}^- \vee \mathbf{t}^-)_+)^- \neq ((\mathbf{s}^- \vee \mathbf{t}^-)^+)^- = \mathbf{s}^- \vee \mathbf{t}^-$, so the lub of $\mathbf{s}^-$ and $\mathbf{t}^-$ is not $\mathbf{s}^- \vee \mathbf{t}^-$, a contradiction. Again the final pair of $\mathbf{s} \vee \mathbf{t}$ will be $\langle q\ q \rangle$, with suffix length 2.

The other three cases, where $h > 1$ and/or $k > 1$, are similar. ∎

4. Submodularity of weighted path-length over the lattices. Huffman codes for a positive descending weight sequence $\mathbf{w} = \langle\, w_1\, w_2\, \cdots\, w_n \,\rangle$ are binary tree path-length sequences $\boldsymbol{\ell} = \langle\, \ell_1\, \ell_2\, \cdots\, \ell_n \,\rangle$ that minimize the weighted path-length

$$g_{\mathbf{w}}(\boldsymbol{\ell}) \quad = \quad \sum_{i=1}^{n} w_i\, \ell_i.$$

In this section we show that $g_{\mathbf{w}}$ is submodular over the lattice of trees, which helps explain why efficient algorithms for finding optimal trees are possible at all.

4.1. Submodularity. Most work on submodular functions assumes that the lattice is the lattice of subsets of a given set, the case originally emphasized by Edmonds [6]. However, the definition applies to any lattice.

DEFINITION 4.1. *A real-valued function* $f : \mathcal{L} \to \Re$ *defined on a lattice* $\langle \mathcal{L}, \sqsubseteq, \sqcap, \sqcup \rangle$ *is* submodular *if*

$$f(x \sqcap y) + f(x \sqcup y) \quad \leq \quad f(x) + f(y)$$

for all $x, y \in \mathcal{L}$. *Equivalently,* f *is submodular if a "differential" inequality holds:*

$$\Delta^2 f(x, y) \stackrel{def}{=} f(x) + f(y) - f(x \sqcap y) - f(x \sqcup y) \geq 0.$$

Section 4.4 discusses the relationship between submodularity and convexity.

4.2. Submodularity of weighted path-length on the majorization lattice. In this section we show that weighted path-length on the imbalance lattice of trees (or a logarithmic variant on the majorization lattice of densities) is a submodular function.

Define the function $G_{\mathbf{w}}$ on the majorization lattice of densities by

$$G_{\mathbf{w}}(\mathbf{x}) \quad = \quad g_{\mathbf{w}}(-\log_2(\mathbf{x})) \quad = \quad -\sum_i \; w_i \; \log_2(x_i).$$

Notice that $G_{\mathbf{w}}$ is convex on $\Re_+{}^n$, since its Hessian

$$\nabla^2 G_{\mathbf{w}} = \left(\frac{\partial^2 G_{\mathbf{w}}(\mathbf{x})}{\partial x_i \partial x_j}\right) = \frac{1}{\ln(2)} \operatorname{diag}\left(\frac{w_i}{x_i^2}\right)$$

is positive semidefinite there [27, p. 448]. (Recall that we are assuming all weights are positive.)

$G_{\mathbf{w}}$ is actually also submodular on the majorization lattice. We prove this directly now and show later how submodularity can be established using only vector calculus.

THEOREM 4.2. *Assuming* $\mathbf{w}$ *is a descending positive sequence of length* n*,* $G_{\mathbf{w}}$ *is submodular on the majorization lattice. That is, for all nonnegative sequences* $\mathbf{x}$*,* $\mathbf{y}$ *of length* n*,*

$$G_{\mathbf{w}}(\mathbf{x} \sqcap \mathbf{y}) \; + \; G_{\mathbf{w}}(\mathbf{x} \sqcup \mathbf{y}) \quad \leq \quad G_{\mathbf{w}}(\mathbf{x}) \; + \; G_{\mathbf{w}}(\mathbf{y}).$$

Proof. By induction on n. For $n = 1$, the inequality is satisfied with equality. Let a_n and b_n be the nth entries of $(\mathbf{x} \sqcap \mathbf{y})$ and $(\mathbf{x} \sqcup \mathbf{y})$, respectively. The theorem follows by induction if we can show that

$$w_n \cdot (-\log_2(a_n)) \; + \; w_n \cdot (-\log_2(b_n)) \quad \leq \quad w_n \cdot (-\log_2(x_n)) \; + \; w_n \cdot (-\log_2(y_n)).$$

Recall that $\mathbf{x} \sqcap \mathbf{y} \;=\; \boldsymbol{\partial}\left((\textstyle\int \mathbf{x})\, \mathbf{min}_{vec}\, (\textstyle\int \mathbf{y})\right)$ and $\mathbf{x} \sqcup \mathbf{y} \;=\; \boldsymbol{\partial}\left((\textstyle\int \mathbf{x})\, \mathbf{max}_{vec}\, (\textstyle\int \mathbf{y})\right)$. There are four cases, depending on $\mathbf{X} = \int \mathbf{x}$ and $\mathbf{Y} = \int \mathbf{y}$ and specifically on the final values

$$X_{n-1} \;=\; \sum_{i=1}^{n-1} x_i, \qquad X_n \;=\; \sum_{i=1}^{n} x_i, \qquad Y_{n-1} \;=\; \sum_{i=1}^{n-1} y_i, \qquad Y_n \;=\; \sum_{i=1}^{n} y_i$$

as follows:

1. if $X_{n-1} \leq Y_{n-1}$ and $X_n \leq Y_n$, then $a_n = x_n$, $b_n = y_n$;
2. if $X_{n-1} \geq Y_{n-1}$ and $X_n \geq Y_n$, then $a_n = y_n$, $b_n = x_n$;
3. if $X_{n-1} \leq Y_{n-1}$ and $X_n \geq Y_n$, then $x_n \geq y_n$, $a_n = Y_n - X_{n-1} = y_n + \epsilon$, $b_n = X_n - Y_{n-1} = x_n - \epsilon$, where $\epsilon = (Y_{n-1} - X_{n-1}) \geq 0$ and $\epsilon \leq x_n - y_n = (x_n \max y_n) - (x_n \min y_n)$;
4. if $X_{n-1} \geq Y_{n-1}$ and $X_n \leq Y_n$, then $y_n \geq x_n$, $a_n = X_n - Y_{n-1} = x_n + \epsilon$, $b_n = Y_n - X_{n-1} = y_n - \epsilon$, where $\epsilon = (X_{n-1} - Y_{n-1}) \geq 0$ and $\epsilon \leq y_n - x_n = (x_n \max y_n) - (x_n \min y_n)$.

Each case satisfies $w_n \cdot (-\log_2(a_n)) + w_n \cdot (-\log_2(b_n)) \;\leq\; w_n \cdot (-\log_2(x_n)) + w_n \cdot (-\log_2(y_n))$ as needed; the first two cases satisfy it with equality, and in the last two we have

$$a_n \quad = \quad (x_n \; \min \; y_n) \; + \; \epsilon, \quad b_n \quad = \quad (x_n \; \max \; y_n) \; - \; \epsilon,$$

but then assuming that $x_n,\, y_n \geq 0$,

$$\log_2(a_n) + \log_2(b_n) \;=\; \log_2(a_n\, b_n) \;=\; \log_2(x_n\, y_n \,+\, \eta) \;\geq\; \log_2(x_n) + \log_2(y_n),$$

where $\eta = \epsilon\, ((x_n \max y_n) - (x_n \min y_n) - \epsilon) \geq 0$ and multiplying by $-w_n$ gives the theorem. □

4.3. Submodularity of weighted path-length on the imbalance lattice.

THEOREM 4.3. *Assuming that* $\mathbf{w}$ *is a descending positive sequence of length* n, $g_{\mathbf{w}}$ *is submodular on the imbalance lattice. That is, for all path-length sequences* $\mathbf{s}$, $\mathbf{t}$ *of length* n,

$$g_{\mathbf{w}}(\mathbf{s} \wedge \mathbf{t}) \;+\; g_{\mathbf{w}}(\mathbf{s} \vee \mathbf{t}) \quad\leq\quad g_{\mathbf{w}}(\mathbf{s}) \;+\; g_{\mathbf{w}}(\mathbf{t}).$$

Proof. The proof is also by induction on n. The theorem holds with equality for $n \leq 6$, since then the lattice of path-length sequences is totally ordered. We sketch the induction step from $n{-}1$ to n, showing $\Delta^2 g_{\mathbf{w}}(\mathbf{s}, \mathbf{t}) \;=\; (g_{\mathbf{w}}(\mathbf{s}) \;+\; g_{\mathbf{w}}(\mathbf{t})) - (g_{\mathbf{w}}(\mathbf{s} \wedge \mathbf{t}) \;+\; g_{\mathbf{w}}(\mathbf{s} \vee \mathbf{t})) \;\geq\; 0$ follows from $\Delta^2 g_{\mathbf{w}}(\mathbf{s}^-, \mathbf{t}^-) \;\geq\; 0$—where $g_{\mathbf{w}}$, when applied to sequences of length $(n{-}1)$, uses only the first $(n{-}1)$ entries of $\mathbf{w}$.

Recall that $2k$ is the suffix length of the path-length sequence

$$\boldsymbol{\ell} = \langle\; \cdots\; (q{-}j)\; \overbrace{q\; \cdots\; q}^{2k}\; \rangle,$$

and j is its suffix increment. The suffix increment is 1 when $(\boldsymbol{\ell}^-)^+ = \boldsymbol{\ell}$, so

$$g_{\mathbf{w}}(\boldsymbol{\ell}) \quad=\quad \begin{cases} g_{\mathbf{w}}(\boldsymbol{\ell}^-) \;+\; (w_{n-1} \;\;\;\;\;\;+\; q \cdot w_n) & \text{if } \boldsymbol{\ell} = (\boldsymbol{\ell}^-)^+ \quad \text{(i.e., } k = 1\text{)}, \\ g_{\mathbf{w}}(\boldsymbol{\ell}^-) \;+\; (w_{n-2k+1} \;+\; q \cdot w_n) & \text{if } \boldsymbol{\ell} = (\boldsymbol{\ell}^-)_+ \quad \text{(i.e., } k > 1\text{)}. \end{cases}$$

Thus $g_{\mathbf{w}}(\mathbf{s}) > g_{\mathbf{w}}(\mathbf{s}^-)$ and $g_{\mathbf{w}}(\mathbf{t}) > g_{\mathbf{w}}(\mathbf{t}^-)$ in all cases.

However, it can happen that $g_{\mathbf{w}}(\mathbf{s} \wedge \mathbf{t}) < g_{\mathbf{w}}(\mathbf{s}^- \wedge \mathbf{t}^-)$ or $g_{\mathbf{w}}(\mathbf{s} \vee \mathbf{t}) < g_{\mathbf{w}}(\mathbf{s}^- \vee \mathbf{t}^-)$ because it is possible either that $\mathbf{s}^- \wedge \mathbf{t}^- \neq (\mathbf{s} \wedge \mathbf{t})^-$ or that $\mathbf{s}^- \vee \mathbf{t}^- \neq (\mathbf{s} \vee \mathbf{t})^-$. Specifically, it is possible that

$$\mathbf{s} \wedge \mathbf{t} \;=\; \langle\; \cdots\; (q{-}j)\; (q{-}j)\; \overbrace{q\; \cdots\; q}^{2k}\; \rangle$$

and

$$\mathbf{s}^- \wedge \mathbf{t}^- \;=\; \langle\; \cdots\; (q{-}j{-}1)\; \overbrace{q\; \cdots\; q}^{2k}\; \rangle,$$

i.e., $\mathbf{s} \wedge \mathbf{t} = (\mathbf{s}^- \wedge \mathbf{t}^-)_+$ and $\mathbf{s} \wedge \mathbf{t}$ has suffix increment $j > 1$, in which case

$$g_{\mathbf{w}}(\mathbf{s} \wedge \mathbf{t}) \quad=\quad g_{\mathbf{w}}(\mathbf{s}^- \wedge \mathbf{t}^-) \;+\; (w_{n-2k+1} \;-\; j \cdot w_{n-2k} \;+\; q \cdot w_n)$$

and the parenthesized expression can be negative.

From Theorem 3.19, the final pairs of entries of $\mathbf{s}$ and $\mathbf{t}$ are always the same as the final pairs of entries of $\mathbf{s} \wedge \mathbf{t}$ and $\mathbf{s} \vee \mathbf{t}$, and the suffix lengths for each of $\mathbf{s}$ and $\mathbf{t}$ cannot be less than those for each of $(\mathbf{s} \wedge \mathbf{t})$ and $(\mathbf{s} \vee \mathbf{t})$. We now consider the same four cases addressed in the proof of Theorem 3.19.

In the case where both $\mathbf{s}$ is the upper expansion of $\mathbf{s}^-$ and $\mathbf{t}$ is the upper expansion of $\mathbf{t}^-$, then by Theorem 3.19, $\mathbf{s} \wedge \mathbf{t} = (\mathbf{s}^- \wedge \mathbf{t}^-)^+$ and $\mathbf{s} \vee \mathbf{t} = (\mathbf{s}^- \vee \mathbf{t}^-)^+$, so

$$\begin{aligned} \Delta^2 g_{\mathbf{w}}(\mathbf{s}, \mathbf{t}) &= (g_{\mathbf{w}}(\mathbf{s}) \;+\; g_{\mathbf{w}}(\mathbf{t})) \;-\; (g_{\mathbf{w}}(\mathbf{s} \wedge \mathbf{t}) \;+\; g_{\mathbf{w}}(\mathbf{s} \vee \mathbf{t})) \\ &= (g_{\mathbf{w}}(\mathbf{s}^-) \;+\; g_{\mathbf{w}}(\mathbf{t}^-)) \;-\; (g_{\mathbf{w}}(\mathbf{s}^- \wedge \mathbf{t}^-) \;+\; g_{\mathbf{w}}(\mathbf{s}^- \vee \mathbf{t}^-)) + 0 \\ &= \Delta^2 g_{\mathbf{w}}(\mathbf{s}^-, \mathbf{t}^-), \end{aligned}$$

with the analysis above for $g_{\mathbf{w}}(\boldsymbol{\ell})$ with $k = 1$. In this situation only the final pairs of entries of $\mathbf{s}$, $\mathbf{t}$ and of $\mathbf{s} \wedge \mathbf{t}$, $\mathbf{s} \vee \mathbf{t}$ can cause the two differences to be unequal, but we now know them to give the same two pairs. Therefore in this case the theorem follows by induction.

It remains to treat the cases where $\mathbf{s}$ is the lower expansion of $\mathbf{s}^-$ or $\mathbf{t}$ is the lower expansion of $\mathbf{t}^-$. In these cases it can happen that $g_{\mathbf{w}}(\mathbf{s} \wedge \mathbf{t}) < g_{\mathbf{w}}(\mathbf{s}^- \wedge \mathbf{t}^-)$ or $g_{\mathbf{w}}(\mathbf{s} \vee \mathbf{t}) < g_{\mathbf{w}}(\mathbf{s}^- \vee \mathbf{t}^-)$ as noted above.

In the case where either $\mathbf{s}$ is the lower expansion of $\mathbf{s}^-$ or $\mathbf{t}$ is the lower expansion of $\mathbf{t}^-$, but not both, then by Theorem 3.19, either $\mathbf{s} \wedge \mathbf{t} = (\mathbf{s}^- \wedge \mathbf{t}^-)_+$ and $\mathbf{s} \vee \mathbf{t} = (\mathbf{s}^- \vee \mathbf{t}^-)^+$, or $\mathbf{s} \wedge \mathbf{t} = (\mathbf{s}^- \wedge \mathbf{t}^-)^+$ and $\mathbf{s} \vee \mathbf{t} = (\mathbf{s}^- \vee \mathbf{t}^-)_+$. The lower expansions among these two cannot yield as large a $g_{\mathbf{w}}$ increase as the lower expansions giving $\mathbf{s}$ and $\mathbf{t}$, because they expand higher-indexed positions (their suffix lengths are never longer), and the suffix increment of $\mathbf{s}^- \wedge \mathbf{t}^-$ or $\mathbf{s}^- \vee \mathbf{t}^-$ can be greater than 1. Therefore $\Delta^2 g_{\mathbf{w}}(\mathbf{s}, \mathbf{t}) \geq \Delta^2 g_{\mathbf{w}}(\mathbf{s}^-, \mathbf{t}^-)$.

In the final case where $\mathbf{s}$ is the lower expansion of $\mathbf{s}^-$ and $\mathbf{t}$ is the lower expansion of $\mathbf{t}^-$, then $\mathbf{s} \wedge \mathbf{t} = (\mathbf{s}^- \wedge \mathbf{t}^-)_+$ and $\mathbf{s} \vee \mathbf{t} = (\mathbf{s}^- \vee \mathbf{t}^-)_+$ (see Theorem 3.19). Moreover, the lower expansions giving $\mathbf{s} \wedge \mathbf{t}$ and $\mathbf{s} \vee \mathbf{t}$ cannot yield as large a $g_{\mathbf{w}}$ increase as the lower expansions giving $\mathbf{s}$ and $\mathbf{t}$, so again $\Delta^2 g_{\mathbf{w}}(\mathbf{s}, \mathbf{t}) \geq \Delta^2 g_{\mathbf{w}}(\mathbf{s}^-, \mathbf{t}^-)$. □

To see an example, the submodularity of $g_{\mathbf{w}}$ can be verified on the lattice for $n = 9$ and the weight sequence shown in Figure 5.1.

4.4. Submodularity as a discrete analogue of convexity. Although it is very simply defined, submodularity is difficult to appreciate. Using only standard vector calculus, we now clarify some basic relationships between submodularity and notions of convexity. We have not seen this done elsewhere.

There are several reasons why submodularity plays an important role here, at the crossroads between information and coding theory. First, submodularity is directly related to the Fortuin–Kasteleyn–Ginibre (FKG) "correlation" inequalities, which generalize a basic inequality of Tchebycheff on mean values of functions (hence expected values of random variables). A fine survey of results with FKG-like inequalities is [15].

Second, submodularity is closely related to convexity. Book-length surveys by Fujishige [9] and Narayanan [28] review connections between submodularity and optimization (and even electrical network theory). The relationship between convexity and submodularity was neatly summarized by Lovász with the following memorable definition and result.

DEFINITION 4.4. *Given a finite set S of cardinality n, we can identify a* $\{0,1\}$-vector $\mathbf{t} \in {\Re_+}^n$ *with any subset $T \subseteq S$ specifying the incidence in T of the elements in S (indexed in some fixed order).*

Any nonnegative vector $\mathbf{x} \in {\Re_+}^n$ *can be decomposed uniquely into a sum of positive real values multiplied by "decreasing"* $\{0,1\}$*-vectors. Specifically,* $\mathbf{x} \in {\Re_+}^n$ *determines an integer k $(0 \leq k \leq n)$ such that* $\mathbf{x}$ *has a* unique greedy decomposition

$$\mathbf{x} \quad = \quad \sum_{i=1}^{k} \lambda_i \, \mathbf{s}_i,$$

where $\lambda_i > 0$, $S_1 \supset \cdots \supset S_k$ are distinct subsets of S, and $\mathbf{s}_i$ is the $\{0,1\}$*-vector identified with S_i. For any function $f : S \to \Re_+$, its* greedy extension $\widehat{f} : {\Re_+}^n \to \Re_+$

to nonnegative vectors is then defined by

$$\widehat{f}(\mathbf{x}) \quad = \quad \widehat{f}\left(\sum_{i=1}^{k} \lambda_i \ \mathbf{s}_i\right) \quad = \quad \sum_{i=1}^{k} \lambda_i \ f(S_i).$$

In fact, $\boldsymbol{\lambda} = \boldsymbol{\partial}\,(\mathrm{sort}{\uparrow}\,(\,\mathbf{x}\,))$, using our notation.

THEOREM 4.5 (see Lovász [25, p. 249]). *$f : S \to \Re_+$ is submodular iff its greedy extension $\widehat{f} : {\Re_+}^n \to \Re_+$ is convex.*

Proof. The essence is that for positive constants $\lambda \leq \kappa$ and sets $T \neq U$,

$$\widehat{f}(\lambda\,\mathbf{t}{+}\kappa\,\mathbf{u}) \ = \ \lambda\,f(T{\cup}U) + (\kappa{-}\lambda)\,f(U) \ \leq \ \lambda\,f(T) + \kappa\,f(U) \ = \ \widehat{f}(\lambda\,\mathbf{t}) + \widehat{f}(\kappa\,\mathbf{u}),$$

where $\mathbf{t}$ and $\mathbf{u}$ are the $\{0,1\}$-vectors corresponding to T and U. The central inequality is due to submodularity. Resisting $0 < \lambda < 1$ and $\kappa = (1-\lambda)$ shows $\widehat{f}$ is convex. □

Lovász goes on [25, p. 250–251] to point out that

$$\min\,\{\ f(X) \mid X \ \subseteq \ S\ \} \quad = \quad \min\,\{\ \widehat{f}(\mathbf{x}) \mid \mathbf{x} \ \in \ [0,1]^n\ \}$$

and that as a consequence there is a polynomial-time algorithm to minimize f.

The vector lattice $\langle {\Re_+}^n, \leq_{vec}, \mathbf{min}_{vec}, \mathbf{max}_{vec}\rangle$, is exactly the extension of the set lattice to nonnegative vectors. Vector lattices, also called *Riesz spaces*, can be more "natural" than set lattices in some ways. For example, submodularity has a natural characterization.

THEOREM 4.6 (see Lorentz [27, p. 150]). *When twice differentiable, f is submodular on the vector lattice $\langle {\Re_+}^n, \leq_{vec}, \mathbf{min}_{vec}, \mathbf{max}_{vec}\rangle$ iff*

$$\frac{\partial^2 f}{\partial x_i \partial x_j} \quad \leq \quad 0 \qquad\qquad (i \neq j,\ 1 \leq i,\, j \leq n).$$

Proof. The proof is essentially by definition. Using the shorthand $f\langle\ u\ v\ \rangle$ to denote the expression $f(\langle\ x_1\ \ldots\ x_{i-1}\ \ u\ \ x_{i+1}\ \ldots\ x_{j-1}\ \ v\ \ x_{j+1}\ \ldots\ x_n\ \rangle)$ gives

$$\begin{aligned}\frac{\partial^2 f}{\partial x_i \partial x_j} &= \lim_{\epsilon_i, \epsilon_j \to 0} \frac{f\langle\ (x_i{+}\epsilon_i)\ (x_j{+}\epsilon_j)\ \rangle - f\langle\ (x_i{+}\epsilon_i)\ x_j\ \rangle - f\langle\ x_i\ (x_j{+}\epsilon_j)\ \rangle + f\langle\ x_i\ x_j\ \rangle}{\epsilon_i \ \ \epsilon_j} \\ &\leq 0,\end{aligned}$$

where the inequality comes from the fact that f is submodular, since with respect to $\leq_{vec}$, the points $\mathbf{x} = \langle\ (x_i{+}\epsilon_i)\ x_j\ \rangle$ and $\mathbf{y} = \langle\ x_i\ (x_j{+}\epsilon_j)\ \rangle$ have the upper and lower bounds $\mathbf{x}\ \mathbf{max}_{vec}\ \mathbf{y} = \langle\ (x_i{+}\epsilon_i)\ (x_j{+}\epsilon_j)\ \rangle$ and $\mathbf{x}\ \mathbf{min}_{vec}\ \mathbf{y} = \langle\ x_i\ x_j\ \rangle$. For the converse, if f is not submodular on a rectangle defined by $\mathbf{x} = \langle\ (x_i{+}a)\ x_j\ \rangle$ and $\mathbf{y} = \langle\ x_i\ (x_j{+}b)\ \rangle$, Lorentz pointed out we can find a subrectangle on which $\partial^2 f / \partial x_i \partial x_j > 0$. □

Note: the derivatives $\frac{\partial^2 f}{\partial x_i^2}$ can still be positive. In fact, the Hessian $\nabla^2 f = (\frac{\partial^2 f}{\partial x_i \partial x_j})$ still can even be positive semidefinite (hence f can be convex), or be an M-matrix [3, Chap. 6].

THEOREM 4.7. *When twice differentiable, F is submodular on the majorization lattice $\langle {\Re_+}^n, \preceq, \sqcap, \sqcup\rangle$ iff for all $i \neq j$ between 1 and $n-1$,*

$$\frac{\partial^2 F(\mathbf{z})}{\partial z_i \partial z_j} \ - \ \frac{\partial^2 F(\mathbf{z})}{\partial z_{i+1} \partial z_j} \ - \ \frac{\partial^2 F(\mathbf{z})}{\partial z_i \partial z_{j+1}} \ + \ \frac{\partial^2 F(\mathbf{z})}{\partial z_{i+1} \partial z_{j+1}} \quad \leq \quad 0.$$

Proof. Theorem 3.16 shows that the Möbius transformation $\boldsymbol{\partial}$ gives a bijection between the majorization lattice $\langle {\Re_+}^n, \preceq, \sqcap, \sqcup \rangle$ and the vector lattice $\langle {\Re_+}^n, \leq_{vec}, \mathbf{min}_{vec}, \mathbf{max}_{vec} \rangle$. Thus $f(\mathbf{x}) = F(\boldsymbol{\partial}\,\mathbf{x})$ is submodular on the vector lattice when F is submodular on the majorization lattice. Expanding the inequality

$$\frac{\partial^2}{\partial x_i \partial x_j}(F(\boldsymbol{\partial}\,\mathbf{x})) \quad = \quad \frac{\partial^2}{\partial x_i \partial x_j}(f(\mathbf{x})) \quad \leq \quad 0$$

(which follows from the previous theorem) with the chain rule gives the stated result, because $\mathbf{z} = \boldsymbol{\partial}\,\mathbf{x} = \langle\, x_1 \;\; (x_2 - x_1) \;\; (x_3 - x_2) \;\; \cdots \;\; (x_n - x_{n-1}) \,\rangle$. ∎

Revisiting Theorem 4.2, $G_{\mathbf{w}}(\mathbf{z}) = -\sum_{i=1}^{n} w_i \;\log_2(z_i)$ satisfies

$$\frac{\partial^2 G_{\mathbf{w}}(\mathbf{z})}{\partial z_i \partial z_j} - \frac{\partial^2 G_{\mathbf{w}}(\mathbf{z})}{\partial z_{i+1} \partial z_j} - \frac{\partial^2 G_{\mathbf{w}}(\mathbf{z})}{\partial z_i \partial z_{j+1}} + \frac{\partial^2 G_{\mathbf{w}}(\mathbf{z})}{\partial z_{i+1} \partial z_{j+1}}$$
$$= \frac{1}{\ln(2)} \begin{cases} 0, & |i-j| > 1, \\ -w_i / z_i^2, & i = j+1, \\ -w_{i+1} / z_{i+1}^2, & j = i+1, \\ w_{i+1}/z_{i+1}^2 + w_i / z_i^2, & i = j, \end{cases}$$

and, e.g., $G_{\mathbf{w}}(\boldsymbol{\partial}\,\mathbf{x}) = -w_1 \;\log_2(x_1) - \sum_{i=1}^{n-1} w_i \;\log_2(x_{i+1} - x_i)$ satisfies

$$\frac{\partial^2}{\partial x_i \partial x_j}(G_{\mathbf{w}}(\boldsymbol{\partial}\,\mathbf{x})) \quad = \quad \frac{1}{\ln(2)} \;\; \frac{-w_{i+1}}{(x_{i+1} - x_i)^2} \quad \leq \quad 0 \qquad \text{when } j = i+1.$$

This gives two alternative proofs of Theorem 4.2, showing how such results can be derived more easily.

DEFINITION 4.8. *A function $F : {\Re_+}^n \to \Re$ is* Schur convex *if it preserves the majorization ordering, i.e., $\mathbf{x} \preceq \mathbf{y}$ implies $F(\mathbf{x}) \leq F(\mathbf{y})$.*

THEOREM 4.9. *F is Schur convex on the majorization lattice iff $f(\mathbf{x}) = F(\boldsymbol{\partial}\,\mathbf{x})$ is monotone on the vector lattice.*

Proof. Again a direct result of the bijection between the two lattices. If f is differentiable, f is monotone on the vector lattice iff $\nabla f(\mathbf{x}) = \langle\, \partial f/\partial x_1 \cdots \partial f/\partial x_n \,\rangle \geq_{vec} \mathbf{0}$, which implies

$$\frac{\partial F}{\partial x_i}(\boldsymbol{\partial}\,\mathbf{x}) - \frac{\partial F}{\partial x_{i+1}}(\boldsymbol{\partial}\,\mathbf{x}) \quad = \quad \frac{\partial}{\partial x_i}(F(\boldsymbol{\partial}\,\mathbf{x})) \quad = \quad \frac{\partial}{\partial x_i} f(\mathbf{x}) \quad \geq \quad 0 \qquad (1 \leq i \leq n-1).$$

This rederives the result that $\partial F/\partial z_i \geq \partial F/\partial z_{i+1}$ when F is Schur convex [34]. ∎

Since monotonicity and convexity are related, Theorems 4.5, 4.6, 4.7, and 4.9 connect convexity, submodularity, Schur convexity, and majorization. There are actually many connections. See the survey [27, Chap. 6], in which submodular functions are called *$\mathcal{L}$-subadditive functions*. Just as Lovász showed for submodular functions [25], Schur convex functions [37, 29] are closed under various operations: min, max, convolution, composition with convex functions, etc. [27, Chap. 3]. Theorem 4.5 is also reminiscent of *symmetric gauge* functions, which are Schur convex; see [27, p. 96].

5. Huffman coding as submodular dynamic programming. The results of the previous sections can now be applied to Huffman coding.

5.1. Nonmonotonicity of weighted path-length over the lattices. It is important to realize that weighted path-length is not monotone on the imbalance lattice, so greedy search may not always find its way to an optimal solution. This is illustrated by the example in Figure 5.1. For this problem the sequence $\langle\, 2\,2\,3\,3\,4\,4\,4\,5\,5 \,\rangle$ with cost 1298 is a local minimum: each of the 7 sequences reachable from it by imbalancing exchanges and each of the 3 sequences reachable from it by balancing exchanges have greater weighted path length. The diagram shows only the transitive reduction of the imbalance lattice, omitting many balancing exchanges (because they would clutter the picture), but it conveys the general situation for larger Huffman coding problems. It shows that, even though it may do very well in practice, simple hill-climbing along ternary exchanges is not guaranteed to find the optimum sequence.

Although weighted path-length $g_{\mathbf{w}}$ is not monotone on the imbalance lattice of trees, a monotone summary of weighted path-length ${g_{\mathbf{w}}}^{mon}$ has the properties we need.

In [25, p. 241], Lovász stated the following definition and theorem for set lattices (easily proved for general lattices) about the "monotonization" of a function f.

DEFINITION 5.1. *If $f : \mathcal{L} \to \Re_+$ is a real-valued function on a lattice $\mathcal{L}$ with ordering relation $\sqsubseteq$, define*

$$f^{mon}(\mathbf{x}) \quad = \quad \min\ \{\ f(\mathbf{x}') \mid \mathbf{x}' \sqsubseteq \mathbf{x}\ \}.$$

THEOREM 5.2. *If f is submodular, then f^{mon} is also submodular.*

Proof. From the definition of f^{mon}, for all $\mathbf{x}$, $\mathbf{y}$ in $\mathcal{L}$, there exists a $\mathbf{x}' \sqsubseteq \mathbf{x}$ such that $f^{mon}(\mathbf{x}) = f(\mathbf{x}')$ and a $\mathbf{y}' \sqsubseteq \mathbf{y}$ such that $f^{mon}(\mathbf{y}) = f(\mathbf{y}')$. But then $(\mathbf{x}' \sqcap \mathbf{y}') \sqsubseteq (\mathbf{x} \sqcap \mathbf{y})$ and $(\mathbf{x}' \sqcup \mathbf{y}') \sqsubseteq (\mathbf{x} \sqcup \mathbf{y})$, so

$$\begin{array}{rcccl} f^{mon}(\mathbf{x} \sqcap \mathbf{y}) & \le & f^{mon}(\mathbf{x}' \sqcap \mathbf{y}') & \le & f(\mathbf{x}' \sqcap \mathbf{y}'), \\ f^{mon}(\mathbf{x} \sqcup \mathbf{y}) & \le & f^{mon}(\mathbf{x}' \sqcup \mathbf{y}') & \le & f(\mathbf{x}' \sqcup \mathbf{y}'). \end{array}$$

$$\begin{array}{rcccl} f^{mon}(\mathbf{x} \sqcap \mathbf{y}) \ + \ f^{mon}(\mathbf{x} \sqcup \mathbf{y}) & \le & f(\mathbf{x}' \sqcap \mathbf{y}') \ + \ f(\mathbf{x}' \sqcup \mathbf{y}') & & \\ & \le & f(\mathbf{x}') \ + \ f(\mathbf{y}') & & \text{(as } f \text{ is submodular)} \\ & = & f^{mon}(\mathbf{x}) \ + \ f^{mon}(\mathbf{y}). & & \square \end{array}$$

Thus ${g_{\mathbf{w}}}^{mon}$ is both submodular and monotone on the tree imbalance lattice.

5.2. Dynamic programming reconstruction of the Huffman algorithm. Based the analysis above, we can derive Huffman codes by using a simple recursion.

DEFINITION 5.3. *The* Huffman contraction $\mathbf{w}^{\frown}$ *of a descending weight sequence* $\mathbf{w} = \langle\, w_1 \cdots w_n \,\rangle$ *is*

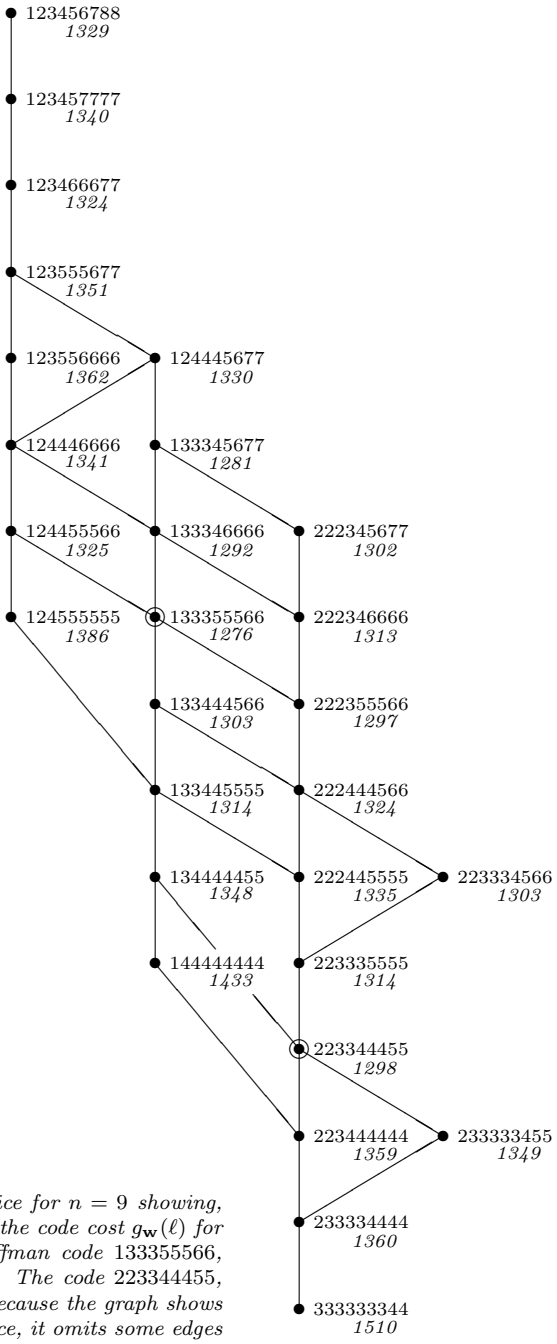
$$\mathbf{w}^{\frown} \quad = \quad \mathrm{sort}{\downarrow}\,(\,\langle\ w_1\ \cdots\ w_{n-2}\ (w_{n-1}{+}w_n)\ \rangle\,).$$

Parenthetically, note that $\boldsymbol{\ell}^{-} \ = \ -\log_2(\,(2^{-\boldsymbol{\ell}})^{\frown}\,)$ *for path-length sequences* $\boldsymbol{\ell}$. *If* $n \ge 1$ *is the length of* $\mathbf{w}$, *then the (most balanced)* Huffman code *for* $\mathbf{w}$ *is defined by*

$$\mathit{Huffman}(\mathbf{w}) \quad = \quad \begin{cases} \langle\ 0\ \rangle & \text{if } \ n = 1, \\ \mathit{better\_expansion}(\ \mathit{Huffman}(\ \mathbf{w}^{\frown}\ )\ , \mathbf{w}) & \text{if } \ n > 1; \end{cases}$$

$$\mathit{better\_expansion}(\ \boldsymbol{\ell}\ , \mathbf{w}) \quad = \quad \begin{cases} \boldsymbol{\ell}_+ & \text{if } \ g_{\mathbf{w}}(\ \boldsymbol{\ell}_+\ ) \ \le \ g_{\mathbf{w}}(\ \boldsymbol{\ell}^+\ ), \\ \boldsymbol{\ell}^+ & \text{otherwise.} \end{cases}$$

Code *Path-length Sequence* ℓ	*Cost* *Weighted Path-length* $g_{\mathbf{w}}(\ell)$
123456788	*1329*
123457777	*1340*
123466677	*1324*
123555677	*1351*
123556666	*1362*
124445677	*1330*
124446666	*1341*
124455566	*1325*
124555555	*1386*
133345677	*1281*
133346666	*1292*
133355566	*1276*
133444566	*1303*
133445555	*1314*
134444455	*1348*
144444444	*1433*
222345677	*1302*
222346666	*1313*
222355566	*1297*
222444566	*1324*
222445555	*1335*
223334566	*1303*
223335555	*1314*
223344455	*1298*
223444444	*1359*
233333455	*1349*
233334444	*1360*
333333344	*1510*



The transitively reduced imbalance lattice for $n = 9$ showing, for $\mathbf{w} = \langle$ 189 95 73 71 23 21 18 9 1 $\rangle$, the code cost $g_{\mathbf{w}}(\ell)$ for each path-length sequence ℓ. The Huffman code 133355566*, with cost* 1276*, is the global minimum. The code* 223344455*, with cost* 1298*, is a local minimum. (Because the graph shows only the transitive reduction of the lattice, it omits some edges corresponding to exchanges, but the minimum is localized.)*

FIG. 5.1. *Costs of all possible codes for the weights* $\mathbf{w} = \langle$ 189 95 73 71 23 21 18 9 1 $\rangle$.

For example, the example in Figure 5.1 can be traced through Figure 3.3 and Table 2.2:

$$\begin{array}{lcl}
\mathit{Huffman}(\ \langle\ 189\ 95\ 73\ 71\ 23\ 21\ 18\ 9\ 1\ \rangle\ ) & = & \langle\ 1\ 3\ 3\ 3\ 5\ 5\ 5\ 6\ 6\ \rangle, \\
\mathit{Huffman}(\ \langle\ 189\ 95\ 73\ 71\ 23\ 21\ 18\ 10\ \rangle\ ) & = & \langle\ 1\ 3\ 3\ 3\ 5\ 5\ 5\ 5\ \rangle, \\
\mathit{Huffman}(\ \langle\ 189\ 95\ 73\ 71\ 28\ 23\ 21\ \rangle\ ) & = & \langle\ 1\ 3\ 3\ 3\ 4\ 5\ 5\ \rangle, \\
\mathit{Huffman}(\ \langle\ 189\ 95\ 73\ 71\ 44\ 28\ \rangle\ ) & = & \langle\ 1\ 3\ 3\ 3\ 4\ 4\ \rangle, \\
\mathit{Huffman}(\ \langle\ 189\ 95\ 73\ 72\ 71\ \rangle\ ) & = & \langle\ 1\ 3\ 3\ 3\ 3\ \rangle, \\
\mathit{Huffman}(\ \langle\ 189\ 143\ 95\ 73\ \rangle\ ) & = & \langle\ 1\ 2\ 3\ 3\ \rangle, \\
\mathit{Huffman}(\ \langle\ 189\ 168\ 143\ \rangle\ ) & = & \langle\ 1\ 2\ 2\ \rangle, \\
\mathit{Huffman}(\ \langle\ 311\ 189\ \rangle\ ) & = & \langle\ 1\ 1\ \rangle, \\
\mathit{Huffman}(\ \langle\ 500\ \rangle\ ) & = & \langle\ 0\ \rangle.
\end{array}$$

This dynamic programming definition is similar to the standard Huffman algorithm, but it differs in a few ways. First, it considers only upper and lower expansions of $\mathit{Huffman}(\mathbf{w}^\frown)$; we prove momentarily that this is sufficient. Second, it produces a *unique* Huffman code, which is the most balanced possible because it uses lower expansions whenever possible. (This avoids the issue of multiplicity of solutions that arises in implementation of the standard Huffman algorithm. For example, if $\mathbf{w} = \langle\ 3\ 2\ 2\ 1\ \rangle$, then $\langle\ 1\ 2\ 3\ 3\ \rangle$ is a Huffman code, but the more balanced sequence $\langle\ 2\ 2\ 2\ 2\ \rangle$ is also and will be produced by the algorithm above.)

Actually, the definition of $\mathit{Huffman}(\mathbf{w})$ can be "simplified" somewhat. Note that when

$$\boldsymbol{\ell} \quad = \quad \mathit{Huffman}(\mathbf{w}^\frown) \quad = \quad \langle\ \cdots\ (q-j)\ \overbrace{q\ \cdots\ q}^{2k}\ \rangle$$

with suffix length $2k$ and suffix increment j, the condition on $\mathit{better\_expansion}(\boldsymbol{\ell}, \mathbf{w})$ is

$$\begin{array}{lrcl}
 & g_{\mathbf{w}}(\boldsymbol{\ell}_+) & \le & g_{\mathbf{w}}(\boldsymbol{\ell}^+) \\
\Leftrightarrow \quad g_{\mathbf{w}}(\boldsymbol{\ell}) + w_{n-2k-1} - (j-1)\,w_{n-2k} + q\,w_n & \le & g_{\mathbf{w}}(\boldsymbol{\ell}) + w_{n-1} + (q+1)\,w_n \\
\Leftrightarrow \quad w_{n-2k-1} - (j-1)\,w_{n-2k} & \le & w_{n-1} + w_n.
\end{array}$$

Going further, the proof of Theorem 5.5 below implies that this condition actually can be simplified to take $j = 1$, so that $\mathit{better\_expansion}(\boldsymbol{\ell}, \mathbf{w}) = \boldsymbol{\ell}_+$ if $w_{n-2k-1} \le w_{n-1} + w_n$.

THEOREM 5.4. *$\mathit{Huffman}(\mathbf{w})$ is the most balanced optimal code for $\mathbf{w}$.*

Proof. Let $\mathbf{s} = \mathit{Huffman}(\mathbf{w})$, so that $\mathbf{s}$ is the cheaper of $\mathit{Huffman}(\mathbf{w}^\frown)_+$ and $\mathit{Huffman}(\mathbf{w}^\frown)^+$, or is the former (which is more balanced) if they have equal cost.

Only these two expansions need be considered. Like the usual Huffman algorithm, this algorithm assigns w_{n-1} and w_n maximal path length. Therefore $(w_{n-1} + w_n)$ must appear in the "suffix" of $\mathbf{w}^\frown$, i.e., among the $2k+1$ final entries, where $2k$ is the suffix length of $\boldsymbol{\ell} = \mathit{Huffman}(\mathbf{w}^\frown)$; and so it has path length either $(q-1)$ or q, corresponding to the two possible expansions. Thus only Huffman codes are derived with the algorithm above.

Submodularity of ${g_{\mathbf{w}}}^{mon}$ now proves that there is a unique most balanced Huffman code (and thus greedy search will find this code). Suppose that $\mathbf{s}$ and $\mathbf{t}$ are maximally balanced Huffman codes that are noncomparable in the balance ordering. Then ${g_{\mathbf{w}}}^{mon}(\mathbf{t}) = g_{\mathbf{w}}(\mathbf{t}) = g_{\mathbf{w}}(\mathbf{s})$. Because $\mathbf{t}$ is optimal ${g_{\mathbf{w}}}^{mon}(\mathbf{s} \vee \mathbf{t}) = {g_{\mathbf{w}}}^{mon}(\mathbf{t})$.

Submodularity of ${g_{\mathbf{w}}}^{mon}$ then implies that ${g_{\mathbf{w}}}^{mon}(\mathbf{s} \wedge \mathbf{t}) \leq {g_{\mathbf{w}}}^{mon}(\mathbf{s})$. By the definition of ${g_{\mathbf{w}}}^{mon}$, ${g_{\mathbf{w}}}^{mon}(\mathbf{s} \wedge \mathbf{t}) = {g_{\mathbf{w}}}^{mon}(\mathbf{s})$. But then $\mathbf{s} \wedge \mathbf{t}$ is optimal—hence a Huffman code—and it is more balanced than both $\mathbf{s}$ and $\mathbf{t}$. This gives a contradiction. □

THEOREM 5.5. *Huffman*$(\mathbf{w}^\frown) = $ *Huffman*$(\mathbf{w})^-$.

Proof. We prove this by induction on the length n of $\mathbf{w}$. The base case $n = 2$ is trivial.

For the induction step, let $\mathbf{s} =$ *Huffman*$(\mathbf{w})$, so by the previous theorem $\mathbf{s}$ is the most balanced optimal code for $\mathbf{w}$. Let $\boldsymbol{\ell} =$ *Huffman*$(\mathbf{w}^\frown)$. Since $\mathbf{w}^\frown$ has length $(n-1)$, $\boldsymbol{\ell}$ is the most balanced optimal code for $\mathbf{w}^\frown$ by the induction hypothesis. By definition $\mathbf{s}$ is the better (cheaper or more balanced if equally cheap) of $\boldsymbol{\ell}^+$ or $\boldsymbol{\ell}_+$. We consider two possibilities.

First, if $\mathbf{s} = \boldsymbol{\ell}^+$, then $\boldsymbol{\ell} = \mathbf{s}^-$ as required, because $\boldsymbol{\ell} = (\boldsymbol{\ell}^+)^-$ by Theorem 3.9.

Second, if $\mathbf{s} = \boldsymbol{\ell}_+$, then $w_{n-2k-1} - (j-1)\, w_{n-2k} \;\leq\; w_{n-1} + w_n$, where $2k$ is the suffix length of $\boldsymbol{\ell}$, j is its suffix increment, and by Theorem 3.1, $j \leq \log_2(2k)$ or equivalently $2^{j-1} \leq k$. If $j = 1$ we find again $\boldsymbol{\ell} = \mathbf{s}^-$ as required.

We now claim that $j > 1$ cannot arise in this second possibility where $\mathbf{s} = \boldsymbol{\ell}_+$. Let us first understand intuitively why this is so. When the suffix length $j > 1$, $\boldsymbol{\ell} = \langle\, \cdots\ (q-j)\ q\ \cdots\ q\, \rangle$ describes a tree that is perfectly balanced over its suffix, but the rest is at least j levels shorter. The Huffman algorithm will construct such a tree only when the final $2k$ weights of $\mathbf{w}^\frown$ are all of similar size, but w_{n-2k-1} is much larger. Specifically, $w_{n-1} + w_n < w_{n-2k-1}$, and w_{n-2k-1} is constrained to be 2^{j-1} larger than the sum of the subsequent weights, or the Huffman algorithm would construct a different tree. But w_{n-2k-1} is also constrained to be small by the inequality in the definition of the Huffman algorithm; if it becomes too large, we get $\mathbf{s} = \boldsymbol{\ell}^+$ instead of $\mathbf{s} = \boldsymbol{\ell}_+$. These two constraints turn out not to be simultaneously satisfiable when $j > 1$.

Suppose that $j > 1$ and $\mathbf{s} = \boldsymbol{\ell}_+$; then, since $w_{n-1} + w_n < w_{n-2k-1}$ we have two cases.

1. $w_{n-3} \geq w_{n-1} + w_n$.
   Let $W = w_{n-3}$. Then $w_{n-2k-1} > w_{n-2k} + (2^{j-1}-1)\, W$ and $2W \geq w_{n-2k} \geq W$, since $j > 1$ implies $2k \geq 4$ by Theorem 3.1, $\ell_{n-2k-1} = (q-j)$, and $\boldsymbol{\ell} =$ *Huffman*$(\mathbf{w}^\frown)$ is constructed by the Huffman algorithm. These give the first bound
   $$\frac{w_{n-2k-1}}{w_{n-2k}} \quad > \quad 1 \;+\; (2^{j-1}-1)/2 \quad = \quad 2^{j-2} + 1/2.$$
   However, from $w_{n-2k-1} - (j-1)\, w_{n-2k} \;\leq\; w_{n-1} + w_n$ it follows that
   $$\begin{aligned} w_{n-2k-1} &> w_{n-2k} \;+\; (2^{j-1}-1)\, W \\ &\geq w_{n-2k} \;+\; (2^{j-1}-1)\, (w_{n-1} + w_n) \\ &\geq w_{n-2k} \;+\; (2^{j-1}-1)\, (w_{n-2k-1} - (j-1)\, w_{n-2k}). \end{aligned}$$
   When $j = 2$, this simplifies to $w_{n-2k-1} > w_{n-2k-1}$, a contradiction. When $j > 2$, it gives the second bound
   $$\frac{w_{n-2k-1}}{w_{n-2k}} \quad < \quad \frac{(j-1) \;-\; 1/(2^{j-1}-1)}{1 \;-\; 1/(2^{j-1}-1)}.$$
   However this contradicts the first bound for all $j > 2$.
2. $w_{n-2k-1} > w_{n-1} + w_n > w_{n-3}$.
   This is like the previous case, but this time when $W = w_{n-3}$ we can derive

only the weaker condition $W > (w_{n-1} + w_n)/2$, because $w_{n-3} \geq w_{n-1} \geq w_n$. Still, the same first bound, essentially the second bound (with $(2^{j-1} - 1)$ divided by 2), and the same contradictions, are derivable.

Thus all cases reach a contradiction, implying as required that, in the second possibility, $j = 1$ and $\boldsymbol{\ell} = \mathbf{s}^-$. □

5.3. The importance of submodularity in dynamic programming. Together, Theorems 5.4 and 5.5 show that Huffman coding (finding the most balanced code that minimizes $g_{\mathbf{w}}{}^{mon}$) is a dynamic programming problem that can be solved in various ways, because the problem enjoys elegant recursive properties.

Huffman coding gives another example of a dynamic programming problem that can be sped up considerably because the objective function is submodular over the solution space. Lawler [24] remarked:

> *If a discrete optimization problem can be solved efficiently, it is quite likely that submodularity is responsible. In recent years there has been a growing appreciation of the fact that submodularity plays a pivotal role in discrete optimization, not unlike that of convexity in continuous optimization.*

Submodularity has a long history in dynamic programming. By 1781, Monge had found a form of submodularity to be important in simplifying the transportation problem [17]. In 1970, Edmonds [6] related submodularity to matroids and greedily solvable optimization problems. In 1980, Yao [42] generalized upon Knuth's famous $O(n^2)$ algorithm for optimum binary search trees [21] by giving an $O(n^2)$ algorithm for the dynamic programming problem

$$\begin{aligned} c(i,i) &= 0, \\ c(i,j) &= w(i,j) + \min_{i<k\leq j} \left(c(i,k-1) + c(k,j) \right) \quad (i < j); \end{aligned}$$

$$\begin{aligned} w(i,j) &\leq w(i',j') && (i' \leq i \leq j \leq j'), \\ w(i,j) + w(i',j') &\leq w(i',j) + w(i,j') && (i \leq i' \leq j \leq j'). \end{aligned}$$

Yao called the final constraint the *quadrangle inequality*, noting that it implies the (inverse) triangle inequality. Writing $I = [i, j]$ and $J = [i', j']$, defining a lattice of intervals of indices in the dynamic programming array, these two constraints require the function $W([a, b]) = -w(a, b)$ to be *monotone decreasing* ($W(I) \geq W(J)$ if $I \subseteq J$) and *submodular* ($W(I) + W(J) \geq W(I \cap J) + W(I \cup J)$). Results from exploiting the quadrangle inequality in dynamic programming appear in [2, 7, 35] for problems ranging from DNA sequencing to minimum cost matching.

Mirroring Theorem 4.7, the *Monge condition* $w_{i,j} + w_{i+1,j+1} \leq w_{i+1,j} + w_{i,j+1}$ on an $n \times n$ weight matrix W is also equivalent to the requirement that, ignoring its first column and row, the matrix $\boldsymbol{\partial}\, W\, \boldsymbol{\partial}^{\top}$ is nonpositive. Burkard, Klinz, and Rudolf [4], compiled a comprehensive survey of many incarnations of the Monge condition.

Recently Klein [20] explored the connection between dynamic programming and submodularity. Golin and Rote [14] developed dynamic programming algorithms for prefix codes when the codeword letters have differing costs, a useful case not handled by Huffman's algorithm; they recently extended this work to exploit the Monge property.

6. Other applications. The results here also can be used to gain further insight about submodular dynamic programming, the Huffman coding problem, and perhaps

also about the applications of lattice concepts in coding. Almost all of the theorems proved here admit interesting extensions and/or special cases. For example, a direct corollary of Theorem 4.3 (using $\mathbf{w} = \langle\, 1 \cdots 1 \,\rangle$) is that the function mapping a path-length sequence to its level of balance is submodular on the imbalance lattice. It would be interesting to extend the work here for the t-ary codes discussed in [19].

Majorization, we believe, can be exploited further in characterizing optimal codes. We have established that the imbalance ordering on tree path-length sequences $\boldsymbol{\ell}$ is isomorphic to the majorization ordering on exponentiated tree path-length sequences $\mathbf{x} = 2^{-\boldsymbol{\ell}}$. Thus any function that is Schur convex (i.e., "majorization-preserving": monotone with respect to the majorization ordering) on exponentiated path-length sequences and hence monotone on the (continuous) majorization lattice will also be monotone on the (discrete) imbalance lattice. Negative entropy is an important example of such a function; related functions are discussed in [32].

Furthermore, the methods developed above hold out hope for entirely new approaches to Huffman coding. We sketch two possibilities.

6.1. Continuous approximation of Huffman codes. One possibility is that we can attack the combinatorial problem of Huffman coding with a continuous, real-valued optimization problem. Recall that Huffman coding can be expressed as an optimization problem:

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{n} w_i\, \ell_i \\ &\text{subject to} && \sum_{i=1}^{n} 2^{-\ell_i} \;=\; 1, \qquad \ell_i > 0, \;\; \text{integer} \;\; (1 \le i \le n). \end{aligned}$$

Dropping the integrality constraint gives an interesting continuous relaxation of Huffman coding that can be attacked numerically. For example, by treating the constraint as a penalty function, the problem above can be solved numerically with something like the system of equations $\partial/\partial\ell_j \left(\sum_{i=1}^{n} w_i\, \ell_i \;+\; 10^{10} \left(1 \;-\; \sum_{i=1}^{n} 2^{-\ell_i} \right)^2 \right) \;=\; 0$ $(1 \le j \le n)$. Using the example weight sequence $\mathbf{w} = \langle\, 189\ 95\ 73\ 71\ 23\ 21\ 18\ 9\ 1 \,\rangle$ studied earlier, a simple program found a unique real solution

$$\boldsymbol{\ell} \approx \langle\, 1.4\ 2.4\ 2.8\ 2.8\ 4.4\ 4.8\ 4.8\ 5.8\ 9.0 \,\rangle$$

for these equations, with objective ≈ 1241. As expected, this solution is near the optimal Huffman code $\langle\, 1\,3\,3\,3\,5\,5\,5\,6\,6 \,\rangle$, with cost 1276.

When the relaxation is faithful to the original, it will be possible to find optimal solutions quickly. The relaxed solution can be used to jump to the right neighborhood in the imbalance lattice, from which balancing exchanges will walk to the optimal code. The penalty function could clearly be varied, and perhaps could be changed to encourage near-integral solutions.

Interior point methods on the majorization lattice may also be possible. Among other things, it may be possible to define $\mathbf{s} \wedge \mathbf{t}$ in terms of $-\log_2 \left(2^{-\mathbf{s}} \sqcap 2^{-\mathbf{t}} \right)$ and $\mathbf{s} \vee \mathbf{t}$ in terms of $-\log_2 \left(2^{-\mathbf{s}} \sqcup 2^{-\mathbf{t}} \right)$: they are often identical and always satisfy

$$2^{-\mathbf{s} \wedge \mathbf{t}} \;\preceq\; 2^{-\mathbf{s}} \sqcap 2^{-\mathbf{t}}, \qquad\qquad 2^{-\mathbf{s}} \sqcup 2^{-\mathbf{t}} \;\preceq\; 2^{-\mathbf{s} \vee \mathbf{t}}$$

(because $2^{-\mathbf{s}} \sqcap 2^{-\mathbf{t}}$ and $2^{-\mathbf{s}} \sqcup 2^{-\mathbf{t}}$ are the glb and lub with respect to majorization).

For perspective, if $\alpha = 7 - \log_2(12) \approx 3.4150375$ and $\beta = (\alpha - 1)$, the following set of examples represent the unusual cases with $n = 9$ where $-\log_2 \left(2^{-\mathbf{s}} \;\sqcap\; 2^{-\mathbf{t}} \right) \neq$

$(\mathbf{s} \wedge \mathbf{t})$ or $-\log_2\left(2^{-\mathbf{s}} \sqcup 2^{-\mathbf{t}}\right) \neq (\mathbf{s} \vee \mathbf{t})$.

$\mathbf{s}$	$\mathbf{t}$	$-\log_2\left(2^{-\mathbf{s}} \sqcap 2^{-\mathbf{t}}\right)$	$\mathbf{s} \wedge \mathbf{t}$
$\langle\ 1\ 2\ 4\ 5\ 5\ 5\ 5\ 5\ 5\ \rangle$	$\langle\ 1\ 3\ 3\ 3\ 4\ 5\ 6\ 7\ 7\ \rangle$	$\langle\ 1\ 3\ 3\ \alpha\ 5\ 5\ 5\ 5\ 5\ \rangle$	$\langle\ 1\ 3\ 3\ 3\ 4\ 4\ 5\ 5\ 5\ 5\ \rangle$
$\langle\ 1\ 2\ 4\ 5\ 5\ 5\ 5\ 5\ 5\ \rangle$	$\langle\ 2\ 2\ 2\ 3\ 4\ 5\ 6\ 7\ 7\ \rangle$	$\langle\ 2\ 2\ 2\ \alpha\ 5\ 5\ 5\ 5\ 5\ \rangle$	$\langle\ 2\ 2\ 2\ 4\ 4\ 5\ 5\ 5\ 5\ \rangle$
$\langle\ 1\ 3\ 4\ 4\ 4\ 4\ 4\ 5\ 5\ \rangle$	$\langle\ 2\ 2\ 2\ 3\ 4\ 5\ 6\ 7\ 7\ \rangle$	$\langle\ 2\ 2\ \beta\ 4\ 4\ 4\ 4\ 5\ 5\ \rangle$	$\langle\ 2\ 2\ 3\ 3\ 4\ 4\ 4\ 5\ 5\ \rangle$
$\mathbf{s}$	$\mathbf{t}$	$-\log_2\left(2^{-\mathbf{s}} \sqcup 2^{-\mathbf{t}}\right)$	$\mathbf{s} \vee \mathbf{t}$
$\langle\ 1\ 4\ 4\ 4\ 4\ 4\ 4\ 4\ 4\ \rangle$	$\langle\ 2\ 2\ 2\ 3\ 4\ 5\ 6\ 7\ 7\ \rangle$	$\langle\ 1\ 4\ \beta\ 3\ 4\ 5\ 6\ 7\ 7\ \rangle$	$\langle\ 1\ 3\ 3\ 3\ 4\ 5\ 6\ 7\ 7\ \rangle$
$\langle\ 1\ 4\ 4\ 4\ 4\ 4\ 4\ 4\ 4\ \rangle$	$\langle\ 2\ 2\ 2\ 3\ 4\ 6\ 6\ 6\ 6\ \rangle$	$\langle\ 1\ 4\ \beta\ 3\ 4\ 6\ 6\ 6\ 6\ \rangle$	$\langle\ 1\ 3\ 3\ 3\ 4\ 6\ 6\ 6\ 6\ \rangle$
$\langle\ 1\ 4\ 4\ 4\ 4\ 4\ 4\ 4\ 4\ \rangle$	$\langle\ 2\ 2\ 2\ 3\ 5\ 5\ 5\ 6\ 6\ \rangle$	$\langle\ 1\ 4\ \beta\ 3\ 5\ 5\ 5\ 6\ 6\ \rangle$	$\langle\ 1\ 3\ 3\ 3\ 5\ 5\ 5\ 6\ 6\ \rangle$
$\langle\ 1\ 4\ 4\ 4\ 4\ 4\ 4\ 4\ 4\ \rangle$	$\langle\ 2\ 2\ 2\ 4\ 4\ 4\ 5\ 6\ 6\ \rangle$	$\langle\ 1\ 4\ \beta\ 4\ 4\ 4\ 5\ 6\ 6\ \rangle$	$\langle\ 1\ 3\ 3\ 4\ 4\ 4\ 5\ 6\ 6\ \rangle$
$\langle\ 1\ 4\ 4\ 4\ 4\ 4\ 4\ 4\ 4\ \rangle$	$\langle\ 2\ 2\ 2\ 4\ 4\ 5\ 5\ 5\ 5\ \rangle$	$\langle\ 1\ 4\ \beta\ 4\ 4\ 5\ 5\ 5\ 5\ \rangle$	$\langle\ 1\ 3\ 3\ 4\ 4\ 5\ 5\ 5\ 5\ \rangle$

These examples suggest there may be algorithms that "round up" $-\log_2\left(2^{-\mathbf{s}} \sqcap 2^{-\mathbf{t}}\right)$ to give $\mathbf{s} \wedge \mathbf{t}$, and "round down" $-\log_2\left(2^{-\mathbf{s}} \sqcup 2^{-\mathbf{t}}\right)$ to give $\mathbf{s} \vee \mathbf{t}$.

6.2. Practical applications in adaptive coding. In many practical situations it is difficult or impossible to know a priori the weights $\mathbf{w}$ used in Huffman coding. A natural idea, which occurred independently to Faller [8] and Gallager [11], is to allow the weights to be determined dynamically and to have the Huffman code "evolve" over time. *Dynamic Huffman coding* is the strategy of repeatedly constructing the Huffman code for the input so far and using it in transmitting the next input symbol. Knuth presented an efficient algorithm for dynamic Huffman coding in [22], and his performance results for the algorithm show it consistently producing compression very near (though not surpassing) the compression attained with static Huffman code for the entire input.

Vitter [40, 41] then developed a dynamic Huffman algorithm that improves on Knuth's in the following way: rather than simply revise the Huffman tree after each input symbol, Vitter also finds a new Huffman tree of minimal external path length $\sum_i \ell_i$ and height $\max_i \ell_i$. With this modification Vitter was actually able to surpass the performance of static Huffman coding on several benchmarks.

A small contribution we can make is to clarify the improvement of Vitter. Basically, Vitter's algorithm differs from Knuth's in *constructing the optimal path-length sequence that is also as balanced as possible.* Note that minimizing the external path length $\sum_i \ell_i$ is identical to maximizing the level of balance. Since there can be more than one optimal code, and unnecessary imbalance tends to penalize the symbol currently being encoded, insisting on maximally balanced codes improves performance.

Another contribution of the lattice perspective here is to encourage development of new adaptive coding schemes. As suggested in section 5.1, a move between adjacent points in the lattice corresponds to minor alteration of codes, and by moving through the lattice we incrementally modify the cost of a code. Hill-climbing then gives greedy coding algorithms, and online hill-climbing gives adaptive coding algorithms. Although we have shown that the codes produced by hill-climbing are not guaranteed to be optimal, lattice-oriented adaptive coding algorithms may still have a role to play in some coding situations, since the Huffman notion of optimality is not really what is needed in the (currently popular and enormously important) adaptive context.

For example, adaptive coding algorithms can start at any point in the lattice, as long as both ends of the communication know which one. Rather than rely on the dynamic Huffman algorithm to derive reasonable operating points for the code, or rely on Knuth's "windowed" algorithm [22], one can immediately begin with a mutually

agreed upon, "reasonable" initial code (depending on the type of information being transmitted), and then adapt this code using some mutually agreed upon greedy algorithm for moving in the imbalance lattice.

Acknowledgments. We are very grateful to Pierre Hasenfratz for insightful comments that improved this paper. A conversation with Mordecai Golin, who provided us with an expanded version of [14], inspired us to discuss dynamic programming explicitly in this paper. He also pointed out the survey [4] to us. Also, we are indebted to two anonymous referees for clarifications of the exposition, especially of the significance of submodularity and of Shannon's work [38].

REFERENCES

[1] J. Abrahams, *Code and parse trees for lossless source encoding*, in Proc. Compression & Complexity of Sequences (SEQUENCES'97), Positano, Italy, 1997, IEEE Press, Piscataway, NJ, to appear.

[2] A. Aggarwal, A. Bar-Noy, S. Khuller, D. Kravets, and B. Schieber, *Efficient minimum cost matching and transportation using the quadrangle inequality*, J. Algorithms, 19 (1995), pp. 116–143.

[3] A. Berman and R.J. Plemmons, *Nonnegative matrices in the mathematical sciences*, SIAM, Philadelphia, PA, 1994.

[4] R.E. Burkard, B. Klinz, and R. Rudolf, *Perspectives of Monge properties in optimization*, Discrete Appl. Math., 70 (1996), pp. 95–161.

[5] B.A. Davey and H.A. Priestley, *Introduction to Lattices and Order*, Cambridge University Press, Cambridge, UK, 1990.

[6] J. Edmonds, *Submodular Functions, Matroids and Certain Polyhedra*, in Combinatorial Structures and Their Applications, R. Guy et al., eds., Gordon & Breach, New York, 1970, pp. 69–87.

[7] D. Eppstein, Z. Galil, R. Giancarlo, and G.F. Italiano, *Sparse dynamic programming.* II. *Convex and concave cost functions*, J. ACM, 39 (1992), pp. 546–567.

[8] N. Faller, *An adaptive system for data compression*, Record of the 7th Asilomar Conference on Circuits, Systems, and Computers, Pacific Grove, CA, 1973, pp. 593–597.

[9] S. Fujishige, *Submodular Functions and Optimization*, North-Holland Elsevier, Amsterdam, 1991.

[10] R.G. Gallager, *Information Theory and Reliable Communications*, John Wiley, New York, 1968.

[11] R.G. Gallager, *Variations on a theme by Huffman*, IEEE Trans. Inform. Theory, IT-24 (1978), pp. 668–674.

[12] E.N. Gilbert, *Codes based on inaccurate source probabilities*, IEEE Trans. Inform. Theory, IT-17 (1971), pp. 304–314. $g(N)$ is analyzed on p. 309.

[13] C.R. Glassey and R.M. Karp, *On the optimality of Huffman trees*, SIAM J. Appl. Math., 31 (1976), pp. 368–378.

[14] M.J. Golin and G. Rote, *A dynamic programming algorithm for constructing optimal prefix-free codes for unequal letter costs,* in Proc. ICALP 95, Z. Fulop and F. Gecseg, eds., Springer-Verlag, New York, 1995, pp. 256–267.

[15] R.L. Graham, *Applications of the FKG inequality and its Relatives*, in Mathematical Programming: The State of the Art, B. Korte, A. Bachem, and M. Grötschel, eds., Springer-Verlag, New York, 1983, pp. 115–131.

[16] G.H. Hardy, J.E. Littlewood, and G. Polya, *Inequalities*, Cambridge University Press, Cambridge, UK, 1934.

[17] A.J. Hoffman. *On Simple Linear Programming Problems*, in Convexity, Proc. Seventh Symposium in Pure Mathematics, Vol. VII, V. Klee, ed., AMS, 1961, pp. 317–327.

[18] D.A. Huffman, *A method for the construction of minimum redundancy codes*, Proc. IRE, 40 (1951), pp. 1098–1101.

[19] F.K. Hwang, *Generalized Huffman trees*, SIAM J. Appl. Math., 37 (1979), pp. 124–127.

[20] C.M. Klein, *A submodular approach to discrete dynamic programming*, European J. Operational Research, 80 (1995), pp. 145–155.

[21] D.E. Knuth, *Optimum binary search trees*, Acta Inform., 1 (1971), pp. 14–25.

[22] D.E. Knuth, *Dynamic Huffman coding*, J. Algorithms, 6 (1985), pp. 163–180.

[23] E. Lawler, *Combinatorial Optimization: Networks & Matroids*, Holt-Rinehart-Winston, New York, 1976.
[24] E.L. Lawler, *Submodular Functions & Polymatroid Optimization*, in Combinatorial Optimization: Annotated Bibliographies, A.H.G. Rinnooy Kan, M. O'hEigeartaigh, and J.K. Lenstra, eds., John Wiley & Sons, New York, 1985, pp. 32–38.
[25] L. Lovász, *Submodular functions and convexity*, in Mathematical Programming: The State of the Art, B. Korte, A. Bachem, and M. Grötschel, eds., Springer-Verlag, New York, 1983, pp. 235–257.
[26] U. Manber, *Introduction to Algorithms*, Addison-Wesley, Reading, MA, 1989.
[27] A.W. Marshall and I. Olkin, *Inequalities: Theory of Majorization and Its Applications*, Academic Press, New York, 1979.
[28] H. Narayanan, *Submodular Functions and Electrical Networks*, North-Holland Elsevier, Amsterdam, 1997.
[29] A. Ostrowski, *Sur quelques applications des fonctions convexes et concaves au sens de I. Schur (offert en homage à P. Montel)*, J. Math. Pures Appl., 31 (1952), pp. 253–292.
[30] J.M. Pallo, *Enumerating, ranking and unranking binary trees*, Computer Journal, 29 (1986), pp. 171–175.
[31] J.M. Pallo, *Some properties of the rotation lattice of binary trees,* Computer Journal, 31 (1988), pp. 564–565.
[32] D.S. Parker, *Conditions for optimality of the Huffman algorithm*, SIAM J. Comput., 9 (1980), pp. 470–489.
[33] D.S. Parker and P. Ram, *Greed and Majorization*, Technical Report CSD-960003, UCLA Computer Science Dept., Los Angeles, 1996.
[34] D.S. Parker and P. Ram, *A Linear Algebraic Reconstruction of Majorization*, Technical Report CSD-970036, UCLA Computer Science Dept., Los Angeles, 1997.
[35] U. Pferschy, R. Rudolf, and G.J. Woeginger, *Monge matrices make maximization manageable*, Oper. Res. Lett., 16 (1994), pp. 245–254.
[36] G.-C. Rota, *On the foundations of combinatory theory* I. *Theory of Möbius functions,* Z. Wahrscheinlichkeitstheorie, 2 (1964), pp. 340–368.
[37] I. Schur, *Über eine Klasse von Mittelbildungen mit Anwendungen auf die Determinantentheorie*, Sitzungsber. Berl. Math. Ges., 22 (1923), pp. 9–20.
[38] C.E. Shannon, *The Lattice Theory of Information*, Proc. IRE Trans. Information Theory, 1 (1950). Reprinted in *Claude Elwood Shannon: Collected Papers*, IEEE Press, Piscataway, NJ, 1993.
[39] N.J.A. Sloane and S. Plouffe, *The Encyclopedia of Integer Sequences*, Academic Press, New York, 1995.
[40] J.S. Vitter, *Design and Analysis of Dynamic Huffman Codes*, J. ACM, 34 (1987), pp. 825–845.
[41] J.S. Vitter, *Algorithm* 673: *Dynamic Huffman coding*, ACM Trans. Math. Software, 15 (1989), pp. 158–167.
[42] F.F. Yao, *Efficient dynamic programming using quadrangle inequalities*, in Proc. 12th Annual ACM Symp. on Theory of Computing, Los Angeles, CA, 1980, pp. 429–435.

SIAM J. COMPUT.
Vol. 28, No. 5, pp. 1906–1922

TRACTABILITY OF PARAMETERIZED COMPLETION PROBLEMS ON CHORDAL, STRONGLY CHORDAL, AND PROPER INTERVAL GRAPHS*

HAIM KAPLAN†, RON SHAMIR‡, AND ROBERT E. TARJAN§

Abstract. We study the parameterized complexity of three NP-hard graph completion problems.

The *minimum fill-in* problem asks if a graph can be triangulated by adding at most k edges. We develop $O(c^k m)$ and $O(k^2 mn + f(k))$ algorithms for this problem on a graph with n vertices and m edges. Here $f(k)$ is exponential in k and the constants hidden by the big-O notation are small and do not depend on k. In particular, this implies that the problem is fixed-parameter tractable (FPT).

The *proper interval graph completion* problem, motivated by molecular biology, asks if a graph can be made proper interval by adding no more than k edges. We show that the problem is FPT by providing a simple search-tree-based algorithm that solves it in $O(c^k m)$-time. Similarly, we show that the parameterized version of the *strongly chordal graph completion* problem is FPT by giving an $O(c^k m \log n)$-time algorithm for it.

All of our algorithms can actually enumerate all possible k-completions within the same time bounds.

Key words. design and analysis of algorithms, parameterized complexity, chordal graphs, proper interval graphs, strongly chordal graphs, minimum fill-in, physical mapping of DNA

AMS subject classifications. 68Q20, 68R15, 05C85

PII. S0097539796303044

1. Introduction.

The focus of this paper is the parameterized complexity of several graph completion problems. Many well-known NP-hard problems can be stated with a parameter k so that they have polynomial-time algorithms when k is fixed. (For example, given a graph, decide if it has a vertex cover of size at most k, an independent set of size at least k, or pathwidth at most k.) The way the complexity depends on k varies dramatically, however. Some problems (e.g., *vertex cover* and *pathwidth*) can be solved in linear time when k is fixed, but for others (like *independent set*) the best known algorithms require $\Omega(n^k)$ steps. How the complexity depends on k can be crucial for applications in which small, fixed parameter values are important, as in the problems we study here.

Downey and Fellows initiated a systematic complexity analysis of such problems [1, 9, 10]. They called those parameterized problems that have algorithms of complexity $O(f(k)n^\alpha)$ (with α a constant) *fixed-parameter tractable* (FPT) and defined

*Received by the editors May 6, 1996; accepted for publication (in revised form) November 4, 1997; published electronically May 26, 1999. Portions of this paper were presented at the 34th Annual IEEE Symposium on the Foundations of Computer Science, Santa Fe, NM, 1994, and were published as *Tractability of parameterized completion problems on chordal and interval graphs: Minimum fill-in and physical mapping* in Proceedings of the 35th Symposium on Foundations of Computer Science, IEEE Computer Science Press, Los Alamitos, CA, 1994, pp. 780–791.

http://www.siam.org/journals/sicomp/28-5/30304.html

†AT&T Labs Research, 180 Park Ave, Florham Park, NJ 07932 (hkl@research.att.com).

‡Department of Computer Science, Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv 69978, Israel (shamir@math.tau.ac.il). The work of this author was supported in part by a grant from the Ministry of Science and the Arts, Israel.

§Department of Computer Science, Princeton University, Princeton, NJ 08544, and InterTrust Technologies Corporation, Sunnyvale, CA 94086 (ret@cs.princeton.edu). The research of this author at Princeton University was partially supported by NSF grant CCR-8920505 and Office of Naval Research contract N00014-91-J-1463.

a hierarchy of parameterized decision problem classes, $FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots$, with appropriate reducibility and completeness notions. They also conjectured that each of the containments in this hierarchy is proper. (See [1, 9, 10] for definitions and details.) Thus, for example, *vertex cover* and *pathwidth* are in FPT [3, 11, 24], but *independent set* is $W[1]$-complete [1] and *bandwidth* is W[t]-hard for all t [4].

Let Π be a family of graphs such that $K_n \in \Pi$ for every n. The Π-*completion* problem is defined as follows. Given a graph $G = (V, E)$, find a smallest set of edges A such that $G = (V, E \cup A) \in \Pi$. The parameterized version of the Π-completion problem, denoted by Π-*completion(k)*, asks whether there exists an edge set A such that $|A| \leq k$ and $G = (V, E \cup A) \in \Pi$.

In this paper we study the parameterized complexity of Π-*completion(k)* for three graph families Π; namely, chordal, proper interval, and strongly chordal graphs.

A graph is *chordal* (or *triangulated*) if every cycle of length four or more contains a chord (an edge between nonadjacent vertices on the cycle). The *chordal completion* problem is also known as the *minimum fill-in* problem and has received a lot of attention in the past due to its importance in sparse matrix computation (cf. [17]). Rose [33] has shown that for a sparse, symmetric matrix, finding an order of Gaussian elimination steps on diagonal elements that minimizes the number of nonzeros generated in the elimination process (assuming no lucky cancellation of nonzeros) is equivalent to solving the minimum fill-in problem on a corresponding graph.

Yannakakis [41] has shown that minimum fill-in is NP-complete. We focus here on *chordal completion(k)* or *fill-in(k)*, the parametrized version of the problem as defined above. Here k is fixed (to be thought of as a small constant) and is not part of the input. For a graph with n vertices and m edges, the problem can be solved by enumeration in $O(n^{2k}m)$-time, but we seek an algorithm with better dependence on k. In section 2 we describe two such algorithms. We first present a fairly simple, $O(c^k m)$-time search-tree-based algorithm, which already implies that the problem is in FPT. The same technique was previously used by Downey and Fellows [11] to prove parameterized tractability of *vertex cover*, *dominating set in planar graph, feedback vertex set,* and *face cover number of planar graph*. We then develop a more involved algorithm that gives a stronger complexity result: its multiplicative factor depending on k is *polynomial*, and the exponential in k appears only as an *additive* factor. Specifically, this algorithm has complexity $O(k^2nm + f(k))$, where $f(k)$ is exponential in k. For appropriate values of k (growing with the graph size), this algorithm will beat the simple algorithm. Here and throughout the paper we specify the dependence of the complexity on k explicitly, and the constants hidden by the big-O notation do not depend on k.

The second part of the paper deals with the parameterized complexity of the *proper interval completion* (PIC) problem. An *interval graph* is a graph for which one can assign an interval on the real line to each vertex so that two vertices are adjacent iff their intervals intersect. It is a *unit interval graph* if all intervals assigned have equal length. It is *proper interval* if it has an assignment in which no interval properly contains another. The last two notions are equivalent for finite graphs [30]. Interval completion problems arise in molecular biology and in the human genome project. In *physical mapping* of DNA, a set of long contiguous intervals of the DNA chain (called *clones*) is given, together with experimental information on their pairwise overlaps. The goal is to build a map describing the relative position of the clones [7, 27, 22, 4]. We concentrate here on the biologically important case in which all clones have equal length. In the presence of "false negative" errors (unidentified overlaps) the problem of

building a map with fewest errors is equivalent to PIC. This problem is NP-hard [18]. But what about its complexity for a small fixed number of errors? Let PIC(k) be the parameterized version of the problem, in which one asks for an augmenting set with no more than k edges if one exists. We prove parameterized tractability of PIC(k) by providing an $O(c^k m)$-time algorithm.

The third part of the paper considers the parameterized version of the *strongly chordal completion* problem (SCC(k)). The class of strongly chordal graphs was defined and characterized by Farber [12]. Denote by $N(v)$ the set of neighbors of a vertex v, including v itself. A *perfect elimination ordering* of a graph $G = (V, E)$ is an ordering $v_1, v_2, \ldots, v_n$ of V with the property that for each i, j, and l, if $i < j, i < l$, and $v_l, v_j \in N(v_i)$, then $v_l \in N(v_j)$. Rose [31] has shown that a graph is chordal iff it admits a perfect elimination ordering. A *strong elimination ordering* of a graph $G = (V, E)$ is an ordering $v_1, v_2, \ldots, v_n$ of V with the property that for each i, j, k, and l, if $i < j$, $k < l$, $v_k, v_l \in N(v_i)$, and $v_k \in N(v_j)$, then $v_l \in N(v_j)$. A graph is *strongly chordal* if it admits a strong elimination ordering. It is easy to see that every strong elimination ordering is also a perfect elimination ordering, and thus every strongly chordal graph is also a chordal graph. In addition every interval graph is strongly chordal. One can obtain a strong elimination order for an interval graph G by fixing a representation R of G and ordering the vertices in increasing right-endpoint order of their intervals in R. Interest in strongly chordal graphs arises in several ways. First, the problems of locating minimum weight dominating sets and minimum weight independent dominating sets in strongly chordal graphs with real vertex weights can be solved in polynomial time, whereas each of these problems is NP-hard for chordal graphs [13]. Second, these graphs have surprisingly nice structural properties and are intimately related to the class of totally balanced matrices [2]. We show that SCC(k) is FPT by describing an $O(c^k m \log n)$-time algorithm for it.

Section 2 contains the algorithms for chordal completion. Section 2.1 describes the simple search-tree–based algorithm, and section 2.2 gives the details of the more involved $O(k^2 nm + f(k))$-time algorithm. Section 3 extends the search tree algorithm of section 2.1 to solve PIC(k), and section 4 extends it to solve SCC(k). These extensions require additional ideas in order to handle the obstructions characterizing each particular graph family. Section 5 contains a summary and suggestions for some further research.

2. Minimum fill-in. In this section we present two algorithms for *fill-in(k)*. In section 2.1 we begin by describing an $O(c^k m)$-time algorithm for the problem. Then in section 2.2 we use additional new ideas to develop an $O(k^2 nm + f(k))$-time algorithm. Which of these algorithms is faster depends on the size of k compared to n and m. Both algorithms can actually enumerate all minimal k-triangulations of the input graph within the same time bounds.

We will use the following notation. Let $G = (V, E)$ be an undirected graph. For $X \subseteq V$, we denote by G_X the subgraph of G induced by the vertex set X. We define the *length* of a path (cycle) as the number of edges on the path (cycle). A *triangulation* of a graph $G = (V, E)$ is a set of edges F where $E \cap F = \emptyset$ and $\tilde{G} = (V, E \cup F)$ is a chordal graph. We will also say that the set of edges F *triangulates* G. If $|F| \leq k$, then F is a *k-triangulation*. We shall also refer to $\tilde{G}$ as a triangulation of G when there is no confusion. We assume without loss of generality that G is connected and $n \geq 2$; thus $n = O(m)$. A triangulation F is *minimal* if no proper subset of F triangulates G.

2.1. A linear algorithm for fixed k. A *triangulation of a chordless cycle* C is a set T of chords of C such that there is no induced chordless cycle in $C \cup T$. We

shall characterize and count the number of minimal triangulations of a cycle C. We call a cycle an *l-cycle* if it contains l vertices. A *triangle* is a 3-cycle. The proof of the following lemma is straightforward by induction.

LEMMA 2.1. *A minimal triangulation T of an n-cycle C consists of $n-3$ chords. It partitions C into $n-2$ triangles. Any two of these triangles are either disjoint or share a chord. Every chord in T is shared by exactly two triangles.*

The following lemma is well known (cf. [34] and the proof of Lemma 4.3, which is similar).

LEMMA 2.2. *There is a* 1-1 *correspondence between the minimal triangulations of a cycle with l vertices and the binary trees with $l-2$ internal nodes.*

Denote by c_l the lth Catalan number, i.e., $c_l = \binom{2l}{l}\frac{1}{l+1}$. Note that $c_l < 4^l$. Denote the number of binary trees with n internal nodes by b_n. The value b_n satisfies the recurrence $b_0 = 1$, $b_n = \sum_{i+j=n-1} b_i b_j$ for $n \geq 1$. The solution to this recurrence is $b_n = c_n$ (cf. [19]). Thus the following lemma is implied by Lemma 2.2.

LEMMA 2.3. *The number of minimal triangulations of an l-cycle is $c_{l-2} \leq 4^{l-2}$.*

The algorithm will traverse part of a search tree in which each node corresponds to a supergraph of G. This search tree is defined as follows. The graph G itself corresponds to the root of the tree. In order to generate the children of an internal node x that corresponds to a graph G', one needs to find a chordless cycle C in G'. Node x will have a child for each minimal triangulation of C. The graph corresponding to a child is obtained by adding the corresponding minimal triangulation to G'. If $|C| = l$, by Lemma 2.3 node x will have c_{l-2} children. Each leaf of the tree corresponds to a chordal supergraph of G. Note that every minimal triangulation of G is represented by at least one leaf.

One can find a chordless cycle C in a nonchordal graph with m edges in $O(m+n)$-time by the maximum cardinality search (MCS) algorithm described in [37, 38]. Using the algorithm described in [35] and the mapping described in Lemma 2.2, one can generate all minimal triangulations in $O(|C|)$-time per triangulation.

The algorithm actually visits only the nodes of the search tree that correspond to supergraphs of G with no more than k additional edges. If one such node is a leaf, then we have found a k-triangulation. Otherwise, no such triangulation exists.

THEOREM 2.4. *All minimal k-triangulations of a graph G can be found in $O(2^{4k}m)$-time.*

Proof. Let T be the subtree of the search tree traversed by the algorithm. For a node $x \in T$ let $G_x = (V, E_x)$ be the corresponding supergraph of G, d_x the maximum length of a path from x to a leaf of T, and $a_x = \max(\{|E_l| - |E_x| \mid l \text{ is a leaf descendant of } x\})$. Denote by l_x the total number of leaves among the descendants of x. By induction we prove that $l_x \leq 4^{d_x+a_x}$. Thus the total number of nodes in T is bounded by $2 \cdot 4^{2k}$. For each such node a linear amount of time is spent, consisting of the time to generate it and the time to find a chordless cycle in the graph corresponding to it.

Here is the induction argument. Assume the claim is true for all the children of a node x. Let l be the length of the cycle detected at x. Let $d_{\max} = \max\{d_y \mid y \text{ is a child of x}\}$, and let $a_{\max} = \max\{a_y \mid y \text{ is a child of x}\}$. Using the induction hypothesis, the number of leaf descendants of any of the c_{l-2} children of x is bounded by $4^{d_{\max}+a_{\max}}$. Thus the total number of leaf descendants of x is bounded by $4^{l-2}4^{d_{\max}+a_{\max}} = 4^{d_{\max}+1+a_{\max}+l-3} = 4^{d_x+a_x}$. The last equality follows from the fact that the size of a minimal triangulation of a chordless l-cycle is $l-3$, as stated in Lemma 2.1. ∎

The algorithm for enumerating minimal k-triangulations can actually list the same triangulation several times. We can eliminate this redundancy by storing solutions in a table and checking each new solution to see if it has been found already. If we use a k-dimensional search tree to store solutions, the extra time per search tree node to test for redundancy is $O(k \log k)$. Using universal hashing [39] or dynamic perfect hashing [16], the extra time per search tree node is $O(k)$, but the algorithm becomes randomized. These ideas apply equally well to the other enumeration algorithms proposed in this paper.

2.2. An algorithm with a polynomial multiplicative factor. To achieve an $O(k^2nm + f(k))$-time bound for minimal k-triangulation, we first describe an algorithm such that if G can be triangulated with no more than k edges, the algorithm partitions the vertex set of G into two subsets A, B such that the size of A is $O(k^3)$ and there are no chordless cycles in G that contain vertices in B. Then we prove that obtaining a k-triangulation of G is equivalent to obtaining a $(k-a)$-triangulation of A for some $a \geq 0$.

2.2.1. Partitioning the graph. The algorithm uses three main procedures, denoted P_1, P_2, P_3, executed in sequence. These procedures are described below.

(P_1) *Extracting independent chordless cycles.* This procedure starts with $B = V, A = \emptyset$ and repeatedly finds a chordless cycle in G_B using the MCS algorithm and moves its vertices to A. Note that when P_1 is finished, the induced subgraph on B is chordal.

Let $C_1, \ldots, C_j$ be the cycles extracted. The minimum number of chords needed to triangulate each C_i is $|C_i| - 3$. The algorithm maintains a dynamic lower bound cc on the minimum number of chords needed to triangulate G. After detecting the chordless cycle C_i, it increases cc by $|C_i| - 3$. Thus, if at some point $cc > k$, the algorithm can stop with a negative answer. Otherwise procedure P_1 ends when there are no more chordless cycles in B and $cc = \sum_{i=1}^{j}(|C_i| - 3) \leq k$.

The complexity of this part is $O(km)$. The MCS algorithm runs in linear time and the number of cycles detected is not greater than k since each cycle adds at least one to the dynamic lower bound cc. The size of the set A after performing this procedure is $O(k)$.

(P_2) *Extracting related chordless cycles with independent paths.* This procedure looks for chordless cycles that intersect both parts of the current partition, A and B, and contain at least two consecutive vertices in B, as long as such cycles exist. Let C be such a cycle, $|C| = l$. If $l > k + 3$, the algorithm stops with a negative answer. Otherwise every maximal subpath of C containing only vertices from B is moved into A if its length is at least one. The increase to cc depends on the structure of C. We need the following lemma in order to specify this increase precisely.

LEMMA 2.5. *Let C be a chordless cycle, and let p be a path in C of length l with $1 \leq l \leq |C| - 2$. If $l = |C| - 2$, then in every minimal triangulation of C there are at least $l - 1$ chords incident with at least one vertex of p. If $l < |C| - 2$, then in every minimal triangulation of C there are at least l chords incident with at least one vertex on p.*

Proof. If $l = |C| - 2$, then every chord in a minimal triangulation of C is incident with some vertex of p; thus the first part of the lemma holds. We prove the second part by induction on the path length. Obviously there must be a chord incident with at least one of the vertices on p; thus the lemma holds for paths of length one. Assume the result is true for every path with length less than l. Let p be a path with length l. Let (a, b) be a chord incident with p dividing the cycle C into two cycles C_1, C_2.

Case 1. $a, b \in p$. Let l_1 be the length of the subpath of p that connects a and b. Without loss of generality we can assume that $l_1 = |C_1| - 1$. Let p' be the path between the endpoints of p passing through (a, b) in C_2, and let $l_2 = |p'|$. There must be at least $l_1 - 2$ chords incident with p in C_1 and according to the induction hypothesis l_2 chords incident with p' in C_2. Thus the total number of chords incident with p will be at least $(l_1 - 2) + l_2 + 1 = l$.

Case 2. $a \in p$, $b \notin p$. Let $p_1 = p \cap C_1$, $p_2 = p \cap C_2$. For at least one $i = 1, 2$, $|p_i| < |C_i| - 2$. Without loss of generality assume that $|p_1| < |C_1| - 2$. By applying the induction hypothesis and using the previous part of the lemma, we find that the total number of chords incident with p is at least $l_1 + (l_2 - 1) + 1 = l$. □

Suppose that C is a chordless l-cycle that contains $j \geq 1$ disjoint maximal subpaths $p_1, \ldots, p_j$, each of length at least one, that are in B. Let $l_i = |p_i|$, $i = 1, \ldots, j$. Obviously if $l_1 = l - 2$, then $j = 1$, i.e., there is only one such subpath. Otherwise $l_i < l - 2$ for every $1 \leq i \leq j$. Using the previous lemma, we can increase our lower bound *cc* as follows. If there is only one such subpath, *cc* is increased by either $(l_1 - 1)$ if $l_1 = l - 2$ or l_1 if $l_1 < l - 2$. Otherwise *cc* is increased by the larger of $\frac{1}{2}\sum_{i=1}^{j} l_i$ (the factor $\frac{1}{2}$ is needed because a chord can be counted twice in the sum) and $\max\{l_i \mid 1 \leq i \leq j\}$. P_2 terminates whenever either *cc* is greater than k, in which case it stops with a negative answer, or when there are no more cycles of the appropriate kind.

In order to complete the description of P_2 we need to specify how to detect a chordless cycle C with consecutive vertices in B if such a cycle exists. The following observation is useful.

Observation 2.6. *There exists a chordless cycle C with at least two consecutive vertices in B iff there exists an edge (x, y), $x \in A$, $y \in B$ and a path between a vertex in $(N(y) - N(x)) \cap B$ and a vertex in $N(x) - N(y)$ that avoids any other vertices in $N(x) \cup N(y)$.*

One can detect whether such a path exists as follows: Delete $N(x) \cap N(y)$ and $(N(y) - N(x)) \cap A$ from G. Find the connected components of G induced on the other vertices. Check whether there is a vertex in $(N(y) - N(x)) \cap B$ and a vertex in $N(x) - N(y)$ in the same connected component. This process requires $O(m)$-time per edge (x, y) and can be implemented so that if the path exists, then the process will output a chordless cycle through (x, y) for which the other neighbor of y is also in B.

Recall that the size of A after the execution of P_1 is $O(k)$. The number of vertices added to A after the detection of each cycle by P_2 is at most twice the increase to *cc*. Since *cc* is never greater than k, the total number of vertices in A when P_2 ends remains $O(k)$.

(P_3) *Adding essential edges in G_A.* For every nonadjacent pair of vertices $y, z \in V$ define $A_{y,z}$ to be the set of all vertices x such that y, x, z appear consecutively on some chordless cycle in G.

Lemma 2.7. *If for some pair $y, z \in A$, $(y, z) \notin E$, $|A_{y,z}| > 2k$, then the edge (y, z) is in every k-triangulation of G.*

Proof. Assume that (y, z) is not in a k-triangulation $\overline{G} = (V, \overline{E})$ of G. Then there must be a chord in $\overline{E} - E$ incident with each vertex in $A_{y,z}$. Since no more than two such vertices can share a chord, $|\overline{E} - E| > k$, which is a contradiction. □

Edges (y, z) satisfying the lemma are called *essential*. For a triple y, x, z such that $(y, x) \in E$, $(x, z) \in E$, $(y, z) \notin E$ one can determine whether y, x, z appear consecutively on some chordless cycle in linear time: They appear consecutively on

a chordless cycle iff y and z are in the same connected component after deleting $N(x) - \{y, z\}$ from G.

P_3 first calculates the sets $A_{y,z}$ for every pair $y, z \in A$, $(y, z) \notin E$. Then for each pair $y, z \in A$ such that $|A_{y,z}| > 2k$, we add (y, z) to G'. Finally, we add to A all vertices in each computed set $A_{y,z}$ such that $|A_{y,z}| \leq 2k$.

We now analyze the overall complexity of the partitioning scheme.

LEMMA 2.8. (1) *The execution of P_2 takes $O(knm)$-time.* (2) *The execution of P_3 takes $O(k^2nm)$-time.*

Proof. (1) For each edge (x, y), $x \in A$, $y \in B$, it takes linear time to find a chordless cycle through (x, y) with consecutive vertices in B. The size of A is always $O(k)$; thus the total number of edges incident with vertices of A is always $O(kn)$. For each such edge we may have to run the test mentioned above once, giving a total time complexity $O(knm)$.

(2) Since the size of A when P_3 begins its execution is $O(k)$, the number of triples y, x, z such that $(y, x), (z, x) \in E$, $(y, z) \notin E$, $y, z \in A$, is $O(k^2n)$. For each triple we need to check whether there exists a path between y and z after deleting $N(x) - \{y, z\}$ from G. As mentioned above, this can be done by identifying connected components of G on the remaining vertices and then checking whether y and z are in the same connected component. □

Thus the overall complexity of the partitioning procedure is dominated by the complexity of P_3, which is $O(k^2nm)$. Before the call to P_3, the size of the set A is $O(k)$. Procedure P_3 may add $O(k)$ additional vertices to A for each pair of vertices in A prior to its execution so that we end up with $O(k^3)$ vertices in A.

Let E_s be the set of essential edges detected by P_3, and let $G' = (V, E \cup E_s)$. Denote by A_2, B_2 the partition of the vertex set before the execution of P_3 and by A, B the final partition.

The following lemma will be useful in establishing the correctness of the partitioning scheme and the completion algorithm.

LEMMA 2.9. *Let $G = (V, E)$ be a graph and $v \in V$. Let F be a set of edges between vertices of G such that*

(1) *each $e \in F$ is a chord in a chordless cycle C_e of G,*

(2) *$F \cap E = \emptyset$,*

(3) *v is not an endpoint of any $e \in F$.*

Denote by G^+ the graph obtained from G by adding the edges in F. If there exists a chordless cycle C in G^+ with v_1, v, v_2 occurring consecutively on C, then either there exists a chordless cycle in G on which v_1, v, v_2 occur consecutively or there exists a chordless cycle $D_e = v, x_1, \ldots, x_t, v$ in G such that the path $p = x_1, \ldots, x_t$ is part of a cycle C_e for some $e \in F$, and p contains one of the endpoints of e.

Proof. For an $e = (x, y) \in F$ let P_e^1 and P_e^2 denote the two paths on C_e between x and y with x and y removed from each path. Since C_e is chordless, for every e such that $v \in C_e$, v is not adjacent to any vertex either on P_e^1 or on P_e^2.

Case 1. For every $e \in F$ such that $v \notin C_e$, there exists a path $P_e \in \{P_e^1, P_e^2\}$ such that v is not adjacent in G to any vertex on P_e. Consider the cycle C. Replacing every edge $e \in F$ along C by P_e, one gets a cycle C' in G (not necessarily chordless or simple) with the property that v is not adjacent to any vertex in $C' - \{v_1, v_2\}$. Since edges in F are not incident with v, the edges (v, v_1) and (v, v_2) exist in G. Since C is chordless, $(v_1, v_2) \notin E$. Thus C' contains a chordless cycle in G on which v_1, v, v_2 occur consecutively.

Case 2. For some $e = (x, y) \in F$, v is adjacent to a vertex $u_1 \in P_e^1$ and a vertex

$u_2 \in P_e^2$. Since C is chordless in G^+, v must be nonadjacent either to x or to y in G. Without loss of generality assume v is not adjacent to x and that u_1 and u_2 are the closest to x among all vertices on P_e^1 and P_e^2, respectively, that are adjacent to v. D_e is the chordless cycle in G consisting of the path between u_1 and u_2 through x on C_e and v. □

The correctness of the partitioning scheme is captured by the following theorem.

THEOREM 2.10. *When the partitioning procedure ends, the graph G' has no chordless cycles with vertices in B.*

Proof. The proof is by contradiction. Suppose that there is a chordless cycle $C \in G'$ such that $C \cap B \neq \emptyset$, and let v be a vertex in $C \cap B$. Denote by v_1 and v_2 the two neighbors of v on C. Cycle C must contain at least one essential edge since otherwise C exists in G and either v would have been moved to A or (v_1, v_2) would have been added as an essential edge. Let F be the set of essential edges on C. By the definition of an essential edge, for each $e = (x, y) \in F$ there is a chordless cycle C_e in G in which e is a chord. Moreover, if P_e^1 and P_e^2 are the two paths connecting x and y on C_e, then either P_e^1 or P_e^2 consists of a single vertex $z_e \in B_2$. Since v is in B, it is not incident with any essential edge. Applying Lemma 2.9 we find that one of the following things must happen.

(1) There exists in G a chordless cycle on which v_1, v, v_2 occur consecutively. Thus either v should have been in A or (v_1, v_2) should have been added as an essential edge, and we obtain a contradiction.

(2) There exists a chordless cycle D_e in G on which v and z_e occur consecutively for some $e \in F$. Since both v and z_e are in B_2, they both should have been moved to A by P_2, and again we obtain a contradiction. □

2.2.2. Triangulating the graph. All that remains is to show that once we have partitioned the graph, it suffices to look for $(k-a)$-triangulations of the smaller graph with vertex set A, where a is the number of essential edges added during the partitioning algorithm. This is the content of Theorem 2.13 below. In order to prove the theorem we need some background and preliminary results.

We define the *elimination* of a vertex v from G as the operation that deletes v from G and adds an edge between every nonadjacent pair among v's neighbors. Let $\alpha = v_1, \ldots, v_n$ be an ordering of the vertices of a graph $G = (V, E)$. We denote by G_i, $0 \leq i \leq n$ the graph obtained from G after eliminating the first i vertices in α ($G_0 = G$). Let x and y be two vertices in $G = (V, E)$. An *x,y separator* of G is a set $S \subseteq V - \{x, y\}$ such that when S is deleted from G, x and y occur in different connected components.

The following characterization of minimal triangulations was proved by Ohtsuki, Cheung, and Fujisawa.

THEOREM 2.11 (see [28]). *A triangulation F of $G = (V, E)$ is minimal iff for each $(x, y) \in F$, there exists no x, y separator S of G such that S is a clique of the triangulated graph $\tilde{G} = (V, E \cup F)$.*

Using this theorem we prove the following lemma that is needed for the proof of Theorem 2.13.

LEMMA 2.12. *Let F be a minimal triangulation of a graph $G = (V, E)$. Any edge in F is a chord in a chordless cycle of G.*

Proof. Let $v_1, \ldots, v_n$ be a perfect elimination ordering of $\tilde{G} = (V, E \cup F)$. We use this ordering to eliminate vertices from G. Since F is minimal, for each edge $e = (u, w) \in F$ there exists an index k, $k \geq 1$, such that $e \in G_k$ but $e \notin G_{k-1}$.

We claim that u and w are connected in G_{k-1} by a path such that none of its

vertices is adjacent to v_k. Here is the proof of the claim. Assume that no such path exists. Then the set $N_{G_{k-1}}(v_k) - \{u, w\}$ separates u and w in G_{k-1}. But it follows from the definition of a perfect elimination ordering that this set is a clique in $\tilde{G}_{k-1}$. Since F is a minimal triangulation of G, we must also have that $\tilde{G}_{k-1}$ is a minimal triangulation of G_{k-1}. This contradicts Theorem 2.11 and the claim follows.

We obtain that e is a chord of a chordless cycle in G_{k-1}. This cycle consists of u, v_k, and w occurring consecutively and a shortest path between u and w that avoids the neighborhood of v_k in G_{k-1}. We finish the proof by showing that e is also a chord of a chordless cycle of G. This is done by arguing that if C is a chordless cycle in G_j for some j, $1 \leq j \leq n$, then there exists a chordless cycle C' in G_{j-1} that is either identical to C or contains one additional vertex. If all the edges in C are in G_{j-1}, then $C' = C$ is a chordless cycle in G_{j-1}. Otherwise there is an edge (x, y) in C that is not in G_{j-1}. For each such edge both its endpoints must be adjacent to v_j. Of the vertices on C only x and y can be adjacent to v_j since otherwise C is not chordless in G_j. Take C' to be C with the vertex v_j added between x and y. □

THEOREM 2.13. *Let A, B be a partition of the vertex set V of a graph $G = (V, E)$ such that the vertices of every chordless cycle in G are contained in A. A set of edges F is a minimal triangulation of G iff F is a minimal triangulation of G_A.*

Proof. Let F be a minimal triangulation of G_A. We need to prove that $\tilde{G} = (V, E \cup F)$ is chordal. Assume that $\tilde{G}$ is not chordal. Let C be a chordless cycle in $\tilde{G}$. Since $\tilde{G}$ induced on A is chordal, $C \cap B \neq \emptyset$. By assumption, G does not contain chordless cycles with vertices in B; hence C must not exist in G and thus it contains at least one edge from F and $|C \cap A| \geq 2$. Let v be a vertex in $C \cap B$. According to Lemma 2.12, each edge $e \in F$ is a chord in a chordless cycle C_e of G whose vertices are in A. Since F is a minimal triangulation of G_A, v is not an endpoint of any edge in F. Using Lemma 2.9 we conclude that there must be a chordless cycle with v on it in G, contradicting the assumptions of the theorem.

To prove the other direction, let F be a minimal triangulation of G. There exists $F' \subseteq F$ that is a minimal triangulation of G_A. According to the first part of the proof, F' also triangulates G. Since F is minimal, we conclude that $F' = F$. □

2.2.3. Overall running time. The final step of the algorithm is to look for $(k - a)$-triangulations in vertex set A, as justified by Theorem 2.13. One can find one or all such triangulations by the algorithm described in section 2.1. Since the size of A is $O(k^3)$, the running time for this step is $O(k^6 2^{4k})$. The total time for the three-step partitioning process is $O(k^2 nm)$, giving a time bound for the entire algorithm of $O(k^2 nm + k^6 2^{4k})$. This algorithm has a better bound than the simple algorithm only for $k = \Omega(\log n)$. It would be a nice result to obtain an algorithm with a running time of $O(km + f(k))$. We leave this as an open problem.

3. Proper interval completion. A proper interval supergraph $G = (V, E \cup F)$ of a graph $G = (V, E)$ with $|F| \leq k$ is called a *k-proper interval supergraph* of G.

The algorithm presented in section 2.1 can be easily modified to produce all possible k-proper interval supergraphs of a graph, using the following observations. Proper interval graphs are exactly the chordal graphs that do not contain any of the three obstructions in Figure 3.1 as an induced subgraph [40]. Deng, Hell, and Huang [8] have recently described an algorithm that checks whether a graph G is a proper interval graph. In case G is indeed a proper interval graph the algorithm can provide a proper interval representation for G. The running time of the algorithm is $O(m)$, and it does not use complicated data structures such as PQ-trees [5]. It is

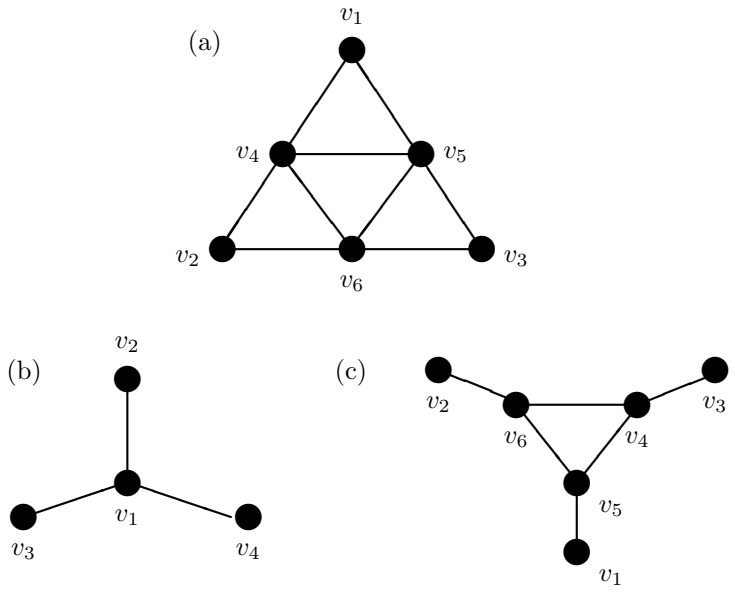


FIG. 3.1. *Obstructions for chordal graphs that are not proper interval.* (a) *Tent.* (b) *Claw.* (c) *Net.*

straightforward to check that in case the input graph is not a proper interval graph one can use the information maintained by the algorithm to extract either a chordless cycle or one of the obstructions in Figure 3.1 in linear time.

The k-completion algorithm will traverse part of a search tree defined as follows. The graph G itself corresponds to the root of the tree. Let x be a node of the search tree corresponding to a supergraph G_x of G that is not a proper interval graph. The children of x are obtained as follows. The algorithm by Deng, Hell, and Huang is applied to G_x to find either a chordless cycle or one of the obstructions in Figure 3.1. If a chordless cycle C is found in G_x, then every minimal triangulation of C gives rise to a child of x as in section 2.1. In case an obstruction is found, x has a child for every edge e between vertices of the obstruction that is not part of the obstruction. The supergraph corresponding to such a child is $G_x \cup \{e\}$. Thus if the obstruction found is a tent the node has six children, if it is a claw it has three, and if it is a net it has nine.

Each leaf in the search tree thus defined corresponds to a proper interval supergraph of G. Note that every minimal proper interval supergraph of G is represented by at least one leaf. As in section 2.1, the nodes of the search tree that are actually traversed correspond to supergraphs with no more than k additional edges. If one such node is a leaf, then we have found a k-proper interval supergraph. Otherwise, no such supergraph exists.

We summarize the result presented in this section in the following theorem. Its proof is analogous to the proof of Theorem 2.4 and is hence omitted.

THEOREM 3.1. *All k-proper interval supergraphs of a graph can be found in $O(2^{4k}m)$-time.*

Remark. Rose, Tarjan, and Lueker proved that if $G = (V, E)$ is triangulated and $G = (V, E \cup F)$ with $F \neq \emptyset$, $F \cap E = \emptyset$ is triangulated, then there exists an edge $e \in F$

such that $G = (V, E \cup \{e\})$ is also triangulated [32, Lemma 2]. Using this lemma, while traversing the search tree as described above one can avoid generating non-triangulated children of nodes that correspond to triangulations of G. Each minimal proper interval completion of G is still guaranteed to be represented by at least one leaf. In this version of the algorithm one uses the MCS algorithm to detect chordality and find a chordless cycle as long as a chordal supergraph has not been reached. When reaching a chordal supergraph, the algorithm by Deng, Hell, and Huang is applied to get one of the obstructions in Figure 3.1. The children of the node are then generated as described above. Finally, the MCS algorithm is applied to each of the children in order to avoid traversing those that are not chordal. Those that are chordal are further expanded. Such an implementation would use the algorithm of Deng, Hell, and Huang only on chordal graphs and hence a somewhat simpler version of it would suffice.

4. Strongly chordal completion. A chord (v, w) in an even cycle C is *odd* if the paths connecting v and w on C contain an odd number of edges.

The following characterization of strongly chordal graphs is due to Farber [12].

THEOREM 4.1. *A graph G is strongly chordal iff G is chordal and every even cycle of length at least six in G has an odd chord.*

An odd chord in an even cycle C partitions C into two smaller even cycles C_1 and C_2. Any odd chord in C_1 or C_2 is an odd chord in C as well. A 4-*cycle decomposition* of an even chordless cycle C is a minimal set T of odd chords in C such that there is no induced even chordless cycle of length at least six in $C + T$.

Next we characterize and count the number of minimal 4-cycle decompositions of an even cycle C. Let $|C| = n$.

The proof of the following lemma is straightforward by induction.

LEMMA 4.2. *A minimal 4-cycle decomposition T of an even n-cycle C consists of $(\frac{n}{2} - 2)$ chords. It partitions C into $(\frac{n}{2} - 1)$ 4-cycles. Every two of these 4-cycles are either disjoint or share a chord. Every chord is shared by exactly two 4-cycles.*

A *ternary tree* is a tree in which each internal node has three children. The following theorem establishes a correspondence between the set of 4-cycle decompositions of an even n-cycle and the set of ternary trees with $n - 1$ leaves and $\frac{n}{2} - 1$ internal nodes. This correspondence is similar to the one stated in Lemma 2.2 between minimal triangulations of a chordless n-cycle and binary trees with $n - 1$ leaves.

LEMMA 4.3. *The number of 4-cycle decompositions of an even n-cycle C is equal to the number of ternary trees with $\frac{n}{2} - 1$ internal nodes.*

Proof. For every even n-cycle C, construct an invertible mapping from the set of 4-cycle decompositions of C to the set of ternary trees with $\frac{n}{2} - 1$ internal nodes, as follows. The construction is by induction on the length of the cycle. Assume that one has constructed an invertible mapping for every cycle C', where $|C'| \leq n - 2$. Let C be an n-cycle, and let e be a fixed edge on C. Let T be a 4-cycle decomposition of C, and let $C_e = \{e, e_1, e_2, e_3\}$ be the 4-cycle in $C + T$ which includes e. If e_i, $i \in \{1, 2, 3\}$ is a chord, let C_i be the cycle $C - C_e + \{e_i\}$. The 4-cycle decomposition T induces a 4-cycle decomposition T_i of C_i. The tree which corresponds to T has a root (associated with the edge e); the ith child of the root is a leaf if $e_i \in C$ or the root of the ternary tree which corresponds to T_i under the mapping associated with C_i if e_i is a chord. It is straightforward to verify that the mapping defined above is indeed invertible. □

Denote the number of ternary trees with n internal nodes by t_n. The value t_n satisfies the following recurrence: $t_0 = 1$ and $t_n = \Sigma_{\{i+j+k=n-1\}} t_i t_j t_k$ if $n \geq$

1. According to Graham, Knuth, and Patashnik [19, p. 349], the solution to this recurrence is

$$t_n = \binom{3n+1}{n} \frac{1}{3n+1},$$

which is no greater than $2^{3n} = 8^n$. Together with Lemma 4.3 we obtain the following.

LEMMA 4.4. *The number of* 4*-cycle decompositions of an even* n*-cycle* C *is no greater than* $8^{\frac{n}{2}-1}$.

4.1. Finding an even cycle without odd chords. The *neighborhood matrix* of a graph is a symmetric 0-1 matrix with rows and columns indexed by the set of vertices of the graph and with an entry of 1 iff the corresponding two vertices are equal or adjacent in the graph. A *doubly lexical ordering* of a matrix is an ordering of the rows and of the columns so that the rows, as vectors, are lexically increasing and the columns, as vectors, are lexically increasing. *Lexical ordering* of vectors is the standard dictionary ordering, except that vectors will be read from highest to lowest coordinate. Thus row vectors will be compared from right to left and column vectors from bottom to top. A matrix M is *symmetric* if its rows and columns are indexed by the same set S and $M(s,t) = M(t,s)$ for all $s,t \in S$. A *symmetric ordering* of such an M is an ordering of S. It is not true that every symmetric matrix has a symmetric doubly lexical ordering. But it was proved by Lubiw [26] that a symmetric matrix that has a *dominant diagonal*, meaning that $M(s,s) \geq M(s,t)$ for all $s,t \in S$, has a symmetric doubly lexical ordering. In particular, the neighborhood matrix of any graph has a symmetric doubly lexical ordering.

A *cycle* matrix is a 0-1 $n \times n$ matrix, $n \geq 3$, with exactly two 1's in each row and in each column and such that no proper submatrix has this property. A *totally balanced matrix* is a 0-1 matrix with no cycle submatrices.

Farber [12] proved the following characterization of strongly chordal graphs.

THEOREM 4.5. *A graph is strongly chordal iff its neighborhood matrix is totally balanced.*

A Γ is an ordered 0-1-valued 2×2 matrix with exactly one 0, in the bottom right corner:

$$\Gamma = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

Lubiw proved the following property [26, Theorem 5.2] of a doubly lexical 0-1 matrix M with rows R and columns C.

THEOREM 4.6. *Let* M *be an ordered doubly lexical* 0-1 *matrix with rows* R *and columns* C. *Any* 2×2 *submatrix of* M *formed by* $r_1 < r_2 \in R$ *and* $c_1 < c_2 \in C$ *with* $M(r_1, c_2) = M(r_2, c_1) = 1$, $M(r_2, c_2) = 0$ *is, for some* $k \geq 3$, *embedded in a* $k \times k$ *submatrix of* M *formed by* $r_1 < r_2 < \cdots < r_k \in R$ *and* $c_1 < c_2 < \cdots < c_k \in C$ *with* $M(r_i, c_{i+1}) = M(r_{i+1}, c_i) = 1$ *for* $i = 1, \ldots, k-1$, $M(r_k, c_k) = 1$, *and* $M(r_i, c_j) = 0$ *for other* i, j *except possibly* $i = j = 1$. *In particular, any* Γ *submatrix is embedded in a cycle submatrix. See Figure* 4.1.

Together with the observation that in any ordering of a cycle submatrix there is a Γ submatrix, Theorem 4.6 reestablishes the following result.

THEOREM 4.7 (see [20, 2]). *A* 0-1 *matrix has a* Γ*-free ordering iff it is totally balanced. Moreover, a doubly lexical ordering of a totally balanced matrix is* Γ*-free.*

The following theorem makes a link between cycle submatrices in a neighborhood matrix of a graph G and chordless cycles or even cycles without odd chords in G.

	c_1	c_2	c_3	c_4			c_i			c_k
r_1	?	1	0	0						
r_2	1	0	1	0						
r_3	0	1	0	1			0			
r_4	0	0	1	0	.					
				.	.	.				
					.	0	1			
r_i						1	0	.		
			0				.	.	.	
								.	0	1
r_k									1	1

FIG. 4.1. *Every* Γ *submatrix can be embedded in a cycle submatrix.*

THEOREM 4.8. *Let* M *be a neighborhood matrix of a graph* G *and let* N *be a* $k \times k$ *cycle submatrix of* M *with rows* $r_1 < r_2 < \cdots < r_k$ *and columns* $c_1 < c_2 \cdots < c_k$. *Let* $V_N = \{v_l \mid l = r_i \text{ or } l = c_j, 1 \le i, j \le k\}$. *Then either the vertices of* V_N *form an even cycle without odd chords or there exists a subset* $C \subseteq V_N$ *that induces a chordless cycle.*

Proof. If $r_i \neq c_j$ for every $1 \le i, j \le k$, V_N clearly forms an even cycle without odd chords. Assume $r_i = c_j$ for some i and j. This implies that $N(i, j) = M(r_i, c_j) = 1$. Let i' be the other row in which column j has a 1 and let j' be the other column in which row i has a 1. $N(i', j') = 0$ since otherwise we get a contradiction to the fact that N is a cycle submatrix. Thus $r_{i'} \neq c_{j'}$. Among the vertices in V_N, v_{r_i} is adjacent only to $v_{r_{i'}}$ and $v_{c_{j'}}$. These two are not adjacent, but there is a path connecting them in $V - \{v_{r_i}\}$. Thus there exists a chordless cycle $C \subseteq V_N$ through v_{r_i}. □

Let M be a symmetric $n \times n$ neighborhood matrix of a connected graph G with m edges and n vertices. Using Paige and Tarjan's implementation [29] of the algorithm described by Lubiw [26], one can obtain a doubly lexical ordering of M in $O(m \log n)$-time. Lubiw [26] also shows how to search for a Γ submatrix in a doubly lexically ordered M in $O(m)$-time. Given a Γ submatrix in a doubly lexically ordered M, a cycle submatrix that contains it can also be found in $O(m)$-time [26]. According to Theorem 4.8, either the rows and columns of this cycle submatrix induce an even cycle without odd chords or a subset of them induce a chordless cycle in G. As suggested by the proof of Theorem 4.8, this cycle can be extracted from the cycle submatrix in $O(m)$-time.

4.2. The *k*-completion algorithm. As in sections 2.1 and 3, the k-completion algorithm will traverse part of a search tree in which each node corresponds to a supergraph of G. This search tree is defined as follows. The graph G itself corresponds to the root of the tree. In order to generate the children of an internal node x that corresponds to a graph G', one needs to find either a chordless cycle or an even cycle without odd chords in G'. In case a chordless cycle C is found, node x will have a child for each minimal triangulation T of C. If an even cycle without odd chords, C, is found, x will have a child for each 4-cycle decomposition of C. The graph corresponding to a child is obtained by adding the corresponding minimal triangulation or 4-cycle decomposition to G'. If C is a chordless l-cycle, by Lemma 2.3 node x will have at most c_{l-2} children. If C is an even l-cycle without odd chords,

then x will have $t_{\frac{l}{2}-1}$ children. Each leaf of the tree corresponds to a strongly chordal supergraph of G. Note that every such supergraph of G that is minimal is represented by at least one leaf.

Remark. In the case that a chordless cycle C is found in the graph corresponding to a node x, it will be more efficient to generate a child only for each triangulation T of C such that $C + T$ has no even cycles without odd chords.

One can find a chordless cycle C in a nonchordal graph with m edges and n vertices in $O(m)$-time by using the MCS algorithm described in [37, 38]. An even cycle without odd chords can be found in a chordal graph that is not strongly chordal in $O(m \log n)$-time using Paige and Tarjan's implementation [29] of Lubiw's algorithm [26], as described in section 4.1. Obviously, one can use Paige and Tarjan's algorithm for both tasks in order to simplify the implementation while getting some penalty in the performance. The algorithm described in [35] can be easily extended to enumerate all ternary trees with n internal nodes, spending $O(n)$-time for each. Applying Lemma 4.3, one obtains an algorithm that enumerates all 4-cycle decompositions of an even cycle C in $O(|C|)$-time for each. It is straightforward to check that a more involved enumeration procedure that enumerates all minimal strongly chordal triangulations of a chordless even cycle C in $O(|C|)$-time for each could be designed as well, based on the ideas in [35].

The nodes of this search tree that are actually traversed correspond to supergraphs of G with no more than k additional edges. If one such node is a leaf, then we have found a strongly chordal supergraph with no more than k additional edges. Otherwise, no such supergraph exists. The proof of the following theorem is analogous to the proof of Theorem 2.4.

THEOREM 4.9. *All minimal strongly chordal supergraphs of a graph G with no more than k additional edges can be found in $O(8^{2k} m \log n)$-time.*

Remark. An alternative implementation that avoids traversing nonchordal children of chordal supergraphs can be designed as described in the remark at the end of section 3.

Remark. For dense matrices, Spinrad describes a faster algorithm which can obtain a doubly lexical ordering in $O(n^2)$-time [36]. Hence the complexity of the algorithm described above can be improved for dense graphs to $O(8^{2k} \min(n^2, m \log n))$-time.

5. Concluding remarks. We have presented polynomial algorithms for the fixed-parameter version of three graph completion problems: *chordal completion(k)*, *strongly chordal completion(k)*, and *proper interval completion(k)*. Note that the class of proper interval graphs is a subset of the strongly chordal graphs, which are a subset of the chordal graphs. Our results immediately imply that *chordal completion*, *strongly chordal completion*, and *proper interval completion* have a polynomial-time algorithm when k is part of the input but restricted to be at most logarithmic in the size of the graph.

Another important graph family that we have not discussed in this paper is interval graphs. The *interval completion(k)* problem has an important application in molecular biology, as discussed in section 1. Its NP-completeness was proved in [23]. NP-completeness is also implied by the proof of Yannakakis [41] for *chordal graph completion*, as the graphs generated in that proof are chordal iff they are interval. To date the complexity status of the parametric version of the problem is open. It is not known whether the problem is in FPT or hard for some level of the W-hierarchy. The obstructions that have to exist in a chordal graph that is not interval are described

in [25]. An arbitrarily large obstruction X could exist in a graph that is not interval but could be made interval with the addition of any one out of $O(|X|)$ edges. This causes difficulties when one tries to apply the techniques of this paper to this graph class.

When the input is restricted to bounded-degree interval graphs for some fixed bound d, the obstruction size is bounded by $O(d)$ and the search tree technique applies to get a quadratic FPT result using the characterization of [25]. It is an open problem whether this obvious bound can be improved.

For the molecular biology application in physical mapping, one can assume that the ratio of sizes of the largest and the smallest clones is at most a small constant c (in practice, $c = 10$ suffices). Fishburn and Graham [15] (see also [14, Chapter 8]) provided characterizations for interval graphs which have such length restrictions. Their results, together with the characterizations of [25], imply that the obstruction size is $O(c)$ and thus for this case, too, the search tree technique applies and the k-completion problem is FPT.

After a preliminary version of this paper appeared in [21], Cai published a paper [6] that rediscovers our simple search-tree-based algorithm for *chordal completion(k)* (see section 2.1). Using a better-known bound on the lth Catalan number, namely, $c_l = O(4^l/l^{3/2})$, and a lemma showing that $c_{i+1}c_{j+1} \leq c_{i+j+1}$, Cai proves that our algorithm in fact runs in $O((4^k/(k+1)^{3/2})(m+n))$-time. Cai also proves a straightforward generalization of our Theorem 3.1. This generalization says that the parameterized version of the graph modification problem with respect to any graph property that can be characterized by a finite set of forbidden induced subgraphs is FPT. The proof of this generalization is similar to the proof of Theorem 3.1.

REFERENCES

[1] K. Abrahamson, R. Downey, and M. Fellows, *Fixed-parameter intractability* II, in Proc. 10th Symposium on Theoretical Aspects of Computer Science (STACS '93), Lecture Notes in Comput. Sci. 665, Springer–Verlag, Berlin, 1993, pp. 374–385.

[2] R. P. Anstee and M. Farber, *Characterizations of totally balanced matrices*, J. Algorithms, 5 (1984), pp. 215–230.

[3] H. L. Bodlaender, *A linear time algorithm for finding tree-decompositions of small treewidth*, in Proc. 25th ACM Symposium on the Theory of Computing, ACM Press, New York, 1993, pp. 226–234.

[4] H. L. Bodlaender, M. R. Fellows, and M. T. Hallet, *Beyond NP-Completeness for problems of bounded width: Hardness for the W hierarchy (extended abstract)*, in Proc. 26th ACM Symposium on the Theory of Computing, ACM Press, New York, 1994, pp. 449–458.

[5] K. S. Booth and G. S. Lueker, *Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 335–379.

[6] L. Cai, *Fixed-parameter tractability of graph modification problems for hereditary properties*, Inform. Process. Lett., 58 (1996), pp. 171–176.

[7] A. V. Carrano, *Establishing the order of human chromosome-specific DNA fragments*, in Biotechnology and the Human Genome, A. D. Woodhead and B. J. Barnhart, eds., Plenum Press, New York, 1988, pp. 37–50.

[8] X. Deng, P. Hell, and J. Huang, *Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs*, SIAM J. Comput., 25 (1996), pp. 390–403.

[9] R. G. Downey and M. R. Fellows, *Fixed-parameter intractability*, in Proc. 7th Structure in Complexity Theory Conference (Structures '92), Boston, MA, 1992, IEEE Computer Society Press, Los Alamitos, CA, pp. 36–49.

[10] R. G. Downey and M. R. Fellows, *Fixed-parameter tractability and completeness* III: *Some structural aspects of the W hierarchy*, in Complexity Theory: Current Research (Proc. 1992 Dagstuhl Workshop on Structural Complexity), Cambridge University Press, Cambridge, UK, 1993, pp. 191–226.

[11] R. G. Downey and M. R. Fellows, *Parameterized computational feasibility*, in Complexity Theory: Current Research, K. Ambos-Spies, S. Homer, and U. Schoning, eds., Cambridge University Press, New York, 1993, pp. 166–191.
[12] M. Farber, *Characterizations of strongly chordal graphs*, Discrete Math., 43 (1983), pp. 173–189.
[13] M. Farber, *Domination, independent domination, and duality in strongly chordal graphs*, Discrete Appl. Math., 7 (1984), pp. 115–130.
[14] P. Fishburn, *Interval Orders and Interval Graphs*, John Wiley, New York, 1985.
[15] P. Fishburn and R. L. Graham, *Classes of interval graphs under expanding length restrictions*, J. Graph Theory, 9 (1985), pp. 459–472.
[16] M. L. Fredman, J. Komlós, and E. Szemerédi, *Storing a sparse table with $o(1)$ worst case access time*, J. ACM, 31 (1984), pp. 538–544.
[17] A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice–Hall, Englewood Cliffs, NJ, 1981.
[18] M. C. Golumbic, H. Kaplan, and R. Shamir, *On the complexity of DNA physical mapping*, Adv. Appl. Math., 15 (1994), pp. 251–261.
[19] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Addison–Wesley, Reading, MA, 1989.
[20] A. J. Hoffman, A. W. J. Kolen, and M. Sakarovitch, *Totally balanced and greedy matrices*, SIAM J. Alg. Discrete Methods, 6 (1985), pp. 721–730.
[21] H. Kaplan, R. Shamir, and R. E. Tarjan, *Tractability of parameterized completion problems on chordal and interval graphs: Minimum fill-in and physical mapping*, in Proc. 35th Symposium on Foundations of Computer Science, IEEE Computer Science Press, Los Alamitos, CA, 1994, pp. 780–791.
[22] R. M. Karp, *Mapping the genome: Some combinatorial problems arising in molecular biology*, in Proc. 25th Annual ACM Symposium on the Theory of Computing, ACM Press, New York, 1993, pp. 278–285.
[23] T. Kashiwabara and T. Fujisawa, *An NP-complete problem on interval graphs*, in IEEE International Symposium on Circuits and Systems (12th), Institute of Electrical and Electronics Engineers, Piscataway, NJ, 1979, pp. 82–83.
[24] T. Kloks, *Treewidth*, Ph.D. thesis, Dept. of Computer Science, Utrecht University, Utrecht, The Netherlands, 1993.
[25] C. G. Lekkerkerker and J. C. Boland, *Representation of a finite graph by a set of intervals on the real line*, Fund. Math., 51 (1962), pp. 45–64.
[26] A. Lubiw, *Doubly lexical orderings of matrices*, SIAM J. Comput., 16 (1987), pp. 854–879.
[27] R. Nagaraja, *Current approaches to long-range physical mapping of the human genome*, in Techniques for the Analysis of Complex Genomes, R. Anand, ed., Academic Press, London, 1992, pp. 1–18.
[28] T. Ohtsuki, L. K. Cheung, and T. Fujisawa, *Minimal triangulation of a graph and optimal pivoting order in a sparse matrix*, J. Math. Anal. Appl., 54 (1976), pp. 622–633.
[29] R. Paige and R. E. Tarjan, *Three partition refinement algorithms*, SIAM J. Comput., 16 (1987), pp. 973–989.
[30] F. S. Roberts, *Indifference graphs*, in Proof Techniques in Graph Theory, F. Harary, ed., Academic Press, New York, 1969, pp. 139–146.
[31] D. J. Rose, *Triangulated graphs and the elimination process*, J. Math. Anal. Appl., 32 (1970), pp. 597–609.
[32] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.
[33] J. D. Rose, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computing, R. C. Reed, ed., Academic Press, New York, 1972, pp. 183–217.
[34] D. D. Sleator, R. E. Tarjan, and W. P. Thurston, *Rotation distance, triangulations, and hyperbolic geometry*, J. AMS, 1 (1988), pp. 647–681.
[35] M. Solomon and R. A. Finkel, *A note on enumerating binary trees*, J. ACM, 27 (1980), pp. 3–5.
[36] J. Spinrad, *Doubly lexical ordering of dense* 0-1 *matrices*, Inform. Process. Lett., 45 (1993), pp. 229–235.
[37] R. E. Tarjan and M. Yannakakis, *Simple linear-time algorithms to test chordality of graphs, text acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, SIAM J. Comput., 13 (1984), pp. 566–579.
[38] R. E. Tarjan and M. Yannakakis, *Addendum: Simple linear-time algorithms to test chordality of graphs, text acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*,

SIAM J. Comput., 14 (1985), pp. 254–255.

[39] M. N. Wegman and J. L. Carter, *New classes and applications of hash functions*, in Proc. 20th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1979, pp. 175–182.

[40] G. Wegner, *Eigenschaften der Nerven Homologische Eihfacher Familien in R^n*, Ph.D. thesis, Götingen University, Götingen, Germany, 1967.

[41] M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Alg. Discrete Methods, 2 (1981), pp. 77–79.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 1923–1940

TREE DATA STRUCTURES FOR N-BODY SIMULATION*

RICHARD J. ANDERSON†

Abstract. In this paper, we study data structures for use in N-body simulation. We concentrate on the spatial decomposition tree used in particle-cluster force evaluation algorithms such as the Barnes–Hut algorithm. We prove that a k-d tree is asymptotically inferior to a spatially balanced tree. We show that the worst case complexity of the force evaluation algorithm using a k-d tree is $\Theta(n \log^3 n \log L)$ compared with $\Theta(n \log L)$ for an oct-tree. (L is the separation ratio of the set of points.)

We also investigate improving the constant factor of the algorithm and present several methods which improve over the standard oct-tree decomposition. Finally, we consider whether or not the bounding box of a point set should be "tight" and show that it is safe to use tight bounding boxes only for binary decompositions. The results are all directly applicable to practical implementations of N-body algorithms.

Key words. N-body simulation, Barnes–Hut algorithm, spatial data structures

AMS subject classifications. 85-08, 68P05

PII. S0097539797326307

1. Introduction.

The gravitational force computation problem is as follows: Given a set of n particles, compute the net gravitational force exerted on each particle by the other particles. The solution of this problem is required in the inner loop of N-body simulation. The problem can be solved by a direct method, which is to compute all pairwise interactions of particles. This yields an $O(n^2)$ algorithm. However, much faster algorithms have been developed that use hierarchical spatial data structures to cluster particles. In this paper we study different tree data structures with the goal of minimizing the number of force evaluations performed during the computation.

We concentrate on particle-cluster algorithms, where the data structure is traversed independently for each particle. The Barnes–Hut algorithm [3] is the standard particle-cluster algorithm now in use. We study issues which relate both to asymptotic performance and to the constant factors of the run time. Our performance measure is the number of force evaluations performed with respect to a fixed error threshold. We give a collection of results which show how different tree properties influence the performance of algorithms. In terms of impact, our most significant results are in section 6 and show that k-d trees are asymptotically less efficient than oct-trees. In section 7 we explore how to improve the constant factors of the algorithm by improving the data structure, and in section 8 we give some results for different definitions of the boundary of a cell.

1.1. Astrophysical simulation.

N-body simulation is a very important tool in astrophysical research. It is essentially the only way to perform experiments to investigate basic cosmological problems. Massive simulations are used to study problems such as the formation of large-scale structure in the universe. Simulations may use as many as 50 million particles and may run for tens of thousands of CPU hours. We consider the problem, How does the choice of a spatial data structure influence the

*Received by the editors August 25, 1997; accepted for publication (in revised form) December 31, 1997; published electronically June 3, 1999.
http://www.siam.org/journals/sicomp/28-6/32630.html
†Department of Computer Science and Engineering, University of Washington, Box 352350, Seattle, WA 98195-2350 (anderson@cs.washington.edu).

performance of the force evaluation algorithm? The work was specifically motivated by discussions with astrophysicists on whether it is better to base a code on k-d trees or on oct-trees.

Our efforts are directed toward the *astrophysical* N-body problem. Although the N-body problem arises in many scientific disciplines, there are important differences between the versions of the problem that have a strong influence on which algorithm is the most practical. An important aspect of the problem in astrophysics is that the data often have a very nonuniform distribution with regions of high concentration and regions of low concentration. This means that algorithms which are best for uniform distributions are not necessarily suggestive of the real cases of interest.

There is a debate on whether "$O(n)$" methods or "$O(n \log n)$" methods are best for N-body simulation. To date, all massive astrophysical simulations have been based on the "$O(n \log n)$" methods, so we concentrate on the methods in actual use, even though they are "asymptotically inferior."

1.2. Overview. The next two sections give background on the force computation algorithm and on the tree data structures. This is followed by a performance analysis of spatial and density decompositions. The results on density decompositions were surprising in that they established bounds which were substantially worse than anticipated and they debunked the use of k-d trees in N-body codes. We then consider the degree of decomposition and argue that binary decomposition is likely to be substantially better than higher-degree decomposition. Finally, we address the question of whether the boundary of a cell should be tightened to the smallest enclosing box. Our result is that the tighter bounds can be used for a binary decomposition, but for higher-degree cases tighter bounds can lead to substantially worse performance.

1.3. Related work. There has been very little theoretical research done on comparing data structures for N-body simulation. More attention has been paid to the asymptotic performance of the algorithms and to the accuracy properties of particular codes.

A series of papers introduced particle-cluster algorithms in the early- to mid-eighties [1, 12, 3], with the version developed by Barnes and Hut [3] receiving the most attention with respect to implementation.

The fastest bounds for the force computation problem are for the fast multipole method of Greengard and Rokhlin [11, 10, 7, 15], which is a cluster-cluster algorithm. Although the algorithm is $O(n)$ time, it is not used in astrophysical research. This is primarily because the constant factors in the algorithm are very large, although memory is also a problem. Greengard's algorithm is more suited for uniform distributions than for the concentrated distributions encountered in astronomy. The issue of asymptotic run time of the force computation is complicated by the fact that tree construction in general is $\Omega(n \log n)$ which dominates in the case of simulation. However, many of the ideas in Greengard's algorithm are applicable to the particle-cluster algorithms and practical N-body codes are becoming more Greengard-like by using higher-degree series expansions. There are also algorithms that may be viewed as a synthesis of Greengard's algorithm and a particle cluster algorithm [4, 5]. Many of the issues relating to the force evaluation algorithm are orthogonal to the data structure used, so some of the results of this paper may carry over to a larger context.

Papers that have addressed the accuracy of N-body simulations have generally compared different force evaluation strategies [6] or have studied the trade-off between accuracy and run time [18]. A paper by Makino [13] studies the performance of

oct-trees versus nearest-neighbor trees (a data structure outside of the scope of this paper.)

In terms of data structures, an important related paper is the fair-split tree paper by Callahan and Kosaraju [7]. Although they developed their data structure to get an improved result for cluster-cluster algorithms, it turns out to be very close to the data structure we advocate in this paper. Some work in mainstream computational geometry also ties in directly with this, in particular work on the all-nearest-neighbor problem [19, 8].

2. Particle-cluster algorithm. The particle-cluster algorithm approximates the force that a set of particles exerts on a single particle by first dividing the set of particles into clusters and then summing the approximate force that each cluster exerts on the particle. The division into clusters is based upon a *spatial decomposition tree*. One version of the particle-cluster algorithm was originally described by Barnes and Hut [3]. They described the algorithm in terms of an oct-tree, although the *definition* of the algorithm does not depend upon the particular type of tree.

A hierarchical decomposition of a set S can be represented by a tree T. Each node of the tree has an associated subset of S. The sets associated with the children of a node form a partition of the set associated with the node. We often refer to the tree nodes as *cells*. Given a tree T, the following recursive algorithm computes the force on a particle x:

ForceEval(x, T)
  if *GoodApproximation(x, T)*
    return *ApproxForce(x, T)*
  else
    return $\displaystyle\sum_{T' \text{ child of } T} ForceEval(x, T').$

Let T_{part} denote the set of particles associated with the root of T. The routine *GoodApproximation* returns **true** if all of the particles in T_{part} are far enough from x that the force approximation is considered accurate. This test is referred to as the *opening criterion*. In this paper the opening criterion is based on the ratio of the diameter of T_{part} to the distance from x to the center of mass of T_{part}. Let d be the distance from x to the center of mass of T_{part}, s the diameter of T_{part}, and θ an accuracy parameter; the good approximation test returns **true** if $s \leq \theta d$ and *false* otherwise.[1] (Note that with this definition of the opening criterion, the accuracy increases as the value of θ decreases.) The routine *ApproxForce* gives an approximation of the force that the particles in T_{part} exert on x. The simplest approximation treats all of the mass of T_{part} as being at the center of mass, although in practice more sophisticated approximations are used.

The run time of the algorithm that solves the N-body problem by calling *Force-Eval* on every particle is generally considered to be $O(n \log n)$, although it does depend upon the distribution of the particles and the height of the tree. The algorithm spends almost all of its time performing the force evaluations. We can measure how good a particular tree is by counting the number of force evaluations that take place—this is a very accurate model of the run time.

[1] The results in this paper are fairly robust with respect to different choices of opening criteria. For example, an alternate opening criterion uses the radius (distance from the center of mass of T_{part} to the farthest point in T_{part}) instead of the diameter. The parameter θ controls the accuracy of the simulation. In practice values close to 1.0 are chosen.

3. Trees. Our problem is to determine the best tree data structure to use in the particle-cluster algorithm. We consider the algorithm to be fixed (meaning there is a given opening criterion and force evaluation function), and we look at different options for the spatial decomposition tree. We measure how good a tree is by the number of force evaluations that the algorithm performs in computing the force.

As an abstract problem, we could consider the following: For a fixed N-body algorithm, given a set of points S, find a tree T that minimizes the number of force evaluations. Instead of tackling this general problem (which appears to be difficult), we restrict ourselves to several tree data structures based upon orthogonal decomposition. There are many different methods for constructing an orthogonal decomposition tree [17]. We consider three options that can be applied independently, yielding eight separate tree data structures.

The first option is the degree of the decomposition. The natural choices are either a binary decomposition or a decomposition into orthants (quadrants in two dimensions, octants in three).

The second option is the method that keeps the decomposition balanced. We can either balance spatially, so that each cut is equidistant from the cell boundary, or balance based upon density, where the number of particles on each side of the split is equal.[2] When performing a binary split, the longest dimension of the cell is split.[3] A quad-tree is a quaternary spatial balanced decomposition, and a k-d tree is a binary density decomposition.

The third option in constructing trees relates to the definition of cell boundary. We consider the particles associated with a cell to be enclosed in a rectangular box, which is the boundary of the cell. One method for choosing the rectangular boxes is to subdivide boxes when cells are split. This method is referred to as *loose bounds* and is the way that boundaries are generally defined for quad-trees. An alternate way of defining the boundary is to take the smallest rectangle that encloses the points. This is referred to as *tight bounds*. When cells are split, the location (and even the dimension) of the split depends upon the boundary. This means that the tight bound and loose bound tree for the same particle set will have a different structure. Note that when tight bounds are used, the bounding box is computed *before* the cell is split; this is different from using loose bounds for computing all of the splits and then using tight bounding boxes when evaluating the opening criterion.

3.1. Results. We give a series of results that suggest that a binary spatially balanced decomposition with tight bounds is the best choice of those considered. We use L to denote the *separation ratio* of a set of points. This is the ratio of the maximum distance between points to the minimum distance between points.

- For k-d trees, we establish an $\Theta(n \log^d n \log L)$ bound for the particle-cluster algorithm in dimension d (where $d = 1, 2$, or 3). This shows that the spatially balanced decomposition algorithms are asymptotically superior to the density decomposition algorithms.
- We show that the particle-cluster algorithm takes $\Theta(n \log L)$ time for loose spatial decompositions. This is asymptotically optimal.
- We give a heuristic argument that shows that binary trees are more efficient

[2] There are a number of possible definitions of density balancing when a nonbinary split is used. The performance of nonbinary density balancing is sufficiently poor that the details are not worth exploring.

[3] An alternate method is to cycle through the dimensions in order when splitting cells. The results in this paper are applicable to this variation.

than higher-degree trees. In two dimensions, the expected number of force evaluations of a binary tree is 75% of the number for a quad-tree, and in three dimensions the expected number of force evaluations of a binary tree is 53% of an oct-tree. We also investigate other methods for improving the constant factor in the number of force evaluations.

- Intuitively, tight bounds should reduce the number of force evaluations because the diameters of the cells will be smaller. For binary trees, we show that tight bounds have the same asymptotic behavior as loose bounds. However, for quad-trees we show that the worst case for tight bounds is $\Omega(n^{3/2})$, so tight bounds can hurt the performance of non–binary tree–based algorithms.

4. Particle-cluster algorithm performance. We begin with a general lower bound on the performance of the particle-cluster algorithm. The result is that there exists a set of n particles with separation ratio L that requires $\Omega(n \log L)$ evaluations per particle for any decomposition tree. Suppose the cell C has diameter s, and the center of mass of C is at distance d from the point of evaluation. The opening criterion is to expand the cell if $s \geq \theta d$ for some fixed parameter θ.

THEOREM 4.1. *For any L, $n < L < e^{\theta n}$, there exists a set of n particles with separation ratio L such that any decomposition tree requires $\Omega(n \log L)$ force evaluations.*

Proof. The bad case distribution consists of $\frac{n}{2}$ particles evenly spaced on the interval $[0, \frac{n}{2})$ (the lower interval) and $\frac{n}{2}$ particles with geometrically increasing spacing on the interval $[L^{1/2}, L]$ (the upper interval). The argument shows that for each particle in $[0, \frac{n}{2})$ there are at least $\alpha \log L$ force evaluations. The particles in the upper interval are $\{p_{\frac{n}{2}+1}, \ldots, p_n\}$, with p_j located at $\hat{L}^j$ where $\hat{L} = L^{1/n}$.

Suppose that there is some particle p_i in the lower interval for which there are fewer than $\frac{\ln L}{4\theta}$ force evaluations. There must be some cell C containing at least $x = \frac{2\theta n}{\ln L}$ particles from the upper interval which is evaluated when computing the force on p_i. (The bound on L ensures that x is at least 2.) We show that the *normalized size* of C is at least θ, so that it fails the *GoodApproximation* test. (The normalized size of a cell with respect to a fixed particle is the cell's size divided by its distance from the particle.) We can consider the case where C contains exactly x particles, since a cell with more particles will have a greater normalized size. Let $C = \{p_j, \ldots, p_{x-1}\}$. We compute the normalized size of C with respect to the particle at 0 (which gives the smallest normalized size for all particles in the lower interval). The diameter s of C is $\hat{L}^{j+x-1} - \hat{L}^j$. The distance from the evaluation point to C is the center of mass of C. This is

$$\frac{1}{x}\sum_{i=j}^{x-1} \hat{L}^i = \frac{1}{x}\hat{L}^j \sum_{i=0}^{x-1} \hat{L}^i = \frac{1}{x}\hat{L}^j \frac{\hat{L}^x - 1}{\hat{L} - 1}$$
$$\leq \frac{1}{x}\hat{L}^j \frac{\hat{L}^x - 1}{\ln \hat{L}}.$$

Thus

$$\frac{s}{d} \geq \frac{x\hat{L}^j(\hat{L}^{x-1} - 1)\ln \hat{L}}{\hat{L}^j(\hat{L}^x - 1)}$$
$$\geq \frac{x \ln \hat{L}}{\hat{L}} = \frac{x \ln L}{n\hat{L}} \geq \frac{x \ln L}{2n} \geq \theta.$$

The one detail we haven't covered yet is the possibility that a cell in the decomposition does not consist of a consecutive set of points. Let C' be a cell containing x points, starting at position j, and let C be the cell containing $\{p_j, \ldots, p_{j+x-1}\}$. We show that $\frac{s_{C'}}{d_{C'}} \geq \frac{s_C}{d_C}$, which implies that any cell containing more than x particles fails the approximation test. The cell with x particles, starting at p_j, ending at p_{k+x-2}, with minimum normalized size is $\{p_j, p_k, \ldots, p_{k+x-2}\}$ (since this is the distance that maximizes center-of-mass distance subject to the constraints). Now, to minimize the normalized size over all cells of the form $\{p_j, p_k, \ldots, p_{k+x-2}\}$, we take $k = j - 1$, getting the cell C. □

L is a measure of the lack of uniformity in a set of particles. Since matter is distributed nonuniformly throughout the universe, simulations exhibit areas of high mass density and areas of low mass density. However, in practice L is bounded since computation is done with fixed precision operations. Thus, although L is a relevant factor, it is not going to be extremely large. It is probably worth distinguishing between $\log L$ and $\log n$ in stating results, but $\log L$ should still be viewed as being a relatively small quantity.[4]

5. Spatial decomposition. Many implementations of the particle-cluster algorithm have used quad-trees or oct-trees, including the implementation of Barnes and Hut. We give a performance bound for the oct-tree–based algorithm which shows how its performance depends upon the number and the distribution of particles. The key for the result is the observation that a sphere of radius r can only intersect a constant number of disjoint cubes with radius at least αr. This is at the heart of the packing arguments we allude to. The proof is similar to the correctness proofs of several all-nearest-neighbor algorithms [19, 8]. The necessity of accounting for "below evaluations" has been ignored by papers in the physics literature.

THEOREM 5.1. *Let D_{ave} be the average leaf depth of an oct-tree T. The number of force evaluations performed by the particle-cluster algorithm on T is $O(nD_{ave})$.*

Proof. We need to account for all of the force evaluations. A force evaluation occurs when we compute the force that a cell C exerts upon a particle x. The depth of a particle x in the tree is the depth of the leaf cell that contains x.

We distinguish between two cases of force evaluation in this analysis. An *above evaluation* occurs between x and C if the depth of C is less than or equal to the depth of x, and a *below evaluation* occurs between x and C if the depth of C is greater than the depth of x.

We begin by arguing that if x is at depth D, x is involved in at most αD above evaluations for an appropriate constant α (which depends upon the accuracy parameter θ). Let C be a cell on level j that is involved in an above evaluation with x. Let C' be the parent of C, d' the distance from x to the center of mass of C', and s' the diameter of C'. Since the *GoodApproximation* test fails for C', we must have $d' < \frac{s'}{\theta}$. The number of cells of diameter s' that are of distance no more than $\frac{s'}{\theta}$ from x is bounded by a constant. This, in turn, bounds the number of above evaluations for x with cells on level j. (The straightforward analysis gives a bound of $\alpha = 3\sqrt{3}\pi(1+\frac{1}{\theta})^3$ in three dimensions.)

We now account for the below evaluations. Suppose that there is a below evaluation between x and C. This evaluation is charged to the closest particle y to x that is

[4] It is a mistake to place much emphasis on pathological examples that use exponential differences in separation. However, it is also a mistake to view uniformly random data as the typical case for simulation.

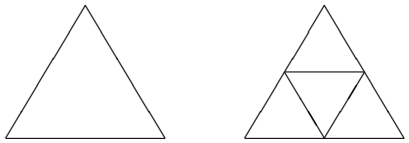


FIG. 5.1. *Triangular decomposition.*

inside of C. We now argue that if y is at depth D, y is charged for at most βD below evaluations for some constant β. The argument is similar to the above argument. Let x be a particle on level j that has a below evaluation charged to y. Suppose the evaluation is between x and the cell C, with C' the parent of C. Let d' be the distance from x to the center of mass of C', s the diameter of the leaf cell containing x, and s' the diameter of C'. We must have $d' < \frac{s'}{\theta}$ and $s \geq s'$. The number of cells of diameter s that are at a distance of no more than $\frac{s'}{\theta}$ from the center of mass of C' is bounded by a constant, giving the result. (The analysis gives a bound of $\beta = 4\sqrt{3}\pi(1 + \frac{1}{\theta})^3$ in three dimensions.)

Putting the two parts together, we have a bound of $O(nD_{ave})$ on the number of force evaluations. □

If the particles have separation ratio L, the depth of the oct-tree is $O(\log L)$, so we have an $O(n \log L)$ bound for the algorithm. The proof does not require that cells be cubes, just that they satisfy a packing lemma. The theorem is applicable to any decomposition that satisfies the following definition.

DEFINITION 5.2. *A decomposition is packable if there exists a constant β such that for every s, a sphere of radius s can interest at most β disjoint cells with diameter at least s.*

Since a binary spatial decomposition is packable, the run time of the particle-cluster algorithm for binary spatial trees also has an $O(n \log L)$ run time bound. One can come up with many other decompositions that are packable. For example, Figure 5.1 shows decomposition of the plane based upon equilateral triangles. This decomposition would satisfy the same asymptotic run time bound as a quad-tree.

6. Density decomposition.

***k*-d trees.** A k-d tree is a generalization of a balanced binary tree to higher dimensions. Each cell of a k-d tree divides a point set into two equal-size sets by splitting along a chosen dimension. A different dimension may be chosen at each level. (The implementation we consider is based upon splitting the longest side of each cell.) A k-d tree has the property that its height is logarithmic in the number of points. k-d trees have been used in a number of N-body implementations, including the early implementation of Appel [1] and in the University of Washington KD-grav code [9].

We show that in the worst case, an N-body algorithm using k-d trees is inferior to an algorithm with a spatially balanced decomposition. There is a bad case point set where the points all lie on a line. This one-dimensional bad case can then be repeated in two and three dimensions to get the worst-case results. We show that in dimension d (for $d = 1, 2$, or 3) the worst-case performance of the k-d tree algorithm is $\Theta(n \log^d n \log L)$, compared with $\Theta(n \log L)$ for spatially balanced trees such as oct-trees.[5] (We restrict attention to $d \leq 3$ since this is the range of practical interest.

[5]Since the lower bound is based upon contrived distributions, it is natural to ask if these apply to practice. It turns out that there are bad case examples based on data distributions used in real astrophysical simulations which give an $\Omega(n \log^2 n \log L)$ bound in three dimensions.

The results appear to generalize to higher dimensions.)

One-dimensional lower bound. We begin with the one-dimensional case. This is really the essence of the bad case for k-d trees. The two- and three-dimensional cases magnify the difficulty in order to get even worse bounds. We assume that L is in the range $n^2 \leq L \leq 3^{n^{1/2}}$ and give the proofs for the case where the accuracy parameter θ is equal to 1.

THEOREM 6.1. *The worst-case performance for the one-dimensional k-d tree–based particle-cluster algorithm is $\Omega(n \log L \log n)$.*

Proof. We construct the point set $P = \{p_1, \ldots, p_n\}$. Let $P_1 = \{p_1, \ldots, p_{n/2}\}$ and $P_2 = \{p_{n/2+1}, \ldots, p_n\}$. We get the bad case bound by counting the number of force evaluations that occur when determining the force that the points in P_2 exert on the points in P_1.

The points in P_1 are evenly spaced in the interval $[1, \frac{n}{2}]$.

The points in P_2 are divided into about $\log L$ intervals, with the sizes of the intervals increasing exponentially. When computing the force on a point in P_1, each one of these intervals is subdivided roughly $\log n$ times, giving the bound of $\Omega(\log L \log n)$ work per point in P_1.

Let I_k be the interval $[3^{k-1}, 3^k]$. Points from P_2 are assigned to the intervals $I_{\log_3 n+1}, \ldots, I_{\log_3 L}$. Let $w = \log_3 L - \log_3 n - 1$ be the number of intervals. For convenience, we assume that both n and w are powers of 2. (The argument can be modified to remove this assumption.) We assign $\frac{n}{2w}$ points to I_k ($\log_3 n < k \leq \log_3 L$). The points assigned to I_k are evenly spaced on the first half of the interval (meaning the subinterval $[3^{k-1} + 1, \frac{3}{2}3^{k-1}]$) with the exception of one point, which is assigned to the high end of the interval.

We now consider the cost of traversing the tree for P_2, using a point from P_1. The points in P_2 are represented by a tree, which has a node i_k corresponding to each of the intervals I_k, for $\log_3 n < k \leq \log_3 L$. From each point in P_1, we visit each of these nodes, and from these nodes we traverse the tree to some leaf. The node i_k has $n/2w$ descendents, so it has height $\log(n/2w)$. Since we are assuming $n^2 \leq L \leq 3^{n^{1/2}}$, it follows that the bound is $\Omega(n \log L \log n)$. □

The only modification necessary to handle the case of an arbitrary θ is to use a base of 3θ instead of a base of 3 in defining the intervals. This gives the bound of $\Omega(n \log_\theta L \log n)$ for arbitrary θ.

One-dimensional upper bound. We now prove a corresponding upper bound on the run time of the one-dimensional particle-cluster algorithm.

THEOREM 6.2. *The worst-case performance for the one-dimensional particle-cluster algorithm using a k-d tree is $O(n \log L \log n)$.*

Proof. A k-d tree has $\log n$ levels. For a fixed point p, consider the set of nodes visited while evaluating the force on p. These nodes can be divided into nodes for intervals to the left of p, including p, and to the right of p. There are at most $\log n$ intervals that contain p. We will now consider the nodes to the right of p and for convenience we assume that p is located at the origin.

Let I_1 and I_2 be on the same level with different parents, and suppose p visits both I_1 and I_2. Let the parents of I_1 and I_2 be I_1' and I_2', respectively, with $I_1' = [x_1, y_1]$ and $I_2' = [x_2, y_2]$. Assuming that I_2 is to the right of I_1, we must have $x_2 \geq (1+\theta)x_1$. (If that were not the case, then the interval I_1' would not have been expanded in the evaluation of p.) It follows that there are at most $\log_{(1+\theta)} L$ intervals to the right of p on a level that are evaluated for p. Thus the number of elements evaluated for p is at most $4 \log_{(1+\theta)} L \log n + \log n$, so the total work is $O(n \log L \log n)$. □

Higher dimensional lower bounds. The bad case generalizes to higher dimensions in a fairly natural manner. The way we do this is by repeating the one-dimensional construction $\log n$ times so that we have $\log n$ columns, each of which gives the one-dimensional bad case. We will have $\frac{n}{2}$ particles in the first column, $\frac{n}{4}$ in the second, $\frac{n}{8}$ in the third, and so on. This construction gives us the following results.

THEOREM 6.3. *The worst-case performance for the two-dimensional particle-cluster algorithm using a k-d tree is* $\Omega(n \log L \log^2 n)$.

Proof. We arrange the n particles in $\log n$ columns, where column C_i has $\frac{n}{2^i}$ particles and one additional particle. The particles in column C_i have x coordinate i and y coordinates that correspond to the one-dimensional construction on $\frac{n}{2^i}$ particles. The one other particle has coordinates $(2L, 0)$. The construction of the k-d tree creates the columns as skinny rectangles, since the extra point at $(2L, 0)$ leads to a series of vertical subdivisions. Figure 6.1 shows how the addition of the single point causes the skinny vertical rectangles to be formed. Each of the vertical rectangles contains points arranged as in the one-dimensional lower bound.

The columns contain about $\frac{n}{2}$ particles that are within distance $\frac{n}{2}$ of the origin. Each one of these particles is going to have $\log L(\log n - i - \log \log L)$ interactions with intervals from C_i, which gives the $\Omega(n \log L \log^2 n)$ bound. □

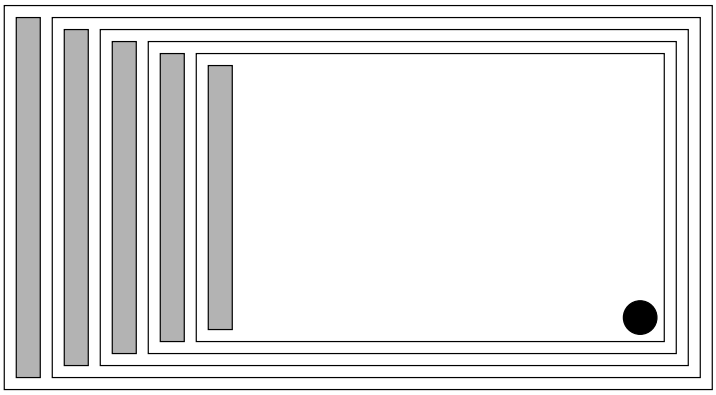


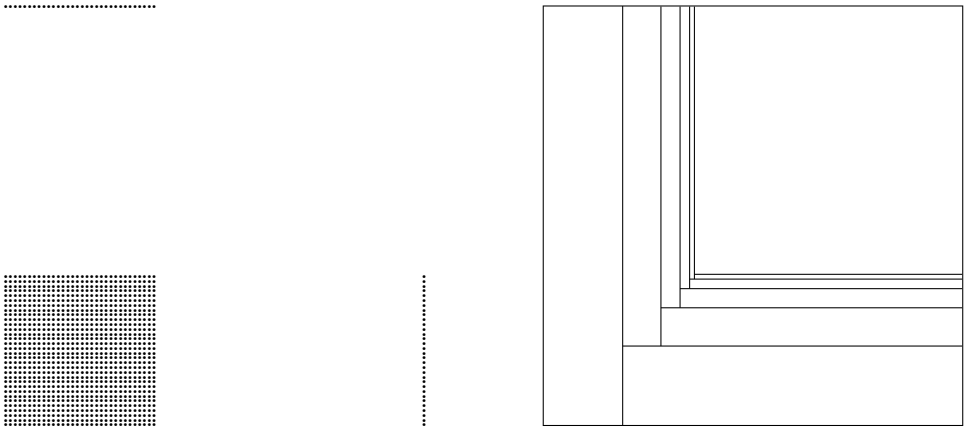
FIG. 6.1. *Decomposition in skinny rectangles.*

In three dimensions, we take a stack of $\log n$ two-dimensional bad case examples to pick up another factor of $\log n$. The same trick is used in which a single, far-off particle causes the k-d tree to have a collection of slices with a specific structure.

THEOREM 6.4. *The worst-case performance for the three-dimensional particle-cluster algorithm using a k-d tree is* $\Omega(n \log L \log^3 n)$.

These examples show that in the worst case k-d trees are a factor of $\log^3 n$ worse than spatially balanced trees. However, the bad case distributions are contrived and relatively delicate, so it is natural to ask if k-d trees are problematic from a practical point of view: Is the bad behavior observed in real simulations? Consider the two-dimensional distribution shown in Figure 6.2A, where n particles are placed on a $\sqrt{n} \times \sqrt{n}$ grid and additional particles are placed at (i, L) and (L, i) for $1 \leq i \leq \sqrt{n}$. When these particles are divided by a k-d tree, a collection of long skinny rectangles is generated, as shown in Figure 6.2B. Each of the long skinny rectangles is further decomposed in a collection of long skinny rectangles. When the force is computed on a particle, the long skinny rectangles will fail the *GoodApproximation* test. It is straightforward to verify that this leads to $\Omega(\log^2 n)$ force evaluations per

particle. Although this is not as bad as the construction used in Theorem 6.3, it is worse than the spatially balanced decomposition. Significantly, this distribution is similar to ones used in simulations that investigate how the internal structure of a galaxy is influenced by neighboring galaxies [14]. The main galaxy is modeled at high resolution, and the other galaxies are modeled as single particles; thus the particles near the origin correspond to the main galaxy, and the particles at (i, L) and (L, i) are the other galaxies in the cluster. Simulations of this type of structure using the k-d tree algorithm were observed to have serious performance problems.



A: Particle distribution B: Partial k-d tree decomposition

FIG. 6.2. *Bad case distribution from galaxy simulation.*

Higher dimensional upper bounds. We now generalize the upper bound result to cover two dimensions. The situation is more complex than the one-dimensional case because we must take the shape of the cells into account. We show that each particle evaluates $O(\log^2 n \log L)$ cells. We begin with a technical lemma, which restricts attention to cells of a given level that intersect a region.

LEMMA 6.5. *Let B be an $s \times s$ region. The number of cells of level j with diameter greater than s intersecting B is $O(\log n)$.*

Proof. A cell is said to be *fat* if both of its sides have length at greater than $\frac{s}{2}$, and it is *skinny* if one side has length longer than $\frac{s}{2}$ and the other has length at most $\frac{s}{2}$.

Our first observation is that the number of fat cells with two fat children that both intersect B is bounded by a constant. This implies that there are $O(\log n)$ fat cells intersecting B (since if we consider the subtree made up of fat cells that intersect B, the number of nodes of degree two is bounded by a constant, and the tree height is at most $\log n$). The number of skinny cells that intersect B and are children of fat cells is also $O(\log n)$. The descendents of a cell S on a given level form a partition of S. If S is skinny with a shorter side of length l, and S' is a skinny descendent of S, S' also has a shorter side of length l since the longest side is partitioned and once both sides of a cell have length at most $\frac{s}{2}$ the cell is no longer considered skinny. This means that S can have at most three skinny descendents on any given level that intersect B. This gives the $O(\log n)$ bound on the number of cells of diameter at least s intersecting B on any level. □

We can now give the bound on the number of evaluations on a fixed particle, which gives the bound on the performance of the algorithm.

THEOREM 6.6. *In two dimensions, a particle p is involved in $O(\log^2 n \log L)$ evaluations.*

Proof. Consider cells of diameter between s and $2s$ which are expanded when computing the force on p. Each one of these cells must be within a distance of $\frac{2s}{\theta}$, since they fail the *GoodApproximation* test. Lemma 6.5 implies that the number of cells of diameter at least s within this distance is $O(\log^2 n)$. Since the maximum cell size is L, we pick up the factor of $\log L$ to sum over all groups of cells. □

Extending the result to three dimensions does not require any new ideas. We pick up an extra logarithmic factor because we need to consider cells that have a small length on one or two sides.

THEOREM 6.7. *In three dimensions, a particle p is involved in $O(\log^3 n \log L)$ evaluations.*

Proof (sketch). The proof follows that of Theorem 6.6. A lemma analogous to Lemma 6.5 is used, which shows that a cube of size s can be intersected by at most $O(\log^2 n)$ cells of size s of a fixed level. □

7. Degree bounds. We now turn our attention to the problem of determining what the best decomposition strategy is for a spatially balanced decomposition. In our model, where we count only force evaluations, we can do better than an oct-tree decomposition. A binary decomposition splits the longest dimension of a cell evenly. This means that three levels of splitting accomplishes exactly the same decomposition as a single level of an oct-tree. However, the binary tree may require fewer force evaluations since there are cases where a single binary split is sufficient for cells to pass the accuracy test, so that two evaluations are done instead of eight. In this section, first we give an analysis that supports the use of binary trees over oct-trees, and then we consider other methods of decomposition which give additional improvements.

7.1. Interaction regions. It is possible to adapt Theorem 5.1 to give a bound that includes the constant factor, although it will be pessimistic. A more accurate "heuristic" analysis based upon the *interaction region* of a cell can be used to evaluate different decomposition strategies. A cell C partitions the points in space into two regions, G_C, the points where the force approximation of C is considered good, and B_C, the points where the approximation is bad. G_C and B_C are the outside and the inside of a sphere of radius $\theta^{-1}s$ centered at the center of mass of C. If C' is the parent of C, the interaction region of C is defined to be $G_C \cap B_{C'}$. The interaction region of C is roughly the set of points for which a force evaluation will use the approximation from the cell C. Figure 7.1 shows the interaction region for cell C (the upper right quadrant of C') as a doughnut with an off-center hole.

We can use the interaction region to determine the probability that the evaluation at a random point involves a particular cell. (The interaction regions can also be used for an average case analysis with respect to a random set of points [6, 16].) Experimental results show that the interaction area, or the size of the interaction region, is a very good predictor of the algorithm's run time [6, 2].

We give a pair of results which suggest that a binary tree requires fewer force evaluations than a quad-tree or an oct-tree for the same set of points.

THEOREM 7.1. *Assuming that the center of mass is at the center of each cell and $\theta < 1.19$, the total interaction area of a binary spatial decomposition tree is 75% of the interaction area of a quad-tree for the same set of points.*

Proof. Consider a $k \times k$ cell C. In a quad-tree, C has four children, each of size $\frac{k}{2} \times \frac{k}{2}$. Each child has an interaction region of size $\frac{3}{2}\pi\theta^{-2}k^2$, so the total interaction area of the children is $6\pi\theta^{-2}k^2$.

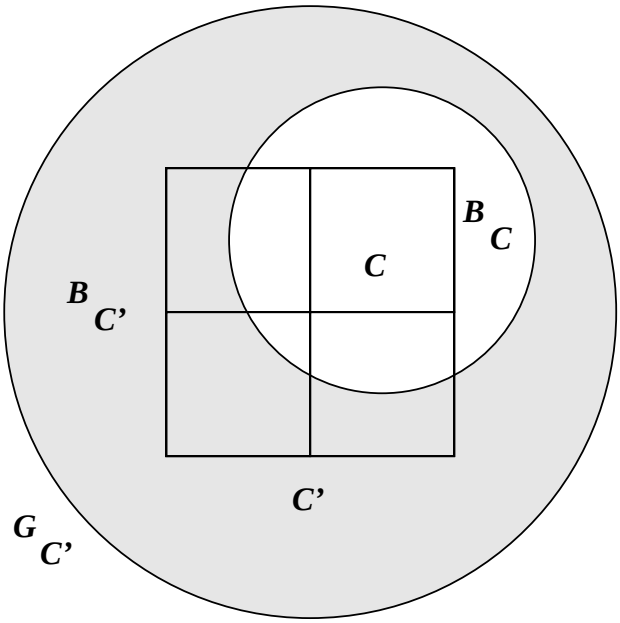


FIG. 7.1. *Interaction region.*

Now consider a $k \times k$ cell C in a binary tree. C has two $k \times \frac{k}{2}$ cells as children and has four $\frac{k}{2} \times \frac{k}{2}$ children as grandchildren. The interaction area of each of the children is $\frac{3}{4}\pi\theta^{-2}k^2$ and the interaction area of each of the grandchildren is also $\frac{3}{4}\pi\theta^{-2}k^2$, giving a total interaction area of $\frac{9}{2}\pi\theta^{-2}k^2$.

The bound on θ ensures that B_C and $G_{C'}$ are disjoint (otherwise a correction factor must be added to account for a small crescent where $G_{C'}$ intersects B_C). □

We can get an even stronger result in three dimensions.

THEOREM 7.2. *Assuming that the center of mass is at the center of each cell and* $\theta < 0.93$, *the total interaction area of a binary spatial decomposition tree is* 53% *of the interaction area of an oct-tree.*

7.2. Other decompositions. Even though it is only a heuristic argument that ties the size of the interaction region to the performance of N-body algorithms, a strong correlation with experimental evidence has been observed. We shall treat the interaction region as the figure of merit of a decomposition. When comparing different decompositions with the same degree, we want to look at the size of interaction regions of cells with the same volume, with the smaller interaction region giving better performance.

We begin by comparing a quad-tree decomposition with the triangular decomposition shown in Figure 5.1. We show that the triangular decomposition will probably be less effective than a quad-tree.

THEOREM 7.3. *A quad-tree cell of unit area has interaction region of area* $6\pi\theta^{-2}$, *and a triangular cell of unit area has interaction region of area* $\frac{12}{\sqrt{3}}\pi\theta^{-2}$.

Hence, the size of the interaction area of the triangular decomposition is 15% larger than the interaction area of the quad-tree decomposition.

A more interesting case is the comparison of a binary decomposition of a square

with a binary decomposition of a rectangle with aspect ratio $1 : \sqrt{2}$. The result is that the latter has an interaction region that is about 5% smaller than the former. This is interesting since it shows that the binary decomposition of the square is not optimal.

THEOREM 7.4. *In two dimensions, the interaction area of a binary tree based on cells with aspect ratio* $1 : \sqrt{2}$ *is* 94.2% *of the interaction area of a binary tree based on square cells.*

Proof. Let C be a unit square. C is decomposed into $1 \times \frac{1}{2}$ cells C_1' and C_2' that in turn are decomposed into $\frac{1}{2} \times \frac{1}{2}$ cells C_1'', C_2'', C_3'', and C_4''. C_1' has interaction area $\pi\theta^{-2}(2 - \frac{5}{4})$ and C_1'' has interaction area $\pi\theta^{-2}(\frac{5}{4} - \frac{1}{2})$. This gives a total interaction area of $\frac{9}{2}\pi\theta^{-2}$ for the six cells.

Let $\hat{C}$ be a $\sqrt[4]{2} \times \frac{1}{\sqrt[4]{2}}$ rectangle. $\hat{C}$ is decomposed into $\frac{1}{\sqrt[4]{2}} \times \frac{\sqrt[4]{2}}{2}$ cells $\hat{C}_1'$ and $\hat{C}_2'$ that in turn are decomposed into $\frac{\sqrt[4]{2}}{2} \times \frac{1}{2\sqrt[4]{2}}$ cells $\hat{C}_1''$, $\hat{C}_2''$, $\hat{C}_3''$, and $\hat{C}_4''$. $\hat{C}_1'$ has interaction area $\pi\theta^{-2}((\sqrt{2}+\frac{1}{\sqrt{2}})-(\frac{1}{\sqrt{2}}+\frac{\sqrt{2}}{4}))$ and $\hat{C}_1''$ has interaction area $\pi\theta^{-2}((\frac{1}{\sqrt{2}}+\frac{\sqrt{2}}{4})-(\frac{\sqrt{2}}{4}+\frac{1}{4\sqrt{2}}))$ This gives a total interaction area of $3\sqrt{2}\pi\theta^{-2}$ for the six cells.

If we evaluate the expressions, we see that the interaction area in the second case is 5.8% less. ∎

We can prove a similar result in three dimensions, giving about a 10% improvement over the decomposition from a cubical cell.

THEOREM 7.5. *In three dimensions, the interaction volume of a binary tree based on cells with aspect ratio* $1 : 2^{1/3} : 2^{2/3}$ *is* 89.7% *of the interaction volume of a binary tree based on cubical cells.*

We now consider a different approach for reducing the number of force evaluations: Instead of using a tree data structure to represent the data structure, use a directed acyclic graph (DAG). The DAG allows a choice of decompositions for cells. For example, with a binary decomposition, a square cell, $\square$, can be decomposed horizontally, $\boxminus$, or vertically, $\boxplus$. The idea is that based upon the location of the point that we are evaluating the force on, we choose the decomposition that minimizes the amount of work. The decompositions $\boxminus$ and $\boxplus$ both decompose into $\boxplus$, so the extra storage required is minimal.

We can show that this data structure reduces the interaction area of the binary spatial decomposition.

THEOREM 7.6. *Assume that the center of mass is at the center of each cell and* $\theta = 1.0$. *In two dimensions the interaction area of the DAG is* 93% *of the interaction area of a tree for a square cell, and in three dimensions the interaction volume of the DAG is* 88% *the interaction volume of a tree for a cubical cell.*

Proof. For the two-dimensional case, we superimpose two binary decompositions, one that is horizontal and one that is vertical. We consider the regions of space that result in different sets of force evaluations and choose the decomposition that minimizes the number of force evaluations. We need only to count the number of force evaluations with respect to the rectangular regions, since a square region will either be evaluated or not be evaluated independent of whether the vertical or horizontal decomposition is used. Figure 7.2 shows a decomposition of space into regions corresponding to how many evaluations a point requires with respect to large rectangles with a horizontal or a vertical decomposition. Figure 7.3 shows the regions (marked X and Y) where a horizontal decomposition gives a savings over a vertical decomposition. In the X region the two horizontal rectangles are used, instead of decomposing a rectangle into squares, and in the Y region one of the rectangles is used and the

TABLE 7.1
Interaction sizes for decomposition DAG.

θ	Two-d	Three-d
0.8	0.947	0.907
0.9	0.940	0.895
1.0	0.934	0.880
1.1	0.927	0.870
1.2	0.921	0.859

other is split into squares. Computing this area (which we did numerically) gives the cost savings.

In three dimensions, there are 12 separate decompositions to consider. The bound was computed numerically. Table 7.1 shows the interaction areas and volumes in two and three dimensions for selected values of θ. The results are scaled so that the tree has interaction volume 1.0. □

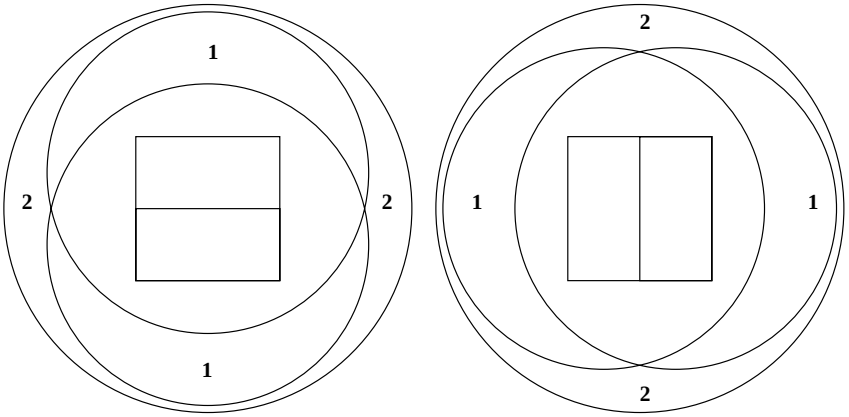


FIG. 7.2. *Interaction regions.*

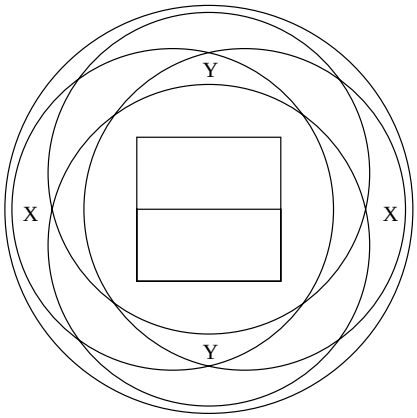


FIG. 7.3. *Regions with lower horizontal cost than vertical cost.*

The significance of these results is that they show that it is possible to improve over the natural decompositions with a small amount of extra work. (However, the

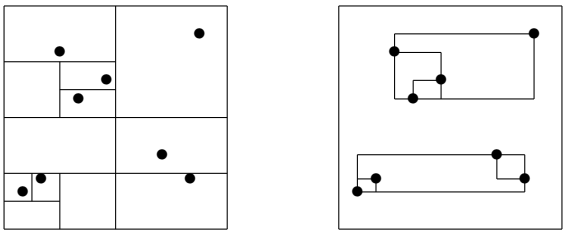


FIG. 8.1. *Binary decomposition with loose and tight bounds.*

two schemes—optimizing the aspect ratio and using a DAG instead of a tree—cannot be combined to gain additional benefit.) Given the tremendous amount of CPU time devoted to simulation algorithms, even a small improvement in the constant factor translates into a big real-time savings.

8. Cell bounds. When we compute the boundary of a cell, we have a choice of using either "loose" or "tight" bounds. The bounds are loose if they are inherited from the bounds of the parent cell, and they are tight if they are the actual bounds of the point set. See Figure 8.1 for an example that shows the distinction between the bound types. Note that the region boundaries are computed as the tree is constructed so that the tree structure depends upon the type of boundary.

The potential advantage of tight bounds is that region sizes will be smaller so that cells are more likely to pass the good approximation test and reduce the number of evaluations. Experimental results show that tight bounds generally do lead to a work savings, especially when the data is clustered.

To give theoretical support for using tight bounds, we show that binary spatial decomposition trees with tight bounds have the same asymptotic run time as the trees with loose bounds. However, this result holds only for binary trees. We show that for quad-trees, tight bounds can lead to trees that are much worse than trees with loose bounds.

8.1. Binary spatial decomposition with tight bounds. We now show that an algorithm that uses binary trees with tight bounds has the same asymptotic run time as an algorithm with loose bounds. We prove this by giving a packing lemma that says that only a constant number of disjoint cells with diameter at least s can intersect a sphere of radius θs. Using the lemma, the argument of Theorem 5.1 applies.

The following lemma is based on a packing lemma for fair-split trees proved by Callahan and Kosaraju [7]. (We establish the lemma in two dimensions; the generalization to three dimensions is straightforward.)

LEMMA 8.1. *The maximum number of disjoint cells of diameter at least s of a tight binary spatial decomposition tree that can intersect a ball of radius θs is bounded by a constant.*

Proof. For each cell C, we define an outer rectangle $C.outer$. If C has children C' and C'', then $C.outer$ is divided to give $C'.outer$ and $C''.outer$. C' and C'' are the contractions of $C'.outer$ and $C''.outer$ to the smallest rectangles enclosing the point sets. The outer rectangles give a valid spatial decomposition of the space. Figure 8.2 illustrates the outer rectangles for a cell C being split into cells C' and C''.

We show that if C' is a child of C, then

$$\min(C'.outer.height, C'.outer.width) \geq \frac{1}{2}\max(C.height, C.width).$$

If C is the root, then the bound holds (we assume that the outer rectangle of the root is a square). Suppose C is not the root. Let $\hat{C}$ be the parent of C and suppose the bound holds for C and $\hat{C}$. We can assume that C' is formed by a horizontal cut of C, so $C.height \geq C.width$. We have

$$C'.outer.height \geq \frac{1}{2}C.height \geq \frac{1}{2}C.width$$

and

$$C'.outer.width = C.outer.width \geq \frac{1}{2}\hat{C}.width \geq \frac{1}{2}C.width.$$

If a cell C' has diameter at least s, then

$$\max(C.height, C.width) \geq \max(C'.height, C'.width) \geq \frac{s}{\sqrt{2}},$$

with C the parent of C'. Thus, we have

$$\min(C'.outer.height, C'.outer.width) \geq \frac{s}{2\sqrt{2}}.$$

This implies that at most a constant number of disjoint cells with diameter at least s can intersect a ball of radius θs. □

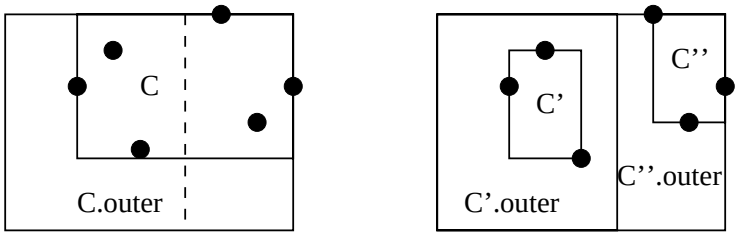


FIG. 8.2. *Cell decomposition into outer regions cost.*

THEOREM 8.2. *The run time of the particle-cluster algorithm using a tight binary spatial decomposition tree on a set of n particles with separation ratio L is* $O(n \log L)$.

Proof. We use Lemma 8.1 with Theorem 5.1 to get the bound. The height of the TBSD tree is $O(\log L)$. □

8.2. Tight bounds for quad-trees. The result that the use of tight bounds does not increase the asymptotic worst-case amount of work that a binary tree–based algorithm performs is not surprising—intuitively the use of more accurate cell bounds should speed up the computation. However, a similar result does not hold for higher degree trees. We show that the use of tight bounds can drastically increase the amount of work performed by a quad-tree–based algorithm. The bound below shows that the amount of work can increase by a factor of $\Omega(n^{1/2}/\log n)$.

The problem is that if tight bounds are used, the regions can have a high aspect ratio. The reason that this is bad is that long skinny regions do not satisfy a suitable packing lemma (such as Lemma 8.1) so that it is not possible to get a good bound on the number of interactions that a particle will have with regions of a particular size. The basic idea used to construct the bad example in two dimensions is to use a set of points distributed uniformly in a rectangle with dimensions h and w where $h \gg w$.

Since the bounds are tight, the bounding box for the region is an $h \times w$ rectangle. The quad-tree splits each dimension evenly so at each level rectangles with aspect ratio $h : w$ are formed. When the force on a particle is evaluated there are a large number of nearby rectangles that must be used. The two-dimensional example generalizes to three dimensions and gives an $\Omega(n^{5/3})$ bound.

THEOREM 8.3. *There exists a set of n points with separation ratio $n^{O(1)}$ for which the particle-cluster algorithm using a quad-tree with tight bounds has $\Omega(n^{3/2})$ run time.*

Proof. For convenience, assume that n is a power of 4 and $\theta = 1$. We place a particle at $(i, j\sqrt{n})$ for $0 \le i, j < n$. The quad-tree decomposition divides this set in cells with aspect ration $1 : \sqrt{n}$. The cells immediately above the leaves each contain four points and have dimensions $1 \times \sqrt{n}$. When computing the force on a particle, we examine at least $\sqrt{n}/2$ of these cells, giving $\Omega(n^{3/2})$ work over all. □

9. Conclusions and future work. The argument of this paper is that a binary, spatially balanced decomposition tree with tight bounds is the best data structure to use in N-body simulation algorithms. We have shown that spatial decompositions are asymptotically superior to density-based decompositions. The result led to changes in one of the major N-body codes in use for astrophysical research. By analyzing the interaction areas of cells, we expect close to a 50% reduction in the number of force evaluations for a binary tree as opposed to an oct-tree. We also show that tight bounding boxes can be used for binary trees without hurting run time bounds, while for higher degree tight bounding boxes can cause a substantial slowdown.

We have established a set of theoretical results that support specific design choices in practical N-body algorithms. The question of which is the best decomposition strategy to use is still open. Lemma 7.6 raises the intriguing possibility that DAG-based algorithms could be superior to tree-based algorithms. There is the potential to extend our results to other N-body algorithms beyond the particle-cluster algorithms and also to look at a wider collection of tree decompositions.

REFERENCES

[1] A. W. APPEL, *An efficient program for many-body simulation*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 85–103.

[2] R. J. ANDERSON AND S. D. SANDYS, *An experimental study of tree data structures for N-body simulation*, manuscript, 1996.

[3] J. E. BARNES AND P. HUT, *A hierarchical $O(N \log N)$ force-calculation algorithm*, Nature, 324 (1986), pp. 446–449.

[4] J. A. BOARD, Z. S. HAKURA, W. D. ELLIOT, D. C. GRAY, W. J. BLANKE, AND J. F. LEATHRUM, *Scalable Implementations of Multipole-Accelerated Algorithms for Molecular Dynamics*, Technical Report 94-002, Department of Electrical Engineering, Duke University, Durham, NC, 1994.

[5] J. A. BOARD, Z. S. HAKURA, W. D. ELLIOT, AND W. T. RANKIN, *Scalable Variants of Multipole-Accelerated Algorithms for Molecular Dynamics Applications*, Technical Report 94-006, Department of Electrical Engineering, Duke University, Durham, NC, 1994.

[6] G. BLELLOCH AND G. NARLIKAR, *A practical comparison of N-body algorithms*, manuscript, 1995.

[7] P. B. CALLAHAN AND S. R. KOSARAJU, *A decomposition of multi-dimensional point-sets with applications to k-nearest-neighbors and n-body potential fields*, J. Assoc. Comput. Mach., 42 (1995), pp. 67–90.

[8] K. L. CLARKSON, *Fast algorithms for the all nearest neighbors problem*, in 24th Symposium on Foundations of Computer Science, 1983, pp. 226–232.

[9] M. DIKAIAKOS AND J. STADEL, *A Performance Study of Cosmological Simulations on Message-Passing and Shared-Memory Multiprocessors*, 1995, available from http://www-hpcc.astro.washington.edu/.

[10] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.
[11] L. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems*, MIT Press, Cambridge, MA, 1988.
[12] J. G. Jernigan and D. H. Porter, *A tree code with logarithmic reduction of force terms, hierarchical regularization of all variables and explicit accuracy controls*, J. Astrophys. Suppl., 71 (1989), pp. 871–893.
[13] J. Makino, *Comparison of two different tree algorithms*, J. Comput. Phys., 88 (1990), pp. 393–408.
[14] B. Moore, N. Katz, G. Lake, A. Dressler, and A. Oelmer, Jr., *Galaxy harassment and the evolution of clusters of galaxies*, Nature, 379 (1996), pp. 613–616.
[15] J. H. Reif and S. R. Tate, *N-body Simulation* I: *Fast Algorithms for Potential Field Evaluation and Trummers's Problem*, Technical Report N-96-002, Department of Computer Science, University of North Texas, Denton, TX, 1996.
[16] J. K. Salmon, *Parallel Hierarchical N-body Methods*, Ph.D. thesis, California Institute of Technology, Pasadena, CA, 1990.
[17] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison–Wesley, Reading, MA, 1989.
[18] J. K. Salmon and M. S. Warren, *Skeletons from the treecode closet*, J. Comput. Phys., 111 (1994), pp. 136–155.
[19] P. M. Vaidya, *An optimal algorithm for the all-nearest-neighbors problem*, in 27th Symposium on Foundations of Computer Science, 1986, pp. 117–122.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 1941–1969

THE POWER OF VACILLATION IN LANGUAGE LEARNING*

JOHN CASE†

Abstract. Some extensions are considered of Gold's influential model of language learning by machine from positive data. Studied are criteria of successful learning featuring convergence in the limit to vacillation between several alternative correct grammars. The main theorem of this paper is that there are classes of languages that can be learned if convergence in the limit to up to $(n+1)$ exactly correct grammars is allowed but which cannot be learned if convergence in the limit is to no more than n grammars, where the no more than n grammars can each make finitely many mistakes. This contrasts sharply with results of Barzdin and Podnieks and, later, Case and Smith for learnability from both positive and negative data.

A subset principle from a 1980 paper of Angluin is extended to the vacillatory and other criteria of this paper. This principle provides a necessary condition for avoiding overgeneralization in learning from positive data. It is applied to prove another theorem to the effect that one can optimally eliminate half of the mistakes from final programs for vacillatory criteria if one is willing to converge in the limit to infinitely many different programs instead.

Child language learning may be sensitive to the order or timing of data presentation. It is shown, though, that for the vacillatory success criteria of this paper, there is no loss of learning power for machines which are *in*sensitive to order in several ways simultaneously. For example, *partly set-driven* machines attend only to the *set* and *length of sequence* of positive data, not the actual sequence itself. A machine $\mathbf{M}$ is *weakly n-ary order independent* $\stackrel{\text{def}}{\Leftrightarrow}$ for each language L on which, for some ordering of the positive data about L, $\mathbf{M}$ converges in the limit to a finite set of grammars, there is a finite set of grammars D (of cardinality $\leq n$) such that $\mathbf{M}$ converges to a subset of this same D for *each* ordering of the positive data for L. The theorem most difficult to prove in the paper implies that machines which are simultaneously partly set-driven and weakly n-ary order independent do not lose learning power for converging in the limit to up to n grammars. Several variants of this theorem are obtained by modifying its proof, and some of these variants have application in this and other papers. Along the way it is also shown, for the vacillatory criteria, that learning power is not increased if one restricts the sequence of positive data presentation to be computable. Some of these results are nontrivial lifts of prior work for the $n = 1$ case done by the Blums; Wiehagen; Osherson, Stob, and Weinstein; Schäfer; and Fulk.

Key words. computational learning theory, inductive reference, language learning, recursion theory, topology

AMS subject classifications. 68Q, 68T05, 68S05, 03D, 92J40, 54E51, D0944

PII. S0097539793249694

1. Introduction.

In [46] Gold introduced his seminal model of language learning: Imagine, as pictured in (1.1) just below, a machine $\mathbf{M}$ being fed data about membership in a (formal) language L and, as a result, outputting over time a series of grammars $p_0, p_1, p_2, \ldots, p_t, p_{t+1}, \ldots$ *conjectured* to be for L:[1]

$$p_0, p_1, p_2, \ldots, p_t, p_{t+1}, \ldots \leftarrow \mathbf{M} \leftarrow \text{data re } L. \tag{1.1}$$

*Received by the editors May 28, 1993; accepted for publication (in revised form) September 22, 1997; published electronically June 3, 1999. This research was supported in part by NSF grant CCR-8713846. This paper is an expansion with corrections of J. Case, *The Power of Vacillation*, in Proceedings of the Workshop on Computational Learning Theory, D. Haussler and L. Pitt, eds., Morgan Kaufmann, San Mateo, CA, 1988, pp. 133–1424.

http://www.siam.org/journals/sicomp/28-6/24969.html

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

[1] We have oriented the arrows in (1.1) so that elements later in the series $p_0, p_1, p_2, \ldots, p_t, p_{t+1}, \ldots$ will *correctly* appear to be coming out of the machine $\mathbf{M}$ later: they will appear closer to $\mathbf{M}$ than earlier elements of the series.

For our present purposes, it will suffice to consider two kinds of data presentation and one kind of success from [46]. For expository convenience, and, as noted in section 2 below, without loss of generality, we can consider all languages L to be subsets of the set of nonnegative integers.

Data about L are *either*

1. *informant*, a listing of every nonnegative integer with a clear indication of whether or not it is in L, *or*
2. *text*, an arbitrary listing of all and only the elements of L.

Gold took quite seriously, as a model of child language learning, the case of data presentation by arbitrary text, where $\mathbf{M}$ receives all and only *positive* information about L. Justification for this point of view can be found, for example, in [9, 13], where it is noted from field work that children don't need corrections to learn language.

Regarding successful language learning, referring to (1.1) above: for Gold, machine $\mathbf{M}$ *identifies* language $L \stackrel{\text{def}}{\Leftrightarrow} \mathbf{M}$ fed any text for L, outputs a corresponding sequence $p_0, p_1, p_2, \ldots$ such that, for some t, $p_t = p_{t+1} = p_{t+2} = \cdots$ and p_t is a correct grammar for L. In other words, $\mathbf{M}$ *identifies* $L \Leftrightarrow$ on each text for L, the corresponding conjectures of $\mathbf{M}$ converge, in the limit, to some fixed final conjecture, and that final conjecture is correct.[2] Gold showed that *no* $\mathbf{M}$ so identifies the entire class of regular languages [48], but *some* $\mathbf{M}$ *does* identify the class of finite languages. Angluin [1, 2] presents other classes $\mathcal{L}$ natural from the perspective of formal language theory such that some $\mathbf{M}$ identifies each language in $\mathcal{L}$.

Many *cognitive scientists* seek to model all of cognition by computer program [77, 50], and Gold's model of language learning from text (positive information) by machine has been very influential in contemporary theories of natural language and in mathematical work explicitly motivated by its possible connection to human language learning (see, for example, [76, 93, 94, 66, 68, 8, 44, 15, 69, 70, 38, 39, 53, 5]).

In the present paper we consider some new criteria of success extending Gold's basic model above. Suppose that we fix an integer $n > 0$. Consider the following criterion of success (again based on (1.1) above). We say that $\mathbf{M}$ $\mathbf{TxtFex}_n$-*identifies* $L \stackrel{\text{def}}{\Leftrightarrow} \mathbf{M}$, on any text for L, outputs corresponding conjectures $p_0, p_1, p_2, \ldots$ such that there is a t for which

1. the sequence $p_t, p_{t+1}, p_{t+2}, \ldots$ contains at most n distinct grammars, *and*
2. each of the grammars $p_t, p_{t+1}, p_{t+2}, \ldots$ is correct.

Of course, Gold's identification criterion above is just $\mathbf{TxtFex}_1$-identification. It is well known [82] that equivalent grammars (e.g., $p_t, p_{t+1}, p_{t+2}, \ldots$ as above) can be so different from one another that in some cases it is not possible to prove in Zermelo–Frankel set theory [49] that they are equivalent. This suggests that a suitably clever $\mathbf{M}$ might be able to $\mathbf{TxtFex}_{n+1}$-identify a larger class of languages than any machine, however clever, could $\mathbf{TxtFex}_n$-identify. Unfortunately, it was already known [12] that, at least in the case where the data are informant instead of text, one gets no more learning power with $(n+1)$ correct programs in the limit than with n. Surprisingly, then, the main theorem of the present paper (Theorem 3.3 in section 3 below) implies that, nonetheless, for learning from *text*, larger classes of languages *can* be learned with up to $(n+1)$ correct programs in the limit than with up to n.

Theorem 3.3 suggests, then, the *possibility* that evolutionary pressure for increased learning power may have resulted in human language learning strategies that

[2] N.B.: It is not required that $\mathbf{M}$ signal when it has reached its final conjecture—in general it doesn't know when and if it has.

involve convergence to *vacillating* between $n > 1$ correct grammars in the limit. This is examined more critically in section 7 below. Regarding, though, the size of n, we note that at least one of n distinct grammars would have to be of size proportional to the size of n (i.e., to $\log n$); hence, for extraordinarily *large* n, at least one of n distinct grammars would be too large to fit in our heads—unless, as seems unlikely, human storage mechanisms admit infinite regress. Osherson and Weinstein [71] introduced the case where the number of final grammars is finite but *unbounded*, and independently [28, 71] (see also [72]) introduced the case where the number of final grammars is infinite (**TxtBc**-identification). We briefly introduced the case, discussed above, of up to n final grammars in [15].

The proof of Theorem 3.3 employs an $(n+1)$-ary self-reference argument [18], and an informal thesis is presented and discussed after the statement of Theorem 3.3 that self-referential examples witnessing an existence theorem portend natural examples witnessing that theorem.

Case and Lynes [28] considered, among other things, the learning of grammars for languages where a *single* final grammar is allowed to have a bounded number of mistakes (*anomalies*). The mistakes are about which objects are (and which are not) in the corresponding language. In [30, 31, 15] there are discussion, motivation, and interpretation of results about inferring anomalous programs for functions. The results in [30, 31, 15] and in this paper show that allowing anomalies increases learning power. Clearly, anomalous programs are tolerable provided the number of anomalies is *small*. Hence, it is plausible that people have evolved language learning strategies that exploit the greater learning power achieved by converging to slightly incorrect grammars. Theorem 3.3 says, more generally than indicated above, that for each $n > 0$ some classes of languages can be algorithmically learned (in the limit) by converging to *up to* $n+1$ different, exactly correct grammars; *but* these classes can*not* be learned by converging (in the limit) to up to n different grammars, where the up to n grammars are each allowed to have a finite number of anomalies

Corollary 3.7 below specifies a two-dimensional hierarchy involving $\mathbf{TxtFex}_b^a$-identification: learning up to b final grammars each with up to a mistakes.

Theorem 3.11 implies that, in passing from learning finitely many anomalous grammars in the limit to learning infinitely many, one can eliminate half of the anomalies, and that's optimal! Intuitively, since (with positive data only) one is missing approximately half of the information, one can eliminate half of the anomalies only.

If L is a nonempty language, then *some* texts for L are *non*computable sequences, but in a completely computable universe, no parents can generate a noncomputable sequence of data for their children. Hence, it is interesting to consider $\mathbf{RecTxtFex}_b^a$-*identification*, which is similar to $\mathbf{TxtFex}_b^a$-identification except that success is required only on *computable* presentations of positive data, on all *recursive* texts. It might be expected that a suitably clever machine $\mathbf{M}$ might be able to exploit the recursiveness of texts to learn larger classes of languages than any machine required to succeed on arbitrary texts, but Corollary 3.1 below implies that this is not the case (generalizing the $b = 1$ case essentially from [96, 3]). We say, then, that *the restriction to recursive texts is circumvented*.

Angluin, in her seminal paper [1], presents a severe constraint on $\mathbf{TxtFex}_1$-identification of classes of languages: the *subset principle*. Basically, she shows that if $\mathbf{M}$ $\mathbf{TxtFex}_1$-identifies a class of languages $\mathcal{L}$, then for each $L \in \mathcal{L}$ there is a *finite* set D (called a *tell tale*) contained in L such that D is not contained in any proper sublanguage of L in $\mathcal{L}$. Intuitively, this necessary condition prevents overgeneralization in

learning from positive data [1, 8]. Theorem 4.4 below generalizes the subset principle to the criteria of success $\mathbf{TxtFex}^a_b$-identification and $\mathbf{TxtBc}^a$-identification, where the a in $\mathbf{TxtBc}^a$-identification allows each of the infinitely many final grammars converged to have up to a anomalies. Theorem 4.4 is also used to prove Theorem 3.11 below.

A child learning a language may or may not be sensitive to *the order or timing of presentation of positive data*. For $\mathbf{TxtFex}^a_b$-identification (and variants thereof) we mathematically consider several kinds of *in*sensitivity of a machine $\mathbf{M}$ to data order:

1. *set-driven:* $\mathbf{M}$'s output at any point depends only on the *set* of positive data it's seen up to that point (not on the sequence in which it was presented).
2. *partly set-driven:* $\mathbf{M}$'s output at any point depends only on the *set* of positive data it's seen up to that point *and on the length* of the sequence in which it was presented.
3. *b-ary order independent:* for languages L on which for some text $\mathbf{M}$ converges to a finite set of final grammars, $\mathbf{M}$ converges to the *same* set (of cardinality $\leq b$) of final grammars for each text for L.
4. *weakly b-ary order independent:* for languages L on which for some text $\mathbf{M}$ converges to a finite set of final grammars, there is a finite set of grammars D (of cardinality $\leq b$) such that $\mathbf{M}$ converges to a subset of this D for each text for L.

In section 5 below, we prove several theorems, each witnessing that, for suitably clever $\mathbf{M}$'s *simultaneously* exhibiting some insensitivities as above and circumventing the restriction to recursive texts, there is *no* loss of learning power (with respect to $\mathbf{TxtFex}^a_b$-identification or the variants thereof). For example, Theorem 5.5 implies that the power of $\mathbf{TxtFex}^a_b$-identification is unaffected by the restriction to $\mathbf{M}$'s which are simultaneously partly set-driven and weakly b-ary order independent and which circumvent the restriction to recursive texts. Theorem 5.5 is the hardest theorem herein to prove, and the other theorems in section 5 are proved by modifications and/or simplifications of the proof of Theorem 5.5. Some of the theorems in section 5 generalize predecessors for $\mathbf{TxtFex}^0_1$-identification [3, 93, 90, 38, 70, 39] but are much harder to prove. Some of the theorems in section 5 are applied in the present paper and in other papers.

In section 7 we discuss briefly computable universe hypotheses, present some critical discussion as promised above, and sketch some areas for future investigation.

2. Preliminaries. We now proceed more formally.

$\mathbf{N}$ denotes the set of *natural numbers*, $\{0, 1, 2, \ldots\}$.

φ denotes a fixed *acceptable* programming system for the partial computable functions: $\mathbf{N} \to \mathbf{N}$ [81, 65, 79, 80, 83]. φ_p denotes the partial computable function computed by the program (with code number) p in the φ-system.[3] Thanks to the device of Gödel or code numbering [82] we can treat languages over any finite alphabet as subsets of $\mathbf{N}$. $W_p \stackrel{\text{def}}{=}$ the domain of φ_p, the r.e. language ($\subseteq \mathbf{N}$) recognized (or enumerated) by program (grammar) p in the φ-system [82].

DEFINITION 2.1. *A* language learning function *is a computable mapping from finite sequences, of natural numbers and #'s, into* (*Gödel numbers of*) *programs* (*grammars*) *in the* φ*-system.*

[3] The *acceptable* systems are those universal programming systems such as Turing machines, C, and Lisp into which one can compile from any programming system. We characterized them as those universal systems for the partial computable functions in which one can implement any control structure [83].

$\mathcal{E}$ denotes the class of all r.e. languages ($\subseteq \mathbf{N}$).

DEFINITION 2.2. *A* text for *a language L is a mapping T from* $\mathbf{N}$ *into* $(\mathbf{N} \cup \{\#\})$ *such that L is the set of natural numbers in the range of T. T is said to be* for $L \Leftrightarrow T$ *is a text for L. The* content of *a sequence, of natural numbers and #'s, is the set of natural numbers in its range;* content$(\cdot)$ *denotes the content of its argument.*

Intuitively, one can think of a text for a language as an enumeration of the objects in the language with the #'s representing pauses in the listing of such objects. For example, the only text for the empty language is just an infinite sequence of #'s.

Intuitively, if $\mathbf{F}$ is a learning function and σ is a finite initial segment of a text for a language L, then $\mathbf{F}(\sigma)$ represents $\mathbf{F}$'s conjecture as to a grammar for L based on the data about L in σ.

Variables σ and τ (with or without decorations[4]) range over finite initial segments of texts T. $\|\sigma\|$ denotes the length of σ. $\sigma \diamond \sigma'$ denotes the sequence formed by adding σ' to the end of σ. Hence, if $\tau = \sigma \diamond \sigma'$, then, $\forall\, x \in \mathbf{N}$,

$$\tau(x) = \begin{cases} \sigma(x) & \text{if} \quad x < \|\sigma\|; \\ \sigma'(x - \|\sigma\|) & \text{if} \quad \|\sigma\| \leq x < \|\sigma\| + \|\sigma'\|; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Furthermore, $\sigma \diamond x$, where $x \in (\mathbf{N} \cup \{\#\})$, denotes $\sigma \diamond \sigma'$, where $\sigma' = \{(0, x)\}$.

card(D) denotes the cardinality of D. $\mathbf{I}^+$ denotes the set of positive integers. We take a and c to range over $(\mathbf{N} \cup \{*\})$ and b and d to range over $(\mathbf{I}^+ \cup \{*\})$. Intuitively, $*$ denotes the unbounded but finite. For example, "card$(D) \leq *$" means that D is finite. We adopt the convention that $(\forall i \in N)[i < * < \infty]$. $\mathbf{\Delta}$ is the symmetric difference operator for sets/languages. $L_1 =^a L_2 \Leftrightarrow \text{card}(L_1 \mathbf{\Delta} L_2) \leq a$. ("$=^0$" denotes, then, ordinary set equality.) $L_1 \neq^a L_2$ means that it is *not* the case that $L_1 =^a L_2$.

"$\subset$" denotes "is a *proper* subset of," and "$\supset$" denotes "is a *proper* superset of." Set theoretically, as in [47], we treat sequences as functions, and, in general, functions, finite, partial, or total, as *single-valued sets of ordered pairs.*[5] Hence, we can and do meaningfully compare them with "$\subseteq$," "$\subset$," "$\supseteq$," and "$\supset$." It *follows*, for example, that, if T is a text, "$\tau \subset T$" means that "the finite sequence τ is an initial segment of the infinite sequence T."

The quantifier "$\exists^\infty \tau$" means "there exists infinitely many τ."

We use "|" to mean "such that."

DEFINITION 2.3. *Suppose* $\mathbf{F}$ *is a learning function and T is a text. We say* $\mathbf{F}(T)$ converges (*written:* $\mathbf{F}(T)\Downarrow$) $\Leftrightarrow \{\mathbf{F}(\tau) \mid \tau \subset T\}$ *is finite. If* $\mathbf{F}(T)\Downarrow$, *then* $\mathbf{F}(T)$ *is defined* $= \{p \mid (\exists^\infty \tau \subset T)[\mathbf{F}(\tau) = p]\}$; *otherwise,* $\mathbf{F}(T)$ *is undefined.*

DEFINITION 2.4. *A language learning function,* $\mathbf{F}$, $\mathbf{TxtFex}_b^a$-identifies *an r.e. language $L \Leftrightarrow (\forall$ texts T for $L)[\mathbf{F}(T)\Downarrow =$ a set of cardinality $\leq b$ and $(\forall p \in \mathbf{F}(T))[W_p =^a L]]$.*

In $\mathbf{TxtFex}_b^a$-identification the b is a "bound" on the number of final grammars and the a a "bound" on the number of anomalies allowed in these final grammars. As above, a "bound" of $*$ just means *un*bounded but *finite.*

DEFINITION 2.5. $\mathbf{TxtFex}_b^a$ *denotes the class of all* classes $\mathcal{L}$ *of languages such that some learning function* $\mathbf{TxtFex}_b^a$*-identifies each language in* $\mathcal{L}$.

[4]Decorations are subscripts, superscripts, primes, and the like.

[5]Hence, the sequence of numbers $w_0, w_1, w_2, \ldots$ is identified with function f such that, for each $i \in \mathbf{N}$, $f(i) = w_i$, and this f is also identified with its graph $\{(i, f(i)) \mid i \in \mathbf{N}\}$.

Intuitively, $\mathcal{L} \in \mathbf{TxtFex}_b^a \Leftrightarrow$ there is an algorithm $\mathbf{p}$, computing a learning function $\mathbf{F}$, such that, if $\mathbf{p}$ is given any listing T of any language $L \in \mathcal{L}$, it outputs a sequence of grammars *converging* in a nonempty *set*, $\mathbf{F}(T)$, of no more than b grammars, and each of these grammars makes no more than a mistakes in generating L, i.e., if $\mathbf{p}$ is given any listing of any language $L \in \mathcal{L}$, it outputs a sequence of grammars, and, past some point in this sequence, each grammar seen (over and over) is from a set of no more than b grammars and each of these "final" grammars makes no more than a mistakes in generating L.

$\mathbf{TxtFex}_1^0$-identification is equivalent to Gold's [46] seminal notion of *identification*, also referred to as **TXTEX**-identification in [28] and (indirectly) as **INT** in [72, 71, 70]. $\mathbf{TxtFex}_1^a$-identification is just $\mathbf{TXTEX}^a$-identification from [28]. For $n > 0$, $\mathbf{TxtFex}_n^0$-identification is just our notion of $\mathbf{TXTFEX}_n$-identification from [15]. Osherson and Weinstein [71] were the first to define $\mathbf{TxtFex}_*^0$ and $\mathbf{TxtFex}_*^*$; they called them **BEXT** and **BFEXT**, respectively.

It is common in the literature to use $\mathbf{TXTEX}_b^a$ to mean the special case of $\mathbf{TXTEX}^a$ where the total number of changes of output (or *mind changes*) is bounded above by b. *N.B.: The b in $\mathbf{TxtFex}_b^a$ has a totally different meaning from the b in $\mathbf{TXTEX}_b^a$; the former is a bound on the number of different programs an associated machine eventually vacillates between in the limit*; *the latter is a bound on mind changes for convergence to a single final program.*

DEFINITION 2.6. *A text T is* recursive $\Leftrightarrow T$, *as a function:* $\mathbf{N} \to (\mathbf{N} \cup \{\#\})$, *is computable.*[6]

Learning power under $\mathbf{TxtFex}_b^a$-identification might be affected if one requires success only on all *recursive* texts for a language. This is interesting since, for example, if the universe is *completely* algorithmic, then all *real* language texts generated by parents for their children *are* recursive![7]

DEFINITION 2.7. *A language learning function*, $\mathbf{F}$, $\mathbf{RecTxtFex}_b^a$-identifies *an r.e. language $L \Leftrightarrow$* ($\forall$ recursive *texts T for L*)[$\mathbf{F}(T)\Downarrow =$ *a set of cardinality* $\leq b$ *and* $(\forall p \in \mathbf{F}(T))[W_p =^a L]$].

DEFINITION 2.8. $\mathbf{RecTxtFex}_b^a$ *denotes the class of all* classes $\mathcal{L}$ *of languages such that some learning function $\mathbf{RecTxtFex}_b^a$-identifies each language in $\mathcal{L}$.*

It is interesting to consider what happens to learning power if the final programs/grammars for $\mathbf{TxtFex}_b^a$-identification are required to be "nearly" minimal size and, hence, even more likely to fit in one's head.

Let mingrammar(L) denote $\min(\{\, p \mid W_p = L \,\})$.

DEFINITION 2.9. $\mathbf{F}$ $\mathbf{TxtMfex}_b^a$-identifies *a class of languages $\mathcal{L} \Leftrightarrow$* ($\exists$ *recursive h*) $(\forall L \in \mathcal{L})$[$\mathbf{F}$ $\mathbf{TxtFex}_b^a$-identifies $L \wedge (\forall T$ *for* $L)(\forall p \in \mathbf{F}(T))[p \leq h(\text{mingrammar}(L))]$].

h in Definition 2.9 plays the role of a computable amount by which the final programs can be larger than minimal size. This size restriction of course does not hold in general, and, for $\mathbf{TxtMfex}_1^0$-identification, it is not as severe as requiring that the final program be strictly minimal size. Mathematically, $\mathbf{TxtMfex}_b^a$-identification is well behaved, e.g., it turns out not to depend on the choice of acceptable system; it also does not depend on the choice of Blum program size measure [11] (by his recursive-relatedness result in [11]). The lack of dependence on the choice of acceptable system is in contrast with the variant of $\mathbf{TxtMfex}_1^0$-identification in which we require h to be the identity function (see [23]). The study of learning nearly minimal

[6] The r.e. languages are characterized as those which are the content of some recursive text [82].
[7] See further discussion in section 3 below.

size programs began with [35] in the context of learning programs for functions (see also [52, 19, 20, 37]).

DEFINITION 2.10. $\mathbf{TxtMfex}_b^a = \{\mathcal{L} \mid (\exists \mathbf{F})[\mathbf{F}\ \mathbf{TxtMfex}_b^a\text{-}identifies\ \mathcal{L}]\}$.

Similarly, we may define $\mathbf{RecTxtMfex}_b^a$-*identification* and $\mathbf{RecTxtMfex}_b^a$ as $\mathbf{TxtMfex}_b^a$-identification and $\mathbf{TxtMfex}_b^a$, respectively, restricted to recursive texts (see Definitions 2.7 and 2.8 above).

Next, for mathematical completeness and interest, we introduce the cases of success criteria for which the number of final grammars is possibly infinite, not necessarily finite as it is for $\mathbf{TxtFex}_b^a$-identification. Definitions 2.11 and 2.12 are from [28]. The $a \in \{0, *\}$ cases were independently introduced in [71, 72].

The quantifier "$\forall^\infty k$" means "for all but finitely many $k \in \mathbf{N}$."

DEFINITION 2.11. $\mathbf{F}$ $\mathbf{TxtBc}^a$-identifies $L \Leftrightarrow (\forall\ texts\ T\ for\ L)(\forall^\infty k)[W_{\mathbf{F}(T[k])} =^a L]$.

DEFINITION 2.12. $\mathbf{TxtBc}^a$ *denotes the class of all classes* $\mathcal{L}$ *of languages such that some learning function* $\mathbf{TxtBc}^a$*-identifies each language in* $\mathcal{L}$.

$\emptyset$ denotes the empty set of natural numbers.

Fix *canonical indexings* of the *finite* sets of natural numbers and of the finite initial segments of texts each one to one onto $\mathbf{N}$ [82, 65].[8] In the following, finite sets and segments are sometimes *identified* with their corresponding canonical indices. Hence, a reference to a *least* finite set or segment really refers to a finite set or segment with least canonical index. Also, *when we compare finite sets or segments by* $<, \leq, \cdots$ *we are comparing their corresponding canonical indices.*

$\langle \cdot, \cdot \rangle$ denotes a fixed *pairing function* [82], a computable, surjective, and injective mapping from $\mathbf{N} \times \mathbf{N}$ into $\mathbf{N}$.

For $A \subseteq \mathbf{N}$, $\overline{A}$ denotes $(\mathbf{N} - A)$, the *complement* of A.

We let $\mathbf{F}$ (with or without decorations) range over learning functions.

3. Results on vacillation in learning. This section presents our main results regarding the vacillatory learning criteria of the present paper.

In this section we defer proofs of three results until we have the benefit of some of the concepts and results from sections 4 and 5 below. Section 6 contains the three deferred proofs.

The definition of $\mathbf{TxtFex}_b^a$-identification (Definition 2.5 above) requires success for *each* order of data presentation. For each nonempty *r.e.* language, there are continuum many such orders (texts) [47] yet only countably many *recursive* ones (since there are only countably many Turing machine programs for computing the recursive texts [82]). In a completely computable universe (which ours might be), there are really only *recursive* texts available to be presented to learning machines. Of course the universe *may* be such that, while all the language learners are computable, there are some noncomputable phenomena too. As noted in [70], since the utterances of children's caretakers depend heavily on external environmental events, such influences might introduce a random component into naturally occurring texts. It is, then, interesting and important to compare learning power where success is required on *all* texts with the cases where it is required only on all *recursive* texts.

Wiehagen [96] essentially notes that $\mathbf{RecTxtFex}_1^0 = \mathbf{TxtFex}_1^0$ (a related result was first proved in [3]), and [28] essentially observes that $(\forall a)[\mathbf{RecTxtFex}_1^a =$

[8] The canonical index of a finite set or segment is, then, a numerical code of it.

$\mathbf{TxtFex}_1^a$]. We have, more generally, the following corollary.[9]

COROLLARY 3.1.

1. $(\forall a, b)[\mathbf{RecTxtFex}_b^a = \mathbf{TxtFex}_b^a]$.
2. $(\forall a, b)[\mathbf{RecTxtMfex}_b^a = \mathbf{TxtMfex}_b^a]$.

Hence, for all the vacillatory learning criteria of the present paper, it makes no difference in learning power whether or not we restrict the texts to be recursive! By contrast, for $\mathbf{TxtBc}^a$, learning procedures *can* exploit the assumption that they are receiving recursive texts; for $\mathbf{TxtBc}^a$, the restriction to recursive text *does* make a difference in learning power [28, 36].

The topic of learning nearly minimal size programs/grammars is treated in greater depth in [23]. Herein we present results about such criteria only in the contexts of recursive text (Corollary 3.1) and of restrictions on learning functions (section 5 below). There is a small amount of additional discussion below in section 7.

$\mathbf{RecTxtFex}_1^0 = \mathbf{TxtFex}_1^0$ entails that, if a given learning function $\mathbf{F}$ $\mathbf{RecTxtFex}_1^0$-identifies some class, *some* learning function $\mathbf{F}'$ will $\mathbf{TxtFex}_1^0$-identify it. However, by the following proposition, in some cases we can*not* have $\mathbf{F}' = \mathbf{F}$.

PROPOSITION 3.2. *There is a learning function which* $\mathbf{RecTxtFex}_1^0$*-identifies a language which it fails to* $\mathbf{TxtFex}_1^0$*-identify.*

Proof. Let $K = \{\, p \mid p \in W_p \,\}$, a well-known r.e., nonrecursive set [82]. Suppose k is a recursive function with range K [82]. Let $K_s = \{\, k(s') \mid s' < s \,\}$. C_A denotes the *characteristic function of* $A \subseteq \mathbf{N}$, the function 1 on A and 0 off A. Clearly, $(\forall x)[\lim_{s\to\infty} C_{K_s}(x) = C_K(x)]$. Let S_τ = the set of all $x \in$ the domain of τ such that $C_{K_{\|\tau\|}}$ agrees with τ on all inputs $\leq x$. Let

$$\operatorname{agree}(\tau) = \begin{cases} 0 & \text{if} \quad S_\tau = \emptyset; \\ 1 + \max(S_\tau) & \text{otherwise.} \end{cases}$$

It is easy to see that

$$[T = C_K \Rightarrow \lim_{\tau \subset T} \operatorname{agree}(\tau) = \infty]$$

and that

$$[T \neq C_K \Rightarrow \lim_{\tau \subset T} \operatorname{agree}(\tau) < \infty].$$

Let p_0 be a program such that $W_{p_0} = \{\, 0, 1 \,\}$. Let

$$\tau^- = \begin{cases} \emptyset & \text{if} \quad \tau = \emptyset; \\ \tau' & \text{if} \quad \tau = \tau' \diamond x. \end{cases} \tag{3.1}$$

Let

$$\mathbf{F}(\tau) = \begin{cases} \operatorname{agree}(\tau) & \text{if} \quad \operatorname{agree}(\tau) > \operatorname{agree}(\tau^-); \\ p_0 & \text{otherwise.} \end{cases}$$

Let $L = \{\, 0, 1 \,\}$. Clearly, C_K is a nonrecursive text T for L such that $\mathbf{F}(T)\Downarrow$, yet $\mathbf{F}$ on any recursive text for L converges to p_0, a program for L. □

Next is our main theorem. It says that, for each $n > 0$, some classes of languages can be algorithmically learned (in the limit) by converging to *up to* $n + 1$ different,

[9] The proof of Corollary 3.1 is deferred to section 6 since it employs Theorems 5.5 and 5.8 from section 5.

exactly correct grammars; *however,* these classes can*not* be learned by converging (in the limit) to up to n different grammars, where the up to n grammars are each allowed to have a finite number of anomalies! Allowing one more grammar in the limit makes a big difference in learning power.

Hence, it is *possible* that, for some $n > 0$, people have evolved language learning strategies that exploit the greater learning power achieved by converging in the limit to up to $n+1$ rather than to up to n grammars (see a critical discussion in section 7 below).

THEOREM 3.3. *Suppose* $n > 0$. *Let* $\mathcal{L}_{n+1}$ *equal*

$$\{L \mid L \text{ is } \infty \wedge (\exists e_0, \ldots, e_n)[W_{e_0} = \cdots = W_{e_n} = L \wedge (\forall^\infty \langle x, y\rangle \in L)[y \in \{e_0, \ldots, e_n\}]]\}.$$

Then $\mathcal{L}_{n+1} \in (\mathbf{TxtFex}^0_{n+1} - \mathbf{TxtFex}^*_n)$.

The detailed proof of Theorem 3.3 is deferred to section 6 since it depends, in part, on Definitions 5.1 and 5.4 and Theorem 5.6 in section 5.

The reader may note that the languages in the class $\mathcal{L}_{n+1}$ from Theorem 3.3 have an intriguing self-referential character. It is useful to discuss this feature a bit in the interest of anticipating and answering a possible objection to the use of self-reference in witnessing the separation result of Theorem 3.3.

In the proof of Theorem 3.3, to handle the self-referential character of $\mathcal{L}_{n+1}$, we employ the $(n+1)$-ary recursion theorem, a folk theorem generalizing the Kleene recursion theorem [82, p. 214] and the Smullyan double recursion theorem [87]; it is also a consequence of our operator recursion theorem [14], an infinitary analogue of the finitary recursion theorems.

Intuitively, the $(n+1)$-ary recursion theorem provides a means for transforming any sequence of $n+1$ programs $p_0, \ldots, p_n$ into a corresponding sequence of programs $e(p_0), \ldots, e(p_n)$ such that each $e(p_i)$ first creates quiescent copies of $e(p_0), \ldots, e(p_n)$ (including a self-copy, a copy of $e(p_i)$ itself), and then each $e(p_i)$ runs p_i on the quiescent copies of $e(p_0), \ldots, e(p_n)$ any together with any externally given input. Each $e(p_i)$, in effect, has complete (low level) knowledge of $e(p_0), \ldots, e(p_n)$ (including self-knowledge, knowledge of $e(p_i)$ itself), and p_i represents how $e(p_i)$ *uses* its self-knowledge, its knowledge of the other $e(p_j)$'s, and its knowledge of the external world. Infinite regress is not required since each $e(p_i)$ creates the copies of $e(p_0), \ldots, e(p_n)$ *externally* to itself. One mechanism to achieve this creation is a generalization of the self-replication trick isomorphic to that employed by single-celled organisms [14]. Another is for the programs $e(p_0), \ldots, e(p_n)$ to look in a common mirror to see which programs they are. Reference [18] provides a tutorial on thinking about and applying recursion theorems.[10] Herein, our application of the $(n+1)$-ary recursion theorem (to prove Theorem 3.3) will be informal and the sequence $p_0, \ldots, p_n$ will be implicit.

Now for the possible objection: On the one hand, we argue above that Theorem 3.3 suggests a possibility regarding human language learning; on the other hand, we prove it by self/other reference, and it is common to regard self-referential examples as *un*natural. For example, Gödel proved his famous incompleteness theorem by a self-reference argument [45, 62], and his self-referential sentence providing an unprovable truth of, for example, *First Order Peano Arithmetic* (**FOPA**) is not natural—no number or combinatorial theorist would care whether it was true or false.

We answer this objection about self-referential proofs of existence theorems with the following.

[10] See [78] for discussion and applications of recursion theorems in severely resource-limited contexts.

INFORMAL THESIS 1. *If a self-referential example witnesses the existence of a phenomenon, there are* natural *examples witnessing same!*

For this informal thesis we present a brief plausibility argument and one piece of empirical evidence. Plausibility: Self-reference arguments lay bare *an* underlying *simplest* reason for the theorems they prove [82, 18]; if a theorem is true for such a simple reason, the "space" of reasons for its truth may be broad enough to admit natural examples. Empirical: Although Gödel proved his famous first incompleteness theorem by a self-reference argument, many years afterwards, Paris and Harrington [75] and later Friedman [84, 85] found quite natural examples of combinatorial truths of first order arithmetic not provable in **FOPA**.[11] In fairness, regarding the above informal thesis, we note, for example, that the Blum speed-up theorem [10] was originally proved by a self-reference argument,[12] but natural witnesses to even exponential speed-up have not (yet) been found. However, even the self-reference proofs of this result are fairly complicated; hence, one might expect that natural examples are especially hard to find.

For some theoretical work instigated by Barzdin and dealing, in part, with eliminating dependence on self-referential examples, see Fulk's work on robust function learning in [40].

COROLLARY 3.4. $(\forall a)[\mathbf{TxtFex}^a_1 \subset \mathbf{TxtFex}^a_2 \subset \cdots \subset \mathbf{TxtFex}^a_*]$.

COROLLARY 3.5 (Osherson and Weinstein [71]). $\mathbf{TxtFex}^0_1 \subset \mathbf{TxtFex}^0_*$.

We announced in [15] that we could prove $\mathbf{TxtFex}^0_1 \subset \mathbf{TxtFex}^0_2$ by analyzing Osherson and Weinstein's proof of the immediately preceding corollary. Under our direction Karen Ehrlich generalized the combinatorics of this proof to get $\mathbf{TxtFex}^0_2 \subset \mathbf{TxtFex}^0_3$. The combinatorics for this approach to the general case are unpleasant. Reference [70] contains a recursion theorem proof of the immediately preceding corollary based on the proof in [71], but the same combinatorial difficulties occur in attempting to generalize this proof. We sought a combinatorially cleaner self-reference proof. A later conversation about this with Royer led to Royer and Kurtz supplying us with essentially the self-referential sets we use in Theorem 3.3 above. We believe their self-referential examples are somewhat simpler than those we had been working with. They also supplied some of the crucial combinatorics for the diagonal argument that goes with a special case.

It is interesting to note that if one modifies the definition of $\mathbf{TxtFex}^a_n$-identification to require that the learning function must converge to *exactly* n grammars, then the hierarchy of Corollary 3.4 above collapses.[13]

If we restrict our attention to languages which are the (pairing function coded) graphs of *total functions*, then it is essentially shown (the $a = 0$ case in [12] and the $a > 0$ cases in [31]) that the hierarchy again collapses. Hence, in the case of "scientific inference," i.e., the case of learning programs for computable functions, there is no power in vacillation.[14]

Therefore, Corollary 3.4 is very sensitive to minor perturbations. We should men-

[11] See [78] for an example from complexity theory.

[12] See also Young's version in [98] and our operator recursion theorem variant in [86].

[13] Just output every nth grammar.

[14] For computable functions f, one can think of input x as coding a scientific experiment and the output $f(x)$ as coding the corresponding experimental result. In this way results about learning programs for functions can be interpreted as results about finding predictive explanations for phenomena—as results about scientific induction. For more on this see [3, 31, 22, 7, 21, 56]. Regarding the names of the learning criteria studied in the present paper, originally [31] "**Ex**" stood for "explanatory," "**Fex**" stood for "finitely explanatory," and **Bc** for "behaviorally correct."

tion, however, that there *are* some interesting effects on learning power for vacillatory function learning wrought by bounding *suitably sensitive* measures of the computational complexity of the learning functions themselves [24] and by the introduction of noisy input data [25].

The next proposition provides a dual to Theorem 3.3. There are classes which can be learned with one program in the limit and with up to $m+1$ anomalies in that program which can*not* be learned with finitely many programs in the limit, but with each having no more than m anomalies.

PROPOSITION 3.6. $(\mathbf{TxtFex}_1^{m+1} - \mathbf{TxtFex}_*^m) \neq \emptyset$.

Proof. We identify total functions f with $\{\, \langle x, f(x)\rangle \mid x \in \mathbf{N} \,\}$. Let $\mathcal{L} = \{\, \text{total } f \mid \varphi_{f(0)} =^{m+1} f \,\}$. Clearly, $\mathcal{L} \in \mathbf{TxtFex}_1^{m+1}$. Also, $\mathcal{L} \in \mathbf{TxtFex}_*^m$ together with Theorems 2.6 and 2.9 from [31] yields a contradiction. □

In [4] it is shown that $\{\, L \mid L =^{m+1} \mathbf{N} \,\}$ also witnesses the separation of Proposition 3.6.

Clearly, from Theorem 3.3 and Proposition 3.6 we have our main corollary.

COROLLARY 3.7. $\mathbf{TxtFex}_b^a \subseteq \mathbf{TxtFex}_d^c \Leftrightarrow [b \leq d \text{ and } a \leq c]$.

In Corollary 3.7, we see that all *and only* the obvious inclusions hold. Hence, allowing more anomalies, final grammars, or both enhances learning power, but anomalies and final grammars cannot in general completely substitute for one another. For example, $\mathbf{TxtFex}_2^0$ is incomparable to $\mathbf{TxtFex}_1^1$. That is, there are classes which can be learned with no mistakes and up to two final grammars which cannot be learned with up to one mistake and one final grammar, and there are other classes which can be learned with up to one mistake and one final grammar which cannot be learned with no mistakes and up to two final grammars.

COROLLARY 3.8 (Case and Lynes [28]). $\mathbf{TxtFex}_1^0 \subset \mathbf{TxtFex}_1^1 \subset \cdots \subset \mathbf{TxtFex}_1^*$.

Osherson and Weinstein [71] independently showed the case of $\mathbf{TxtFex}_1^0 \subset \mathbf{TxtFex}_1^*$ from the previous corollary.

COROLLARY 3.9 (Osherson and Weinstein [71]). $\mathbf{TxtFex}_*^0 \subset \mathbf{TxtFex}_*^*$.

Next we spell out the connections between $\mathbf{TxtFex}_b^a$ and $\mathbf{TxtBc}^{a'}$. Of course, allowing infinitely many grammars in the limit is not so realistic for modeling language learning but, nonetheless, it is mathematically interesting to make the comparisons.

PROPOSITION 3.10. $\mathbf{TxtBc}^0 - \mathbf{TxtFex}_*^* \neq \emptyset$.

Proof. As in the proof of Proposition 3.6, we identify total functions f with $\{\, \langle x, f(x)\rangle \mid x \in \mathbf{N} \,\}$. Let $\mathcal{L} = \{\, \text{total } f \mid (\forall^\infty k)[\varphi_{f(k)} = f] \,\}$. Clearly $\mathcal{L} \in \mathbf{TxtBc}^0$. Also, $\mathcal{L} \in \mathbf{TxtFex}_*^*$ together with Theorems 2.12 and 3.1 from [31] yields a contradiction. □

REMARK 1. *Proposition* 3.10 *still holds even if we restrict* $\mathbf{TxtBc}^0$*-identification to recursive texts.*

The next theorem says that, in passing from learning finitely many anomalous grammars in the limit to learning infinitely many, one can eliminate half of the anomalies, and that's optimal! This contrasts with the function learning case [31], where, by a result of Steel, one can so eliminate *all* of finitely many anomalies. Intuitively, in the present context, since one is missing in the input data the negative information, i.e., since one is missing approximately half the information, one can eliminate only half of the anomalies.

THEOREM 3.11. $\mathbf{TxtFex}_*^m \subseteq \mathbf{TxtBc}^{m'} \Leftrightarrow m \leq 2.m'$; *furthermore*, $\{\, L \mid L =^{2m+1} \mathbf{N} \,\} \in (\mathbf{TxtFex}_1^{2m+1} - \mathbf{TxtBc}^m)$.

In Theorem 3.11 we see that some excluded inclusions are, at first glance, unexpected. Its proof is deferred to section 6 since it depends on Theorem 4.4 in section 4

below.

Clearly, we have the following corollary.

COROLLARY 3.12 (see [28]). *The class of cofinite sets is in*

$$\left(\mathbf{TxtFex}_1^* - \bigcup_{m\in\mathbf{N}} \mathbf{TxtBc}^m\right).$$

We have not yet worked out all the relationships analogous to those in Theorem 3.11 and Corollary 3.12 for the cases in which $\mathbf{TxtBc}^a$-identification is restricted to recursive texts. As noted above in this section, the restriction to recursive texts *does* affect $\mathbf{TxtBc}^a$-identification [28, 36].

4. Topological results. We next present several useful results which can be described as topological. The exact connections to topology (actually, to Baire category theory and Banach–Mazur games [49]) we will not pursue herein, but on that subject the interested reader can consult [67, 70].

DEFINITION 4.1. *Suppose that $\sigma \subseteq \tau \subset T$, with T a text. Then*

$$\mathbf{F}[\sigma,\tau] = \{\, p \mid (\exists \sigma' \supseteq \sigma \mid \sigma' \subseteq \tau)[p = \mathbf{F}(\sigma')] \,\}$$

and

$$\mathbf{F}[\sigma,T] = \{\, p \mid (\exists \sigma' \supseteq \sigma \mid \sigma' \subset T)[p = \mathbf{F}(\sigma')] \,\}.$$

Suppose that σ is a finite initial segment of a text T. Picture $\mathbf{F}$ being fed T one element at a time and imagine watching the successive corresponding output programs. Then, for example, from Definition 4.1 immediately above, $\mathbf{F}[\sigma,T]$ is the set of all these output programs one sees *from* the time $\mathbf{F}$ is fed all of σ.

DEFINITION 4.2. σ in $L \Leftrightarrow \text{content}(\sigma) \subseteq L$.

Just below is a variant of a fundamental lemma from [71] convenient for this paper. An original, not-so-general version of this lemma is from [3] (see also [67, 70]). Variations on its proof will appear in other proofs.

LEMMA 4.3. *Suppose $L \in \mathcal{E}$. Suppose that, for each text T for L, an arbitrary $\sigma_T \subset T$ is chosen. Then, for these choices, let*

$$P = \bigcup_{T \text{ for } L} \mathbf{F}[\sigma_T, T].$$

It follows that

$$(\forall \sigma \text{ in } L)(\exists \tau \supseteq \sigma \mid \tau \text{ in } L)(\forall \tau' \supseteq \tau \mid \tau' \text{ in } L)[\mathbf{F}(\tau') \in P]. \tag{4.1}$$

Proof. Suppose the hypotheses. Suppose for contradiction the negation of (4.1). Hence,

$$(\exists \sigma \text{ in } L)(\forall \tau \supseteq \sigma \mid \tau \text{ in } L)(\exists \tau' \supseteq \tau \mid \tau' \text{ in } L)[\mathbf{F}(\tau') \notin P]. \tag{4.2}$$

Let T be a fixed text for L. We recursively define another text T' for L as follows. Let $\tau_0 = \sigma$ and $\tau_0' = \tau_0 \diamond T(0)$. Suppose (recursively) that τ_n and $\tau_n' \supset \sigma$ are defined and in L. By (4.2) we may take τ_{n+1} to be the least $\supseteq \tau_n'$ such that $[\tau_{n+1} \text{ in } L \wedge \mathbf{F}(\tau_{n+1}) \notin P]$. Let $\tau_{n+1}' = \tau_{n+1} \diamond T(n+1)$. Let $T' = \bigcup_{n\in\mathbf{N}} \tau_n'$. Clearly, T' is a text for L and $T' = \bigcup_{n\in\mathbf{N}} \tau_n$ too, with $\tau_0 \subset \tau_1 \subset \tau_2 \subset \cdots$. By the choice of τ_n's, for each $n \in \mathbf{N}$, $\mathbf{F}(\tau_{n+1}) \notin P$. Therefore, $\mathbf{F}[\sigma_{T'}, T'] \not\subseteq P$, a contradiction. ∎

The $I = \mathbf{Fex}_1^0$ case of the following theorem is from [1]. She calls the finite sets D featured *tell tales*. The theorem witnesses a severe constraint called the *subset principle* on learning from positive data. See [1, 8] regarding the importance of the subset property for avoidance of *overgeneralization* in learning languages *from positive data*. See [54, 95] for discussion regarding the possible connection between this subset principle and a more traditionally linguistically oriented one in [64].

We let $2* \stackrel{\text{def}}{=} *$.

THEOREM 4.4. *Suppose* $I \in \{ \mathbf{Fex}_b^a, \mathbf{Bc}^a \}$ *and* $\mathbf{F}$ $\mathbf{Txt}I$*-identifies* L. *Then*

$$(\exists D \text{ finite } \subseteq L)(\forall L' \subseteq L \mid D \subseteq L' \wedge L' \neq^{2a} L)[\mathbf{F} \text{ does not } \mathbf{Txt}I\text{-identify } L']. \tag{4.3}$$

It would be interesting to have a complete characterization from Theorem 4.4. Some progress was made in [6], where it is essentially shown that, for any *uniformly decidable class of recursive languages* $\mathcal{L}$, a learning function $\mathbf{F}$ witnesses that $\mathcal{L}$ is in $\mathbf{TxtBc}^a \Leftrightarrow$ *each* $L \in \mathcal{L}$ satisfies (4.3) above.[15]

To prove Theorem 4.4 it is useful to have the following combinatorial lemma whose proof is omitted by reason of being straightforward.

LEMMA 4.5. *Suppose* $[A =^a B \wedge B =^a C]$. *Then* $A =^{2a} C$.

Proof. Suppose the hypotheses. For each T for L, choose a suitably large $\sigma_T \subset T$ that

$$(\forall \tau \supseteq \sigma_T \mid \tau \subset T)[W_{\mathbf{F}(\tau)} =^a L].$$

Let

$$P = \bigcup_{T \text{ for } L} F[\sigma_T, T].$$

Then $(\forall p \in P)[W_p =^a L]$. Hence, by Lemma 4.3,

$$(\exists \tau \supseteq \emptyset \mid \tau \text{ in } L)(\forall \tau' \supseteq \tau \mid \tau' \text{ in } L)[\mathbf{F}(\tau') \in P]. \tag{4.4}$$[16]

Let $D = \text{content}(\tau)$, a *finite* subset of L. Suppose $[D \subseteq L' \subseteq L \wedge L' \neq^{2a} L]$. Let T' be a text for L' such that $T' \supset \tau$. Then, since any $\tau' \subset T'$ is in L, we have by (4.4) that

$$(\forall \tau' \supseteq \tau \mid \tau' \subset T')[\mathbf{F}(\tau') \in P].$$

Hence, $(\forall \tau' \supseteq \tau \mid \tau' \subset T')[W_{\mathbf{F}(\tau')} =^a L \neq^{2a} L']$. Therefore, by Lemma 4.5, $(\forall \tau' \supseteq \tau \mid \tau' \subset T')[W_{\mathbf{F}(\tau')} \neq^a L']$. Hence, $\mathbf{F}$ does not $\mathbf{Txt}I$-identify L'. ∎

Clearly, in the proof of Theorem 4.4 there is *no* use of the computability of $\mathbf{F}$. The limitation Theorem 4.4 witnesses on learning from texts is purely topological having nothing to do with algorithmicity. Corollary 4.6 and the nonlearnability half of Theorem 3.11 above, proved from Theorem 4.4, likewise do not depend on algorithmicity.

[15] This complements a related characterization in [1] of the uniformly decidable classes of recursive languages in $\mathbf{TxtFex}_1^0$. Reference [6] also provides a related characterization of the uniformly decidable classes of recursive languages in $\mathbf{TxtFex}_1^*$. References [63, 57] contain characterizations of uniformly decidable classes of recursive languages in important special cases of $\mathbf{TxtFex}_1^0$, and [26] contains characterizations of language learning with noisy texts.

[16] τ is, then, what is suggestively called a *locking sequence* [70].

COROLLARY 4.6 (see [71, 28]). *Suppose that $\mathcal{L}$ contains an infinite language L and all its finite sublanguages. Then $\mathcal{L} \notin \mathbf{TxtBc}^*$. Hence, the class of regular languages $\notin \mathbf{TxtBc}^*$.*

Theorem 4.4 does *not* imply that if a learning function $\mathbf{TxtFex}_1^0$-identifies an infinite language, it must fail to $\mathbf{TxtFex}_1^0$-identify *each* proper sublanguage. In fact we have the following proposition, a variant of which, regarding function learning, appears in [18].

PROPOSITION 4.7. *There is in $\mathbf{TxtFex}_1^0$ an infinite r.e. collection of infinite languages of the form $\{ W_{e_0} \supset W_{e_1} \supset W_{e_2} \supset \cdots \}$.*

Proof. By the operator recursion theorem [14], there is an infinite r.e. sequence of self-other referential programs $e_0, e_1, e_2, \ldots$ such that, for each $i \in \mathbf{N}$,

$$W_{e_i} = \{ e_i, e_{i+1}, e_{i+2}, \ldots \}.$$

We omit the straightforward verification. □

Gold [46] proved Corollary 4.6 with $\mathbf{TxtFex}_1^0$ in place of $\mathbf{TxtBc}^*$ and was clearly concerned that his result meant that only rather puny language classes could be learned from positive data. However, Wiehagen [96] presents a class of r.e. languages in $\mathbf{TxtFex}_1^0$ which contains a finite variant of each r.e. language. Wiehagen's class is obviously quite hefty. Angluin presents examples natural from the perspective of formal language theory that also are in $\mathbf{TxtFex}_1^0$ [1, 2]. All these classes in $\mathbf{TxtFex}_1^0$ (of course) satisfy the subset principle (of Theorem 4.4), and, in particular, they are not closed under finite sublanguages as is the class of regular languages.

Suppose that $\mathcal{N}$ is a class of natural languages learnable from text and which contains some language L and also an infinitely different natural sublanguage L' of L. For example, L' might be the class of imperative sentences of L. Theorem 4.4 above causes no apparent problem since a finite tell-tale D for L need not (and should not) be contained in L'. It may be useful for linguists to try to find such tell-tale D's for natural languages L. Of course such a D shouldn't be contained in, for example, L', the set of imperative sentences of L, but should nonetheless be salient empirically to the learning of L.

The following stability property is useful for studying the criteria $\mathbf{RecTxtFex}_b^a$.

DEFINITION 4.8. *Suppose that L is r.e. Then L recursively b-stabilizes $\mathbf{F} \Leftrightarrow$*

$$(\forall \text{ recursive } T \text{ for } L)(\exists D \mid \operatorname{card}(D) \leq b)[\mathbf{F}(T)\Downarrow = D].$$

The following lemma, which is useful to this paper, combines the topological with the algorithmic. It generalizes predecessors from [3, 38, 70].

LEMMA 4.9. *Suppose that L is r.e. and recursively b-stabilizes $\mathbf{F}$. Then*

$$(\forall \sigma \text{ in } L)(\exists D \mid \operatorname{card}(D) \leq b)(\exists \tau \supseteq \sigma \mid \tau \text{ in } L)(\forall \tau' \supseteq \tau \mid \tau' \text{ in } L)[\mathbf{F}(\tau') \in D]. \tag{4.5}$$

Proof. Suppose the hypothesis on L and, for contradiction, the negation of (4.5). Hence,

$$(\exists \sigma \text{ in } L)(\forall D \mid \operatorname{card}(D) \leq b)(\forall \tau \supseteq \sigma \mid \tau \text{ in } L)(\exists \tau' \supseteq \tau \mid \tau' \text{ in } L)[\mathbf{F}(\tau') \notin D]. \tag{4.6}$$

Let T be a fixed recursive text for L. We recursively define another recursive text T' for L as follows. Let $\tau_0 = \sigma$ and $\tau_0' = \tau_0 \diamond T(0)$. Let $\tau_0'' =$ the *shortest* $\subseteq \tau_0'$ such that $\operatorname{card}(\mathbf{F}[\tau_0'', \tau_0']) \leq b$. (For $b = *$, τ_n'' will $= \emptyset$, for all $n \in \mathbf{N}$.) Let $D^0 = \mathbf{F}[\tau_0'', \tau_0']$. Suppose (recursively) that τ_n, τ_n', and τ_n'' are defined and in L, $\tau_n, \tau_n' \supset \sigma$, $\tau_n'' \subseteq \tau_n'$,

and that $D^n = \mathbf{F}[\tau''_n, \tau'_n]$. By (4.6) we may algorithmically find a $\tau_{n+1} \supseteq \tau'_n$ such that $[\tau_{n+1}$ in $L \ \wedge \ \mathbf{F}(\tau_{n+1}) \notin D^n]$. Let $\tau'_{n+1} = \tau_{n+1} \diamond T(n+1)$. Let $\tau''_{n+1} =$ the *shortest* $\subseteq \tau'_{n+1}$ such that $[\tau''_{n+1} \supseteq \tau''_n \ \wedge \ \text{card}(\mathbf{F}[\tau''_{n+1}, \tau'_{n+1}]) \leq b]$. Let $D^{n+1} = \mathbf{F}[\tau''_{n+1}, \tau'_{n+1}]$. Let $T' = \bigcup_{n \in \mathbf{N}} \tau'_n$. Clearly, T' is a recursive text for L and $T' = \bigcup_{n \in \mathbf{N}} \tau_n$ too, with $\tau_0 \subset \tau_1 \subset \tau_2 \subset \cdots$. By the choice of τ_n's, for each $n \in \mathbf{N}$, $\mathbf{F}(\tau_{n+1}) \notin D^n$. Therefore, $\mathbf{F}(T')\Downarrow$ to a set of cardinality $\leq b$, a contradiction to the hypothesis on L. □

5. Insensitive or restricted learning functions. It is interesting to ask whether or not child language learning exhibits sensitivity to *the order or the timing of presentation of data*. We consider herein some mathematical versions of this question. Several mathematical definitions have been given for various different notions of insensitivity to order, essentially for the case of $\mathbf{TxtFex}^0_1$-identification [3, 93, 90, 38, 70, 39].

We extend these definitions of insensitive or restricted learning functions naturally to the context of the vacillatory learning criteria of the present paper,[17] and we investigate the interesting mathematical questions of whether learning functions with these insensitivities or restrictions thereby lose learning power. Answering many of these questions for the vacillatory criteria is much more difficult than for the $\mathbf{TxtFex}^0_1$ case.[18]

As noted above, we also apply some of our results in this section to help us prove results in this and other papers.

DEFINITION 5.1 (Wexler [93, 70]). $\mathbf{F}$ *is called* set-driven $\Leftrightarrow (\forall \sigma, \tau \mid \text{content}(\sigma) = \text{content}(\tau))[\mathbf{F}(\sigma) = \mathbf{F}(\tau)]$.

Reference [93] essentially notes that *set-driven* learning functions are insensitive to *time* (unlike text learnability). The next defined restriction in effect provides some degree of sensitivity to timing.

DEFINITION 5.2 (Schäfer [90, 70], Fulk [38, 39]). $\mathbf{F}$ *is called* partly set-driven (*synonym* [38, 39]: rearrangement independent) $\Leftrightarrow (\forall \sigma, \tau \mid \|\sigma\| = \|\tau\| \ \wedge \ \text{content}(\sigma) = \text{content}(\tau))[\mathbf{F}(\sigma) = \mathbf{F}(\tau)]$.

Intuitively, $\mathbf{F}$ is set-driven (respectively, partly set-driven) iff, for each σ, $\mathbf{F}(\sigma)$ depends only on the *content* of σ (respectively, depends only on the *length and content* of σ).

First Schäfer [90, 70] and later Fulk [38, 39] independently showed that set-driven learning functions can't $\mathbf{TxtFex}^0_1$-identify some classes of languages that unrestricted learning functions can, but partly set-driven learning functions do not restrict learning power with respect to $\mathbf{TxtFex}^0_1$-identification. Fulk additionally showed that set-driven learning functions can't even $\mathbf{TxtBc}^0$-identify some language classes in $\mathbf{TxtFex}^0_1$. He interprets the difference in power between set-driven and partly set-driven learning functions as witnessing the need for time greater than the size of the content of the input to "think" about the input.

Osherson, Stob, and Weinstein [70] observed that the power of $\mathbf{TxtFex}^0_1$-identification on *infinite* r.e. languages is not limited by set-driveness.

[17] For the so-called order independence notions (Definition 5.4), in the interest of conceptual parsimony, but without loss of generality in theorems, we render them purely syntactically rather than as a mixture of syntactical and semantical (as their precursor notions are in the prior literature). The precursor notions required the final programs/grammars also to be correct, a *semantic* constraint which we eliminate from the definitions.

[18] In many cases it is especially difficult to prove that the *simultaneous* presence of several insensitivities leads to *no* loss of learning power. We had several painful experiences, for example, with subtly incorrect, alternative constructions to the one in the proof of Theorem 5.5.

The following definition presents a convenient term paralleling that from Definition 4.8.

DEFINITION 5.3. *A text T* stabilizes $\mathbf{F} \Leftrightarrow \mathbf{F}(T)\Downarrow$.

While identification of a language L requires identification for *each* order of presentation of (text for) L, the *final* (correct) grammar(s) converged to may be different for different texts. As noted in [70], this would seem to be a source of strength, since for a learning machine's forcing the final grammars to be the same for each text might involve its (algorithmically) recognizing *grammar equivalence*, i.e., recognizing $\{\langle x, y\rangle \mid W_x = W_y\}$, but as is well known [82], this set is *not* algorithmically recognizable (r.e.) (nor is its complement).[19]

Order independent machines are insensitive to which text is used for L in that their final grammars depend only on L, not on the order of presentation. Their grammars along the way can, of course, depend on the text.

If, for some $n > 0$ and for some a, humans $\mathbf{TxtFex}^a_{n+1}$-identify a language L but do not $\mathbf{TxtFex}^a_n$-identify it, it is interesting whether, nonetheless, *some* environments and corresponding texts for L cause them to output fewer final conjectures than $n + 1$. There is a corresponding and ostensibly weaker notion of order independence in which, for each text, the set of final grammars converged to is always *contained in* (but not necessarily equal to) some *finite* set of final grammars.

These order independence notions clearly capture a very different kind of insensitivity to order of data presentation than the set-driven notions above.[20] The formal definition for our order independence notions immediately follows.

DEFINITION 5.4.

1. *We call a learning function*, $\mathbf{F}$, b-ary order independent $\Leftrightarrow$ ($\forall$ L *r.e.* | some *text for* L *stabilizes* $\mathbf{F}$)($\exists D$ *of cardinality* $\leq b$)($\forall$ *texts* T *for* L)[$\mathbf{F}(T)\Downarrow = D$].
2. *We call a learning function*, $\mathbf{F}$, weakly b-ary order independent $\Leftrightarrow$ ($\forall$ L *r.e.* | some *text for* L *stabilizes* $\mathbf{F}$)($\exists D$ *of cardinality* $\leq b$)($\forall$ *texts* T *for* L)[$\mathbf{F}(T)\Downarrow \subseteq D$].

Osherson, Stob, and Weinstein [70], adapting a related result of L. Blum and M. Blum [3], essentially show that order independent learning functions can $\mathbf{TxtFex}^0_1$-identify the same classes of languages that unrestricted learning functions can.

The first theorem of the present section (Theorem 5.5 below) implies that learning power (with respect to $\mathbf{TxtFex}^a_b$-identification) is not decreased by restricting learning functions to be *simultaneously* partly set-driven and weakly b-ary order independent. Furthermore, it implies that one can also simultaneously circumvent the restriction to recursive texts. It generalizes parts of Fulk's kitchen sink theorem [39, Theorem 13, p. 6] and [38, Chapter 5, Theorem 21], which covered the $\mathbf{TxtFex}^0_1$ case only; however, the lift to Theorem 5.5 ostensibly requires a much more difficult proof.[21]

For nontrivially vacillatory criteria, it is open whether (full) order independence can be combined with partly set-driven without loss of learning power.

Theorem 5.5's proof is the most difficult of the present paper. Fortunately, the other proofs of theorems in this section are modifications and/or simplifications of the proof of Theorem 5.5.

---

[19] In fact, more importantly, since this set is Π^0_2-complete [82], it is not even algorithmically recognizable by a limiting [88] or mind-changing procedure (but its complement is).

[20] One can think of them as *global* and the set-driven notions as *local*.

[21] In the present paper we do not consider the restriction to so-called *prudence* [70], a primary concern of [39]. *Prudent learning functions* are those which never conjecture a grammar p without being able to learn W_p. On that subject the interested reader may also wish to consult [51, 55].

THEOREM 5.5. *There is an algorithm for transforming any b and (an algorithm for) a learning function $\mathbf{F}$ into a corresponding (algorithm for a) learning function $\mathbf{F}'$ such that*

1. *$\mathbf{F}'$ is both partly set-driven and weakly b-ary order independent, and*
2. *$(\forall$ r.e. $L)[\mathbf{F}$ $\mathbf{RecTxtFex}_b^a$-identifies $L \Rightarrow \mathbf{F}'$ $\mathbf{TxtFex}_b^a$-identifies $L]$.*

Proof. Suppose that pad is a one-to-one computable function such that $(\forall n, p)$ $[W_{\mathrm{pad}(p,n)} = W_p]$ [65, 83]. Intuitively, $\mathrm{pad}(p,0), \mathrm{pad}(p,1), \mathrm{pad}(p,2), \ldots$ are just padded variants of program p which have the same recognizing behavior as p but which differ from one another syntactically.

Suppose $\mathbf{F}$ and b are given. Define $\mathbf{F}'$ on τ thus. Set $n = \|\tau\|$ and $A = \mathrm{content}(\tau)$.

$(*$ In the definition of $\mathbf{F}'(\tau)$ the only dependence on τ will be on n and A to make sure $\mathbf{F}'$ is partly set-driven. $*)$

Search for the *least* $\langle D^1, \sigma^1 \rangle$ such that[22]

1. $\mathrm{card}(D^1) \leq b$,
2. $\mathrm{content}(\sigma^1) \subseteq A$, and
3. $(\forall \sigma' \supseteq \sigma^1 \mid \sigma' \leq n \ \wedge \ \mathrm{content}(\sigma') \subseteq A)[\mathbf{F}[\sigma^1, \sigma'] \subseteq D^1]$.[23]

$(*$ Clearly, such a $\langle D^1, \sigma^1 \rangle$ will always exist since σ^1 may be chosen big enough not to be contained in any $\sigma' \leq n$. $*)$

$(*$ Suppose τ in L. Clause 3 provides a bounded (by n) approximation to

$$(\forall \sigma' \supseteq \sigma^1 \mid \mathrm{content}(\sigma') \subseteq L)[\mathbf{F}[\sigma^1, \sigma'] \subseteq D^1]. \tag{5.1}$$

(5.1) is a useful stability condition. $*)$

Once $\langle D^1, \sigma^1 \rangle$ is found:

set $i = 1$;
while $[\mathrm{card}(D^i) > 1 \ \wedge$ a *least* $\langle D', \sigma' \rangle \leq n$ is found such that $D' \subset D^1 \ \wedge \ \sigma' \supset \sigma^i \ \wedge \ \mathrm{content}(\sigma') \subseteq A \ \wedge \ (\forall \sigma'' \supseteq \sigma' \mid \sigma'' \leq n \ \wedge \ \mathrm{content}(\sigma'') \subseteq A)[\mathbf{F}[\sigma', \sigma''] \subseteq D']]$[24]
do $(*$ Pump down from D^i and ratchet up from σ^i, preserving apparent stability. $*)$
increment i by 1;
set $\langle D^i, \sigma^i \rangle = \langle D', \sigma' \rangle$
endwhile;
set $\mathbf{F}'(\tau) = \mathrm{pad}(\mathbf{F}(\sigma^i), \langle D^1, \sigma^1 \rangle)$.

Clearly, by construction, $\mathbf{F}'$ is partly set-driven.

Intuitive discussion. Something like the while loop in (the algorithm for) $\mathbf{F}'$ is essential. It is crucial for establishing Claim 3 below. If stopping with a search for $\langle D^1, \sigma^1 \rangle$ sufficed, Theorem 3.3 above could not hold. Nothing like this while loop is needed to handle the cases of $\mathbf{TxtFex}_1^a$. The use of pad is a variant of its use in [38, 39] and serves below in the proof of $\mathbf{F}'$'s weak b-ary order independence in a combinatorially similar role.[25]

---

[22] It is useful to recall here that, from section 2, $\langle \cdot, \cdot \rangle$ is a numerical pairing function *and* that we identify finite sets and initial segments of texts with their corresponding canonical indices (numbers). The word *least*, then, refers to least *numerical* value.

[23] Again, it is useful to recall that, from section 2, we identify finite initial segments of texts with their corresponding canonical indices (numbers). Hence, in the inequality, "$\sigma' \leq n$," we are treating σ as its numerical canonical index.

[24] N.B.: It is useful to recall here that, from section 2 above, "$\subset$" denotes "is a *proper* subset of," and "$\supset$" denotes "is a *proper* superset of."

[25] Intuitively, it *helps* make weak b-ary order independence true by *preventing*, for many r.e. languages L, the presence of some text for L that stabilizes $\mathbf{F}'$.

Here's an intuitive way to think about this construction. Imagine a chimpanzee given an infinite collection of different kinds of sticks, some of which can be joined together to make longer sticks. Each stick points overhead in a particular direction with respect to the vertical. Above the chimp, but out of its sight, is a bunch of bananas it would like to knock down with a suitably large joined-together stick pointing in just the right direction to hit the bananas. However, it can't tell when it has actually reached the bunch of bananas even though it does reach them (so the poor thing never knows when it has succeeded and it never actually gets to eat the bananas). All it can tell is that some time after any choice of a (leaning) tower of sticks is not pointing quite right, one of the sticks will explode, knocking down all of the sticks above it, and it has to try again. The exploding sticks are quite like the injuries in a recursion-theoretic priority argument [88].

The sticks correspond to the σ's, and one should think of them as initial segments of branches in an infinite-branching, upward-pointing tree similar to the finite-branching (rightward pointing) tree in [82, p. 157]. For each input τ to $\mathbf{F}'$, when the while loop finishes, it provides some sequence of successively longer joined together sticks $\sigma^1 \subset \cdots \subset \sigma^m$, with m the final value of i. A larger input to $\mathbf{F}'$, $\tau' \supset \tau$, may result in a different sequence of sticks, $\sigma_1 \subset \cdots \subset \sigma_{m'}$, from the while loop. The stick that exploded is the σ^i with least i such that $\sigma^i \neq \sigma_i$. "Success" for the chimpanzee is described by Claim 1 below. We continue this discussion after the statement of that claim.

CLAIM 1. *If L recursively b-stabilizes $\mathbf{F}$, then, for each text T for L, there is a* maximum *$j \geq 1$ such that the algorithm for $\mathbf{F}'$ above on T* eventually *has stable values for $\langle D^1, \sigma^1\rangle, \ldots, \langle D^j, \sigma^j\rangle$, i.e., values that are the same for (the algorithm for) $\mathbf{F}'$'s calculation of $\mathbf{F}'(\tau)$ for all but finitely many $\tau \subset T$. This j will also be $\leq b$. Furthermore, if there is such a maximum j for* some *text for L, values of this maximum j and associated stable values of $\langle D^1, \sigma^1\rangle, \ldots, \langle D^j, \sigma^j\rangle$ will be* independent *of the choice of text for L.*

Continued discussion. If L recursively b-stabilizes $\mathbf{F}$ and T is a text for L, then this claim does *not* imply that, for all but finitely many $\tau \subset T$, the while loop on input τ stops with the same $\langle D^1, \sigma^1\rangle, \ldots, \langle D^j, \sigma^j\rangle$—only that the while loop stops with $\langle D^1, \sigma^1\rangle, \ldots, \langle D^i, \sigma^i\rangle$, for some $i \geq j$. Success for the chimpanzee discussed above is the stabilization on $\sigma^1 \subset \cdots \subset \sigma^j$, but even after this stability is reached, any sticks returned by the while loop, σ^i, for $i > j$, will "explode" on some longer input to $\mathbf{F}'$.

Suppose L recursively b-stabilizes $\mathbf{F}$ and T is a text for L. Suppose j is as in the previous paragraph. As $\mathbf{F}$ is being fed successively longer initial segments of T, eventually σ^j is reached. We also like to think about the changing σ^i's subsequently found, where $i > j$, as a *flickering flame* above σ^j. Stability implies that, for infinitely many $\tau \subset T$, the flame *may* die down to exactly the level of σ^j itself; however, for all but finitely many $\tau \subset T$, it does not dip below or destroy σ^j.

Proof of Claim 1. Suppose that L recursively b-stabilizes $\mathbf{F}$. Then by Lemma 4.9,

$$(5.2) \qquad (\exists D \mid \operatorname{card}(D) \leq b)(\exists \sigma \supseteq \emptyset \mid \sigma \text{ in } L)(\forall \sigma' \supseteq \sigma \mid \sigma' \text{ in } L)[\mathbf{F}(\sigma') \in D].$$

(The algorithm for) $\mathbf{F}'$ on texts for L will eventually stabilize in its choice of $\langle D^1, \sigma^1\rangle$ to be the *same for each T' for L:* it will stabilize its choice of $\langle D^1, \sigma^1\rangle$ to be the least $\langle D, \sigma\rangle$ satisfying (5.2). This is since, for all but finitely many $\tau \subset T$, $\|\tau\|$ and content(τ) will be big enough to find counterexamples to all the *finitely* many $\langle D', \sigma'\rangle <$ this least $\langle D, \sigma\rangle$ satisfying (5.2). Of course, once a $\langle D', \sigma'\rangle$ is rejected for

being $\langle D^1, \sigma^1\rangle$, it's not picked up again by (the algorithm for) $\mathbf{F}'$ on bigger input since counterexamples don't go away for bigger input. Once the choice of $\langle D^1, \sigma^1\rangle$ has stabilized on a T for L, say, on all sufficiently large $\tau \subset T$; on such suitably large τ, the while loop eventually terminates with a final value for i, say i_τ, which is $\leq b$ since $\mathrm{card}(D^1) \leq b$, and the while loop looks for *proper* subsets of the D^i's. Clearly, as above, on suitably large $\tau \subset T$, there is a maximum $i \leq$ the while loop's i_τ's with $\langle D^1, \sigma^1\rangle, \ldots, \langle D^i, \sigma^i\rangle$ eventually stable, and, also clearly, this maximum i is independent of texts for L. □

CLAIM 2. $\mathbf{F}'$ *is weakly b-ary order independent.*

Proof of Claim 2. Suppose that T for L stabilizes $\mathbf{F}'$. We need to show, then, that $(\exists D \text{ of cardinality} \leq b)[\bigcup_{T' \text{ for } L} \mathbf{F}'(T')\Downarrow \subseteq D]$. Once (the algorithm for) $\mathbf{F}'$ on (successively longer $\tau \subset$) T rejects a candidate for $\langle D^1, \sigma^1\rangle$, it cannot rechoose that candidate later since counterexamples to the stability demanded of $\langle D^1, \sigma^1\rangle$ do not go away. $\mathbf{F}'$ on T outputs programs of the form $\mathrm{pad}(\mathbf{F}(\sigma^i), \langle D^1, \sigma^1\rangle)$, where $\langle D^1, \sigma^1\rangle$ is a candidate for stability at the first level, so to speak. Since, by assumption just above, T *does* stabilize $\mathbf{F}'$ for some *finite* D, $\mathbf{F}'(T)\Downarrow = D$, and then, since pad is one-to-one, the $\langle D^1, \sigma^1\rangle$ argument to it cannot take on infinitely many values as $\mathbf{F}'$ is fed T. Since $\mathbf{F}'$ can't jump back to rejected previous choices of $\langle D^1, \sigma^1\rangle$, (the algorithm for) $\mathbf{F}'$ on T eventually finds a stable value for $\langle D^1, \sigma^1\rangle$. Hence, by a simple restatement of the proof of Claim 1 above, there is a maximum i such that (the algorithm for) $\mathbf{F}'$ on T eventually has stable values for $\langle D^1, \sigma^1\rangle, \ldots, \langle D^i, \sigma^i\rangle$, and the value of i is independent of texts for L. Let imax denote this maximum i. Hence, $(\forall \tau \supseteq \sigma^{\mathrm{imax}} \mid \tau \text{ in } L)[\mathbf{F}(\tau) \in D^{\mathrm{imax}}]$. Therefore,

$$\bigcup_{T' \text{ for } L} \mathbf{F}'(T')\Downarrow \subseteq \mathrm{pad}(D^{\mathrm{imax}}, \langle D^1, \sigma^1\rangle). \tag{5.3}$$

This latter set of programs has cardinality $\leq b$ since D^{imax} does. Therefore, $\mathbf{F}'$ is weakly b-ary order independent. □

N.B.: There is no guarantee that (5.3) is an equality since we may have that, on some T for L, for all but finitely many $\tau \subset T$, and for the corresponding σ^{i_τ}'s from the while loop, the programs $\mathrm{pad}(\mathbf{F}(\sigma^{i_\tau}), \langle D^1, \sigma^1\rangle)$ miss some values in $\mathrm{pad}(D^{\mathrm{imax}}, \langle D^1, \sigma^1\rangle)$.

CLAIM 3. *Suppose L recursively b-stabilizes* $\mathbf{F}$. *Let* imax *be the maximum i from Claim* 1 *(independent of the choice of text for L). Let*

$$D^{Rec} = \bigcup_{\substack{\text{recursive } T \text{ for } L \\ T \supset \sigma^{\mathrm{imax}}}} \mathbf{F}(T).$$

Equivalently,

$$D^{Rec} = \{\, \mathbf{F}(\tau) \mid \tau \text{ in } L \wedge \tau \supseteq \sigma^{\mathrm{imax}} \wedge (\exists \text{ recursive } T \text{ for } L)(\exists^\infty \tau' \subset T)[\mathbf{F}(\tau') = \mathbf{F}(\tau)]\}. \tag{5.4}$$

Then $D^{Rec} = D^{\mathrm{imax}}$.

Proof of Claim 3. Suppose the hypotheses. Clearly, $D^{\mathrm{Rec}} \subseteq D^{\mathrm{imax}}$. It remains to show $D^{\mathrm{imax}} \subseteq D^{\mathrm{Rec}}$. In that interest, suppose $p \in D^{\mathrm{imax}}$. We will show $p \in D^{\mathrm{Rec}}$. By the maximality of imax,

$$\neg(\exists \sigma' \supseteq \sigma^{\mathrm{imax}} \mid \sigma' \text{ in } L)(\forall \sigma'' \supseteq \sigma' \mid \sigma'' \text{ in } L)[\mathbf{F}[\sigma', \sigma''] \subseteq D^{\mathrm{imax}} - \{\, p \,\}].$$

Hence,

$$(\forall \sigma' \supseteq \sigma^{\text{imax}} \mid \sigma' \text{ in } L)(\exists \sigma'' \supseteq \sigma' \mid \sigma'' \text{ in } L)[\mathbf{F}(\sigma'') = p]. \tag{5.5}$$

Let T be a fixed recursive text for L. We recursively define another recursive text T' for L as follows. Let $\tau_0 = \sigma^{\text{imax}}$ and $\tau_0' = \tau_0 \diamond T(0)$. Suppose (recursively) that τ_n and $\tau_n' \supset \sigma^{\text{imax}}$ are defined and in L. By (5.5) we may algorithmically find a $\tau_{n+1} \supseteq \tau_n'$ such that $[\tau_{n+1} \text{ in } L \wedge \mathbf{F}(\tau_{n+1}) = p]$. Let $\tau_{n+1}' = \tau_{n+1} \diamond T(n+1)$. Let $T' = \bigcup_{n \in \mathbf{N}} \tau_n'$. Clearly, T' is a recursive text for L and $T' = \bigcup_{n \in \mathbf{N}} \tau_n$ too, with $\tau_0 \subset \tau_1 \subset \tau_2 \subset \cdots$. By the choice of τ_n's for each $n \in \mathbf{N}$, $\mathbf{F}(\tau_{n+1}) = p$. Hence, T' is a recursive text for L such that $[T' \supset \sigma^{\text{imax}} \wedge \mathbf{F} \text{ on } T' \text{ outputs } p \text{ infinitely often}]$. Therefore, by (5.4), $p \in D^{\text{Rec}}$. □

CLAIM 4. $(\forall$ *r.e.* $L)[\mathbf{F}$ **RecTxtFex**^a_b*-identifies* $L \Rightarrow \mathbf{F}'$ **TxtFex**^a_b*-identifies* $L]$.

Proof of Claim 4. Suppose that L is r.e. and $\mathbf{F}$ **RecTxtFex**^a_b-identifies L. It remains to be shown $\mathbf{F}'$ **TxtFex**^a_b-identifies L. Clearly, L recursively b-stabilizes $\mathbf{F}$. Therefore, by Claim 1, a maximum imax exists with eventually stable values for $\langle D^1, \sigma^1 \rangle, \ldots, \langle D^{\text{imax}}, \sigma^{\text{imax}} \rangle$ independent of texts for L in the operation of (the algorithm for) $\mathbf{F}'$. Clearly, $(\forall p \in D^{\text{Rec}})[W_p =^a L]$. By Claim 3, $D^{\text{Rec}} = D^{\text{imax}}$, so we have $(\forall p \in D^{\text{imax}})[W_p =^a L]$. Hence, $(\forall p \in \text{pad}(D^{\text{imax}}, \langle D^1, \sigma^1 \rangle))[W_p =^a L]$. Therefore, by (5.3), $\mathbf{F}'$ **TxtFex**^a_b-identifies L. □

This ends the proof of Theorem 5.5. □

The next theorem (Theorem 5.6) implies that learning power for *infinite* r.e. languages (with respect to **TxtFex**^a_b-identification) is not decreased by restricting learning functions to be *simultaneously* (completely) set-driven and weakly b-ary order independent. Furthermore, it implies that one can also simultaneously circumvent the restriction to recursive texts.

THEOREM 5.6. *There is an algorithm for transforming any* b *and (an algorithm for) a learning function* $\mathbf{F}$ *into a corresponding (algorithm for a) learning function* $\mathbf{F}'$ *such that*

1. $\mathbf{F}'$ *is both set-driven and weakly* b*-ary order independent, and*
2. $(\forall \infty$ *r.e.* $L)[\mathbf{F}$ **RecTxtFex**^a_b*-identifies* $L \Rightarrow \mathbf{F}'$ **TxtFex**^a_b*-identifies* $L]$.

Proof. Modify (the algorithm for) $\mathbf{F}'$ in the proof above of Theorem 5.5 by setting $n = \text{card}(\text{content}(\tau))$ (instead of setting $n = \|\tau\|$). Since for *infinite* L this n grows, one can apply the rest of the proof of Theorem 5.5 *mutatis mutandis*.[26] □

Royer and Kurtz suggested to us that the use of set-driven learning functions could simplify the proof of at least a special case of Theorem 3.3 and Jun Tarui pointed out to us that weak b-ary order independence would further simplify proving Theorem 3.3. The proof herein of Theorem 3.3 makes use of Theorem 5.6.

We believe it is not possible to replace weak b-ary order independence with b-ary order independence in Theorems 5.5 and 5.6, contrary to our slightly overzealous claims in [16]. However, we have the following result (Theorem 5.7) with Fulk (who is not responsible for the possibly incorrect claims in [16]). Theorem 5.7 implies that learning power (with respect to **TxtFex**^a_b-identification) is not decreased by simultaneously restricting learning functions to be (fully) b-ary order independent and circumventing the restriction to recursive texts. It also implies that one can also simultaneously have a technical property we call *determination by single text* (part 2 of the theorem).

This theorem has application in [23], and the (full) b-ary order independence is important for that application.

[26] With appropriate (and straightforward) changes being made.

THEOREM 5.7 (Case and Fulk). *There is an algorithm for transforming any b and (an algorithm for) learning function $\mathbf{F}$ into a corresponding (algorithm for a) learning function $\mathbf{F}'$ such that*

1. *$\mathbf{F}'$ is b-ary order independent,*
2. *$(\forall$ r.e. $L)[\mathbf{F}'$ $\mathbf{TxtFex}_b^a$-identifies L on* some *text for $L \Rightarrow$ $\mathbf{F}'$ $\mathbf{TxtFex}_b^a$-identifies $L]$, and*
3. *$(\forall$ r.e. $L)[\mathbf{F}$ $\mathbf{RecTxtFex}_b^a$-identifies $L \Rightarrow$ $\mathbf{F}'$ $\mathbf{TxtFex}_b^a$-identifies $L]$.*

Proof. Suppose that $\mathbf{F}$ and b are given. The algorithm for $\mathbf{F}'$ is much like that in the proof of Theorem 5.5 above, with some exceptions as noted below. τ^- is as defined in (3.1). In defining $\mathbf{F}'$ on τ, we assume we have iteratively (on successively larger $\tau' \subset \tau$) kept a priority queue of programs/grammars, which queue is initially empty. Proceed initially as in the algorithm in the proof of Theorem 5.5 above, but if the value of $\langle D^1, \sigma^1\rangle$ associated with τ is $\neq$ the value of $\langle D^1, \sigma^1\rangle$ associated with τ^-, empty the priority queue, and output $\|\tau\|$; otherwise, continue down through the end of the while loop and then let

$$\sigma = \sigma^{i_\tau}, \tag{5.6}$$

where, as in the proof of Theorem 5.5, i_τ is the final value of i from the while loop for input τ. Next, in increasing order of σ' such that σ' in A, $\sigma' \leq n$, and $\sigma' \supseteq \sigma$ (σ from (5.6)), put $\mathbf{F}(\sigma')$ on the *tail* of the priority queue; when that is all done, output the *front* of the priority queue.

The outputting of $\|\tau\|$ upon witnessing an instability in the choice of $\langle D^1, \sigma^1\rangle$ is essentially a combinatorial device from [3], and it plays the role that pad did in the proof of Theorem 5.5 above, similarly controlling thrashing in the choice of $\langle D^1, \sigma^1\rangle$ when some T for L stabilizes $\mathbf{F}'$. This makes $\mathbf{F}'$ weakly b-ary order independent. Clearly, if some text for L stabilizes $\mathbf{F}'$, by the priority queue mechanism, for any T for L, $\mathbf{F}'(T)\Downarrow = D^{\text{imax}}$, where D^{imax} is from the proof of Theorem 5.5. Hence, $\mathbf{F}'$ is b-ary order independent. Clause 2 of Theorem 5.7 clearly follows. To prove clause 3 of Theorem 5.7, one can apply appropriate portions of the proof of Theorem 5.5 *mutatis mutandis*. ∎

We do not know if there are analogues of Theorems 5.5 and 5.6 above for $\mathbf{TxtMfex}_b^a$-identification. The use of pad in the proofs of those theorems wreaks havoc with program/grammar size. However, we do have the next three theorems, the first of which has application in [23].

These theorems say that we can have, for $\mathbf{TxtMfex}_b^a$-identification, without loss of learning power, either

1. b-ary order independence, determination by single text, and circumvention of the restriction to recursive texts (Theorem 5.8);
2. partly set-driven learning functions and circumvention of the restriction to recursive texts (Theorem 5.9); *or*
3. (completely) set-driven learning functions and circumvention of the restriction to recursive texts (Theorem 5.10), *but this latter conjunction is guaranteed for infinite languages only.*

THEOREM 5.8 (Case and Jain). *There is an algorithm for transforming any b and (an algorithm for) a learning function $\mathbf{F}$ into a corresponding (algorithm for a) learning function $\mathbf{F}'$ such that*

1. *$\mathbf{F}'$ is b-ary order independent,*
2. *$(\forall$ r.e. $L)[\mathbf{F}'$ $\mathbf{TxtMfex}_b^a$-identifies L on* some *text for $L \Rightarrow$ $\mathbf{F}'$ $\mathbf{TxtMfex}_b^a$-identifies $L]$, and*

3. $(\forall$ *r.e.* $L)[\mathbf{F}$ $\mathbf{RecTxtMfex}^a_b$*-identifies* $L \Rightarrow \mathbf{F}'$ $\mathbf{TxtMfex}^a_b$*-identifies* $L]$.

Proof. The proof of Theorem 5.7 suffices *mutatis mutandis*. □

The next theorem was independently noticed by Jain.

THEOREM 5.9 (Case and Jain). *There is an algorithm for transforming any* b *and* (*an algorithm for*) *a learning function* $\mathbf{F}$ *into a corresponding* (*algorithm for* a) *learning function* $\mathbf{F}'$ *such that*

1. $\mathbf{F}'$ *is partly set-driven, and*

2. $(\forall$ *r.e.* $L)[\mathbf{F}$ $\mathbf{RecTxtMfex}^a_b$*-identifies* $L \Rightarrow \mathbf{F}'$ $\mathbf{TxtMfex}^a_b$*-identifies* $L]$.

Proof. The proof of Theorem 5.5 above with the elimination of any mention of pad and weak b-ary order independence, *mutatis mutandis*, suffices to prove the present theorem. □

Similarly, the proof of Theorem 5.6 above may be modified along the lines suggested in the proof of Theorem 5.9 to prove the following theorem.

THEOREM 5.10. *There is an algorithm for transforming any* b *and* (*an algorithm for*) *a learning function* $\mathbf{F}$ *into a corresponding* (*algorithm for* a) *learning function* $\mathbf{F}'$ *such that*

1. $\mathbf{F}'$ *is set-driven, and*

2. $(\forall \infty$ *r.e.* $L)[\mathbf{F}$ $\mathbf{RecTxtMfex}^a_b$*-identifies* $L \Rightarrow \mathbf{F}'$ $\mathbf{TxtMfex}^a_b$*-identifies* $L]$.

We expect that the theorems of this section will be generally useful for work in the area.

6. Proofs deferred from section 3. In section 3 we deferred proofs of three results until we had the benefit of some of the concepts and/or results from sections 4 and 5. The present section contains those deferred proofs and, for convenience, we restate each result being proved.

Clearly, the second conclusion of Theorem 5.5 and the third conclusion of Theorem 5.8 yield the following corollary.

COROLLARY 6.1.

1. $(\forall a, b)[\mathbf{RecTxtFex}^a_b = \mathbf{TxtFex}^a_b]$.

2. $(\forall a, b)[\mathbf{RecTxtMfex}^a_b = \mathbf{TxtMfex}^a_b]$.

As we noted in section 3, the proof of the next theorem depends, in part, on Definitions 5.1 and 5.4 and Theorem 5.6.

THEOREM 6.2. *Let* $\mathcal{L}_{n+1}$ *equal*

$$\{\, L \mid L \text{ is } \infty \wedge (\exists e_0, \ldots, e_n)[W_{e_0} = \cdots = W_{e_n} = L \wedge (\forall^\infty \langle x, y\rangle \in L)[y \in \{\, e_0, \ldots, e_n \,\}]] \,\}.$$

Then $\mathcal{L}_{n+1} \in (\mathbf{TxtFex}^0_{n+1} - \mathbf{TxtFex}^*_n)$.

Proof. Clearly, $\mathcal{L}_{n+1} \in \mathbf{TxtFex}^0_{n+1}$.[27]

Suppose for contradiction that $\mathbf{F}$ $\mathbf{TxtFex}^*_n$-identifies $\mathcal{L}_{n+1}$. Each member of $\mathcal{L}_{n+1}$ is infinite; hence, thanks to Theorem 5.6, we may suppose, without loss of generality, that $\mathbf{F}$ is set-driven *and* weakly n-ary order independent. Therefore, in particular, we may write $\mathbf{F}(D)$ for $\mathbf{F}(\sigma)$, where $D = \text{content}(\sigma)$. By implicit application of a padded version of the $n+1$-ary recursion theorem there are *distinct* self-other referentials $e_0, e_1, \ldots, e_n$ defining $W_{e_0}, W_{e_1}, \ldots, W_{e_n}$, respectively, in successive stages s as follows.[28]

For each $i \leq n$, let $W_{e_i,s}$ = the finitely much of W_{e_i} defined *before* stage s described below; also set $W_{e_i,0} = \emptyset$. Go to stage 0.

[27] The role of self-reference in this proof is, in part, to make this positive portion of the theorem immediate while scarcely affecting the difficulty of the negative portion.

[28] The padding is just to make $e_0, e_1, \ldots, e_n$ syntactically pairwise distinct. It should be clear in the staging construction how each e_i significantly uses its knowledge of $e_0, \ldots, e_i, \ldots, e_n$.

begin *stage s*
if $\text{card}(\{\, \mathbf{F}(W_{e_0,s}), \mathbf{F}(W_{e_1,s}), \ldots, \mathbf{F}(W_{e_n,s}) \,\}) \leq n$
then
for each $i \leq n$, set $W_{e_i,s+1} = W_{e_i,s} \cup \{\, \langle s, e_i \rangle \,\}$
else
for each $i \leq n$, set $W_{e_i,s+1} = [[\bigcup_{j \leq n} W_{e_j,s}] \cup \{\, \langle s, e_0 \rangle \,\}]$
endif;
go to stage $s+1$
end ($*$ *stage* s $*$).

Case 1. $(\forall^\infty s)[\text{card}(\{\, \mathbf{F}(W_{e_0,s}), \mathbf{F}(W_{e_1,s}), \ldots, \mathbf{F}(W_{e_n,s}) \,\}) \leq n]$. Then each of W_{e_0}, $W_{e_1}, \ldots, W_{e_n} \in \mathcal{L}_{n+1}$, yet they are pairwise $\neq^*$. Hence, since this is Case 1, at each sufficiently large stage s, for at least one of the $(n+1)$ i's $\leq n$, program/grammar $\mathbf{F}(W_{e_i,s})$ fails to generate a finite variant of W_{e_i}. Therefore, for at least one $i \leq n$, $(\exists^\infty s)[W_{\mathbf{F}(W_{e_i,s})} \neq^* W_{e_i}]$. Hence, *this* W_{e_i} is *not* $\mathbf{TxtFex}^*_n$-identified by $\mathbf{F}$, a contradiction.

Case 2. $(\exists^\infty s)[\text{card}(\{\, \mathbf{F}(W_{e_0,s}), \mathbf{F}(W_{e_1,s}), \ldots, \mathbf{F}(W_{e_n,s}) \,\}) = n+1]$ (say, at stages $s_0 < s_1 < s_2 < \cdots$). Then $W_{e_0} = W_{e_1} = \cdots = W_{e_n} \in \mathcal{L}_{n+1}$. Let $\vec{W}_{e_i,s}$ be an increasing order enumeration of $W_{e_i,s}$. Hence, $\vec{W}_{e_i,s}$ is also a finite initial segment of a text. Let $T_i = \vec{W}_{e_i,s_0} \diamond \vec{W}_{e_i,s_1} \diamond \vec{W}_{e_i,s_2} \diamond \cdots$. Clearly T_i is a text for W_{e_i}, which equals W_{e_0}. Since $\mathbf{F}$ is weakly n-ary order independent, there is a set D of cardinality $\leq n$ such that

$$\bigcup_{i \leq n} \mathbf{F}(T_i)\Downarrow \subseteq D. \tag{6.1}$$

However, since this is Case 2 and by the choice of $s_0, s_1, s_2, \ldots$, the left-hand side of (6.1) has cardinality $> n$, a contradiction. □

As we noted in section 3 above, the proof of the next theorem depends on Theorem 4.4.

THEOREM 6.3. $\mathbf{TxtFex}^m_* \subseteq \mathbf{TxtBc}^{m'} \Leftrightarrow m \leq 2.m'$; *furthermore*, $\{\, L \mid L =^{2m+1} \mathbf{N} \,\} \in (\mathbf{TxtFex}^{2m+1}_1 - \mathbf{TxtBc}^m)$.

Proof. This proof employs previously unpublished techniques used to prove a similar result for $\mathbf{TxtFex}^a_1$ in [28].

Suppose that $\mathbf{F}$ $\mathbf{TxtFex}^{2m}_*$-identifies $\mathcal{L}$. We will construct an $\mathbf{F}'$ which $\mathbf{TxtBc}^m$-identifies $\mathcal{L}$, and then it will suffice to prove the *furthermore* clause.

Define $\mathbf{F}'$ on τ thus. First calculate $p = \mathbf{F}(\tau)$. By Kleene's S-m-n theorem [82], find p_τ such that $W_{p_\tau} = ((W_p \cup \text{content}(\tau)) -$ the m least numbers not in $\text{content}(\tau))$. Output p_τ.

Suppose that T is a text for $L \in \mathcal{L}$. Let $D = \mathbf{F}(T)\Downarrow$. Hence, $(\forall p \in D)[W_p =^{2m} L]$. For all sufficiently large $\tau \in T$, $p = \mathbf{F}(\tau) \in D$ and p_τ patches any mistakes of omission of p; furthermore, p_τ removes m elements, including up to m of the mistakes of commission of p (if any) and, perhaps in the process, it creates new mistakes of omission.

Case 1. The number of mistakes of commission in such a p is $\geq m$. Of course this number of mistakes is $\leq 2m$. Then p_τ removes m of these mistakes of commission leaving a residue of $\leq m$ errors.

Case 2. The number m' of mistakes of commission in such a p is $< m$. Then p_τ removes *all* these errors of commission, but creates $m - m'$ new errors (of omission); however, this number is still $\leq m$.

In each case, for such p, p_τ has $\leq m$ errors. Therefore, $\mathbf{F}'$ $\mathbf{TxtBc}^m$-identifies $L \in \mathcal{L}$.

Let $\mathcal{L} = \{ L \mid L =^{2m+1} \mathbf{N} \}$. Clearly, $\mathcal{L} \in \mathbf{TxtFex}_1^{2m+1}$. Suppose for contradiction that $\mathcal{L} \in \mathbf{TxtBc}^m$ as witnessed by learning function $\mathbf{F}$. Hence, in particular, $\mathbf{F}$ $\mathbf{TxtBc}^m$-identifies $\mathbf{N}$. Therefore, by Theorem 4.4,

(6.2) $(\exists D \text{ finite })(\forall L \mid D \subseteq L \ \wedge \ L \neq^{2m} \mathbf{N})[\mathbf{F} \text{ does not } \mathbf{TxtBc}^m\text{-identify } L]$.

Pick $L \supseteq D$ such that $\text{card}(\overline{L}) = 2m + 1$. Then $L \in \mathcal{L}$, but by (6.2), $\mathbf{F}$ does not $\mathbf{TxtBc}^m$-identify L, a contradiction. □

7. Concluding remarks. In this section we discuss briefly computable universe hypotheses, present some critical discussion about the applicability to human language learning of Gold-style models and our main theorem (Theorem 3.3 above), and sketch some areas for future investigation.

We have considered (among other possibilities) computable models of learning on computable data sequences. The whole universe or humanly significant portions of it may be computable and/or discrete. Such possibilities are taken seriously—for example, in [99, 92, 91, 34, 17, 15, 29]. In a discrete, *random* universe with only computable probability distributions for its behavior (e.g., a discrete, quantum mechanical universe), the *expected* behavior will still be computable [32, 42, 43].[29] In such a universe any beings (e.g., humans) who have cognition, including language learning and scientific induction, will be subject to the constraint that at least their expected behavior will be computable; hence, any theorems about computable learning agents will inform us, to some extent, of the possible behaviors of those beings. It would appear that human genetic programs make use of error correction in an attempt to circumvent "random" influences, including those from the quantum mechanical level. It is plausible that human cognitive programs built on top of the wetware the genetic programs partly construct do likewise. Hence, computability of cognition may be a pretty good model.

Even if cognition is computable (although perhaps too complicated for mere humans to figure out how it's done), there are still problems realistically modeling human language learning with Gold's paradigm. References [58, 59] present empirical evidence that semantics in addition to positive information may be essential to human language learning. It seems clear that denotation and social reinforcers play crucial roles in the human case, but not in Gold's paradigm. In [15] the report on Chapter 6 of [38] is partly motivated by treating negative information as a more mathematically tractable possible substitute for semantic information. Reference [61] notes that in homes where parents do supply improvements to child utterances (a subtle form of correction or negative information), there is increased *speed* of language acquisition. It is not clear if the relation is causal, but Theorem 22 in [5] implies there are cases where a significant improvement in language learning *speed* (as calibrated by the number of mind-changes required to reach a single final correct grammar) results from the presence of minimal negative information. Largely unexplored, but of some interest, is the extension of [5] to $\mathbf{TxtFex}_b^a$-identification.

We originally suggested in [16] on the basis of our main corollary (Corollary 3.7 to Theorem 3.3) that Gold's model be extended to embrace the success criteria $\mathbf{TxtFex}_b^a$

[29] Sources such as [73, 74], sadly, seem to have overlooked the important result in [32] that the expected input/output behavior of a Turing machine with random oracle subject to a computable probability distribution is computable (and constructively so).

for "small" values of a and b. We consider next a possible difficulty. In the proof of Theorem 3.3, for each $\mathbf{F}$, the associated set(s) $W_{e_0}, W_{e_1}, \ldots, W_{e_n}$ may, in some cases, differ considerably in computational complexity from one another, and Osherson pointed out to us that there is no apparent corresponding vacillation in human language performance. However, in the proof of Theorem 3.3, for each $\mathbf{F}$, the associated set(s) $W_{e_0}, W_{e_1}, \ldots, W_{e_n}$ are each actually recursive; hence, for each $\mathbf{F}$, there is a Blum complexity measure Φ [10, 48] such that $\Phi_{e_0} = \Phi_{e_1} = \cdots = \Phi_{e_n}$; therefore, if performance were measured by such a Φ, vacillatory learning would increase learning power but without a corresponding vacillation in performance. Technical questions remain open regarding which stronger quantificational variants of the argument in the previous sentence can be made. In another direction, we note that the proof of Theorem 3.3 permits a modification so that the relative density of output of all the final programs/grammars but one is as small as we like. Hence, the performance vacillation may exist, but significant degradations in articulateness potential might be confined to rare episodes. Even if such episodes do not exist for humans, they might be tolerated in an artificial system.

In spite of the limitations to date of modeling human language learning with (extensions of) Gold's paradigm, we believe that many of the theorems (e.g., Theorem 4.4) in this area nonetheless give some insights. The state of the art is weakly analogous to modeling the thermodynamics of fluids without taking into account van der Waal's forces: one may still get some understanding of the reality so modeled.

Speaking of Theorem 4.4, it would be mathematically interesting to explore what happens to the subset principle for **TxtBc**-identification restricted to *recursive* texts.

It is interesting to place further feasibility restrictions on the criteria of success. As noted in section 5 above, [23] studies $\mathbf{TxtMfex}^a_b$-identification, the restricted variant of $\mathbf{TxtFex}^a_b$-identification which requires that final programs/grammars be nearly minimal size. For language learning, bounding complexity of learning machines as in [33] or [24] remains to be explored. Translating relative solvability results into relative feasibility results, as in [97], would be very interesting to pursue in the context of the present paper. In section 5 there are several results about no loss of learning power in passing from some learning function $\mathbf{F}$ to an insensitive or restricted learning function $\mathbf{F}'$. How does the complexity of such $\mathbf{F}'$'s compare to that of $\mathbf{F}$? If the complexity of $\mathbf{F}'$ in some cases must be significantly greater than that of $\mathbf{F}$, then one could plausibly conjecture that child language learning is highly sensitive to the order of data presentation.

Can we get versions of our separation results robust in the sense of [40]?

Much of the work in Gold-style learning theory on success criteria extending Gold's is motivated by attempts to assuage the negative results in this area. Reference [53] mentions a common argument to the effect that very strong negative results about language learnability in [46] provide evidence that human language learning must involve some innately stored information! The negative results suggest, among other things, that

1. general purpose learning is not possible, and
2. alleged human general purpose learning is an illusion brought about by our having innate information stored for a large and varied collection of domains [41, 89].

In the practical context of robot planning, McDermott [60] says, "Learning makes the most sense when it is thought of as filling in the details in an algorithm that is

already nearly right." In the context of function learning, [27] provides several models of learning from examples *together with approximately correct programs*. Included are models in which the maximal probability of learning *all* the computable functions is proportional to how tightly the approximately correct programs envelope the data. Unexplored, but very interesting, is how to provide such models for language learning from positive data. Success might provide some insight into the form of innate knowledge for human language learning.

Acknowledgments. The author thanks the University of Rochester's Computer Science Department for support and for providing an excellent working environment in the academic year 1987–88, during which some of the work on the present paper was completed. The author is also grateful to the anonymous referees and others mentioned in the text for many helpful comments.

REFERENCES

[1] D. Angluin, *Inductive inference of formal languages from positive data*, Inform. and Control, 45 (1980), pp. 117–135.
[2] D. Angluin, *Inference of reversible languages*, J. ACM, 29 (1982), pp. 741–765.
[3] L. Blum and M. Blum, *Toward a mathematical theory of inductive inference*, Inform. and Control, 28 (1975), pp. 125–155.
[4] G. Baliga and J. Case, *Learnability: Admissible, co-finite, and hypersimple sets*, J. Comput. System Sci., 53 (1996), pp. 26–32.
[5] G. Baliga, J. Case, and S. Jain, *Language learning with some negative information*, J. Comput. System Sci., 51 (1995), pp. 273–285.
[6] G. Baliga, J. Case, and S. Jain, *Synthesizing enumeration techniques for language learning*, in Proceedings of the Ninth Annual Conference on Computational Learning Theory, Desenzano del Garda, Italy, ACM Press, New York, 1996, pp. 169–180.
[7] G. Baliga, J. Case, S. Jain, and M. Suraj, *Machine learning of higher order programs*, J. Symbolic Logic, 59 (1994), pp. 486–500.
[8] R. Berwick, *The Acquisition of Syntactic Knowledge*, MIT Press, Cambridge, MA, 1985.
[9] R. Brown and C. Hanlon, *Derivational complexity and the order of acquisition in child speech*, in Cognition and the Development of Language, J. Hayes, ed., Wiley, New York, 1970.
[10] M. Blum, *A machine independent theory of the complexity of recursive functions*, J. ACM, 14 (1967), pp. 322–336.
[11] M. Blum, *On the size of machines*, Inform. and Control, 11 (1967), pp. 257–265.
[12] J. Barzdin and K. Podnieks, *On the theory of inductive inference*, in Proceedings of the Mathematical Foundations for Computer Science, Math. Inst. Slovak Acad. Sci., Bratislava, 1973, pp. 9–15 (in Russian).
[13] M. Braine, *On two types of models of the internalization of grammars*, in The Ontogenesis of Grammar: A Theoretical Symposium, D. Slobin, ed., Academic Press, New York, 1971.
[14] J. Case, *Periodicity in generations of automata*, Math. Systems Theory, 8 (1974), pp. 15–32.
[15] J. Case, *Learning machines*, in Language Learning and Concept Acquisition, W. Demopoulos and A. Marras, eds., Ablex, Stanford, CT, 1986.
[16] J. Case, *The power of vacillation*, in Proceedings of the Workshop on Computational Learning Theory, D. Haussler and L. Pitt, eds., Morgan Kaufmann, San Francisco, 1988, pp. 133–142.
[17] J. Case, *Turing machine*, in Encyclopedia of Artificial Intelligence, 2nd ed., S. Shapiro, ed., John Wiley and Sons, New York, 1992.
[18] J. Case, *Infinitary self-reference in learning theory*, J. Experiment. and Theoret. Art. Intell., 6 (1994), pp. 3–16.
[19] K. Chen, *Tradeoffs in Machine Inductive Inference*, Ph.D. thesis, Computer Science Department, State University of New York at Buffalo, Buffalo, NY, 1981.
[20] K. Chen, *Tradeoffs in the inductive inference of nearly minimal size programs*, Inform. and Control, 52 (1982), pp. 68–86.
[21] J. Case, S. Jain, and S. Ngo Manguelle, *Refinements of inductive inference by Popperian and reliable machines*, Kybernetika, 30 (1994), pp. 23–52.

[22] J. Case, S. Jain, and A. Sharma, *On learning limiting programs*, Internat. J. Found. Comput. Sci., 3 (1992), pp. 93–115.
[23] J. Case, S. Jain, and A. Sharma, *Vacillatory learning of nearly minimal size grammars*, J. Comput. System Sci., 49 (1994), pp. 189–207.
[24] J. Case, S. Jain, and A. Sharma, *Complexity issues for vacillatory function identification*, Inform. and Comput., 116 (1995), pp. 174–192.
[25] J. Case, S. Jain, and F. Stephan, *Vacillatory and BC learning on noisy data*, in Proceedings of the 7th International Workshop on Algorithmic Learning Theory (ALT'96), Sydney, Australia, October, 1996, Lecture Notes in Artificial Intelligence 1160, Springer-Verlag, Berlin, 1996, pp. 285–298.
[26] J. Case, S. Jain, and A. Sharma, *Synthesizing noise-tolerant language learners*, in Proceedings of the 8th International Workshop on Algorithmic Learning Theory (ALT'97), Sendai, Japan, Lecture Notes in Artificial Intelligence, Springer, Berlin, 1997.
[27] J. Case, S. Kaufmann, E. Kinber, and M. Kummer, *Learning recursive functions from approximations*, J. Comput. System Sci., 55 (1997), pp. 183–196.
[28] J. Case and C. Lynes, *Machine inductive inference and language identification*, in Proceedings of the 9th Annual Colloquium on Automata, Languages, and Programming (July 1982), Lecture Notes in Comput. Sci., 140, Springer-Verlag, Berlin, pp. 107–115.
[29] J. Case, D. Rajan, and A. Shende, *Representing the spatial/kinematic domain and lattice computers*, J. Experiment. Theoret. Art. Intell., 6 (1994), pp. 17–40.
[30] J. Case and C. Smith, *Anomaly hierarchies of mechanized inductive inference*, in Proceedings of the 10th Annual Symposium on the Theory of Computing, ACM, New York, 1978, pp. 314–319.
[31] J. Case and C. Smith, *Comparison of identification criteria for machine inductive inference*, Theoret. Comput. Sci., 25 (1983), pp. 193–220.
[32] K. deLeeuw, E. Moore, C. Shannon, and N. Shapiro, *Computability by probabilistic machines*, in Automata Studies, Ann. of Math. Stud., 34, Princeton University Press, Princeton, NJ, 1956, pp. 183–212.
[33] R. Daley and C. Smith, *On the complexity of inductive inference*, Inform. and Control, 69 (1986), pp. 12–40.
[34] R. Feynman, *Simulating physics with computers. Phusics of computation, Part* II, Internat. J. Theoret. Phys., 21 (1981/82), pp. 467–488.
[35] R. Freivalds, *Minimal Gödel numbers and their identification in the limit*, Lecture Notes in Comput. Sci., 32, Springer-Verlag, Berlin, 1975, pp. 219–225.
[36] R. Freivalds, *Recursiveness of the enumerating functions increases the inferrability of recursively enumerable sets*, Bull. Euro. Assoc. Theoret. Comput. Sci., 27 (1985), pp. 35–40.
[37] R. Freivalds, *Inductive inference of minimal programs*, in Proceedings of the Third Annual Workshop on Computational Learning Theory, M. Fulk and J. Case, eds., Morgan Kaufmann, San Francisco, 1990, pp. 3–20.
[38] M. Fulk, *A Study of Inductive Inference Machines*, Ph.D. thesis, State University of New York at Buffalo, Buffalo, NY, 1985.
[39] M. Fulk, *Prudence and other conditions on formal language learning*, Inform. and Comput., 85 (1990), pp. 1–11.
[40] M. Fulk, *Robust separations in inductive inference*, in Proceedings of the 31st Annual Symposium on Foundations of Computer Science, St. Louis, MO, IEEE Press, Piscataway, NJ, 1990, pp. 405–410.
[41] C. Gallistel, A. Brown, S. Carey, R. Gelman, and F. Keil, *Lessons from animal learning for the study of cognitive development*, in Epigenesis of Mind: Essays on Biology and Cognition, S. Carey and R. Gelman, eds., Erlbaum, Hillsdale, NJ, 1991, pp. 3–37.
[42] J. Gill, *Probabilistic Turing Machines and Complexity of Computation*, Ph.D. thesis, University of California, Berkeley, CA, 1972.
[43] J. Gill, *Computational complexity of probabilistic Turing machines*, SIAM J. Comput. 6 (1977), pp. 675–695.
[44] L. Gleitman, *Biological dispositions to learn language*, in Language Learning and Concept Acquisition, W. Demopoulos and A. Marras, eds., Ablex, Stanford, CT, 1986.
[45] K. Gödel, *On formally undecidable propositions of Principia Mathematica and related systems* I, in Kurt Gödel: Collected Works, Vol. I, S. Feferman, ed., Oxford University Press, Oxford, UK, 1986, pp. 145–195.
[46] E. Gold, *Language identification in the limit*, Inform. and Control, 10 (1967), pp. 447–474.
[47] P. Halmos, *Naive Set Theory*, Springer-Verlag, New York, 1974.
[48] J. Hopcroft and J. Ullman, *Introduction to Automata Theory Languages and Computation*, Addison-Wesley, Reading, MA, 1979.

[49] T. Jech, *Set Theory*, Academic Press, New York, 1978.
[50] P. Johnson-Laird, *The Computer and the Mind: An Introduction to Cognitive Science*, Harvard University Press, Cambridge, MA, 1988.
[51] S. Jain and A. Sharma, *Prudence in vacillatory language identification*, Math. Systems Theory, 28 (1995), pp. 267–279.
[52] E. Kinber, *On a theory of inductive inference*, in Fundamentals of Computation Theory, Lecture Notes in Comput. Sci. 56, Springer-Verlag, Berlin, 1977, pp. 435–440.
[53] D. Kirsh, *PDP learnability and innate knowledge of language*, in Connectionism: Theory and Practice, S. Davis, ed., Oxford University Press, New York, 1992, pp. 297–322.
[54] S. Kapur, B. Lust, W. Harbert, and G. Martohardjono, *Universal grammar and learnability theory: The case of binding domains and the "subset principle,"* in Knowledge and Language, vol. I, E. Reuland and W. Abraham, eds., Kluwer, Dordrecht, 1993, pp. 185–216.
[55] S. Kurtz and J. Royer, *Prudence in language learning*, in Proceedings of the Workshop on Computational Learning Theory, D. Haussler and L. Pitt, eds., Morgan Kaufmann, San Francisco, 1988, pp. 143–156.
[56] S. Lange and P. Watson, *Machine discovery in the presence of incomplete or ambiguous data*, in Algorithmic Learning Theory, Reinhardsbrunn Castle, Germany, Lecture Notes in Artificial Intelligence 872, K. Jantke and S. Arikawa, eds., Springer-Verlag, Berlin, 1994, pp. 438–452.
[57] S. Lange, T. Zeugmann, and S. Kapur, *Characterizations of monotonic and dual monotonic language learning*, Inform. and Comput., 120 (1995), pp. 155–173.
[58] D. Moeser and A. Bregman, *The role of reference in the acquisition of a miniature artificial language*, J. Verbal Learning and Verbal Behavior, 11 (1972), pp. 759–769.
[59] D. Moeser and A. Bregman, *Imagery and language acquisition*, J. Verbal Learning and Verbal Behavior, 12 (1973), pp. 91–98.
[60] D. McDermott, *Robot planning*, AI Magazine, 13 (1992), pp. 55–79.
[61] D. McNeill, *Developmental psycholinguistics*, in The Genesis of Language, F. Smith and G. A. Miller, eds., MIT Press, Cambridge, MA, 1966, pp. 15–84.
[62] E. Mendelson, *Introduction to Mathematical Logic*, 3rd ed., Brooks-Cole, San Francisco, CA, 1986.
[63] Y. Mukouchi, *Characterization of finite identification*, in Proceedings of the Third International Workshop on Analogical and Inductive Inference, Dagstuhl Castle, Germany, October 1992, Lecture Notes in Artificial Intelligence 642, K. P. Jantke, ed., Springer-Verlag, Berlin, 1992, pp. 260–267.
[64] R. Manzini and K. Wexler, *Parameters, binding theory and learnability*, Linguistic Inquiry, 18 (1987), pp. 413–444.
[65] M. Machtey and P. Young, *An Introduction to the General Theory of Algorithms*, North-Holland, Amsterdam, 1978.
[66] D. Osherson, M. Stob, and S. Weinstein, *Ideal learning machines*, Cognitive Sci., 6 (1982), pp. 277–290.
[67] D. Osherson, M. Stob, and S. Weinstein, *Note on a central lemma of learning theory*, J. Math. Psych., 27 (1983), pp. 86–92.
[68] D. Osherson, M. Stob, and S. Weinstein, *Learning theory and natural language*, Cognition, 17 (1984), pp. 1–28.
[69] D. Osherson, M. Stob, and S. Weinstein, *An analysis of a learning paradigm*, in Language Learning and Concept Acquisition, W. Demopoulos and A. Marras, eds., Ablex, Stanford, CT, 1986.
[70] D. Osherson, M. Stob, and S. Weinstein, *Systems That Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists*, MIT Press, Cambridge, MA, 1986.
[71] D. Osherson and S. Weinstein, *Criteria for language learning*, Inform. and Control, 52 (1982), pp. 123–138.
[72] D. Osherson and S. Weinstein, *A note on formal learning theory*, Cognition, 11 (1982), pp. 77–88.
[73] R. Penrose, *The Emperor's New Mind*, Oxford University Press, New York, 1989.
[74] R. Penrose, *Shadows of the Mind*, Oxford University Press, New York, 1994.
[75] J. Paris and L. Harrington, *A mathematical incompleteness in Peano arithmetic*, in Handbook of Mathematical Logic, J. Barwise, ed., North–Holland, Amsterdam, 1977.
[76] S. Pinker, *Formal models of language learning*, Cognition, 7 (1979), pp. 217–283.
[77] Z. Pylyshyn, *Computation and Cognition: Toward A Foundation For Cognitive Science*, MIT, Cambridge, MA, 1984.
[78] J. Royer and J. Case, *Subrecursive Programming Systems: Complexity and Succinctness*, Progr. Theoret. Comput. Sci., Birkhäuser, Boston, 1994.

[79] G. Riccardi, *The Independence of Control Structures in Abstract Programming Systems*, Ph.D. thesis, State University of New York at Buffalo, Buffalo, NY, 1980.
[80] G. Riccardi, *The independence of control structures in abstract programming systems*, J. Comput. System Sci., 22 (1981), pp. 107–143.
[81] H. Rogers, *Gödel numberings of partial recursive functions*, J. Symbolic Logic, 23 (1958), pp. 331–341.
[82] H. Rogers, *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York, 1967 (reprinted, MIT Press, Cambridge, MA, 1987).
[83] J. Royer, *A Connotational Theory of Program Structure*, Lecture Notes in Comput. Sci. 273, Springer-Verlag, New York, 1987.
[84] S. Simpson, *Nonprovability of certain combinatorial properties of finite trees*, in Harvey Friedman's Research on the Foundations of Mathematics, L. Harrington, M. Morley, A. Schedrov, and S. Simpson, eds., North–Holland, Amsterdam, 1985, pp. 87–117.
[85] S. Simpson, *Unprovable theorems and fast-growing functions*, in Logic and Combinatorics, Contemp. Math. 65, S. Simpson, ed., AMS, Providence, RI, 1987, pp. 359–394.
[86] C. Smith, *A Recursive Introduction to the Theory of Computation*, Springer-Verlag, New York, 1994.
[87] R. Smullyan, *Theory of Formal Systems*, Ann. of Math. Stud. 47, Princeton University Press, Princeton, NJ, 1961.
[88] R. Soare, *Recursively Enumerable Sets and Degrees*, Springer-Verlag, New York, 1987.
[89] E. Spelke, *Initial knowledge: Six suggestions*, Cognition, 50 (1994), pp. 431–445.
[90] G. Schäfer-Richter, *Über Eingabeabhangigkeit und Komplexitat von Inferenzstrategien*, Ph.D. thesis, RWTH Aachen, Aachen, Germany, 1984.
[91] T. Toffoli and N. Margolus, *Cellular Automata Machines*, MIT Press, Cambridge, MA, 1987.
[92] T. Toffoli, *Cellular Automata Machines*, Technical report 208, Comp. Comm. Sci. Dept., University of Michigan, Ann Arbor, MI, 1977.
[93] K. Wexler and P. Culicover, *Formal Principles of Language Acquisition*, MIT Press, Cambridge, MA, 1980.
[94] K. Wexler, *On extensional learnability*, Cognition, 11 (1982), pp. 89–95.
[95] K. Wexler, *The subset principle is an intensional principle*, in Knowledge and Language, vol. I, E. Reuland and W. Abraham, eds., Kluwer, Dordrecht, 1993, pp. 217–239.
[96] R. Wiehagen, *Identification of formal languages*, in Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 53, Springer-Verlag, New York, 1977, pp. 571–579.
[97] R. Wiehagen and T. Zeugmann, *Too much information can be too much for learning efficiently*, in Proceedings of the Third International Workshop on Analogical and Inductive Inference, Dagstuhl Castle, Germany, October 1992, Lecture Notes in Artificial Intelligence 642, K. Jantke, ed., Springer-Verlag, Berlin, 1992, pp. 72–86.
[98] P. Young, *Easy constructions in complexity theory: Gap and speed-up theorems*, Proc. Amer. Math. Soc., 37 (1973), pp. 555–563.
[99] K. Zuse, *Rechnender Raum*, Vieweg, Braunschweig, 1969 (translated as *Calculating Space,* Tech. Transl. AZT–70–164–GEMIT, MIT Project MAC, MIT, Cambridge, MA, 1970).

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 1970–1971

AN ASSOCIATIVE BLOCK DESIGN ABD(8,5)*

A. E. BROUWER†

To Maja, on the occasion of her seventeenth birthday.

Abstract. An associative block design is a certain balanced partition of a hypercube into smaller hypercubes. We construct such a design, thus settling the smallest open case.

AMS subject classification. 05B30

PII. S0097539797316622

An ABD(k,w) is a $b \times k$ matrix (where $b = 2^w$) with entries from $\{0,1,*\}$ such that (i) the stars form a 1-design: each row has $k-w$ stars and each column $b(k-w)/k$ stars, and (ii) the rows represent disjoint subsets of $\{0,1\}^k$. Here a row represents the set of binary vectors of length k obtained by replacing its stars in all possible ways by 0s and 1s.

This concept was introduced in 1974 by Rivest [4, 5, 6] in order to find a hash function with good worst-case behavior with respect to partial-match queries. For example, the eight rows

```
00*0   11*1
100*   011*
*100   *011
1*10   0*01
```

form an ABD$(4,3)$.

In order to save space, let us extend our alphabet with the minus sign, where a row containing r minus signs stands for the 2^r rows obtained by replacing these minus signs in all possible ways by 0s and 1s. Then the only other ABD$(4,3)$ is the following:

```
*000
*111
-*10
-0*1
-10*
```

The theory is as follows (see [1, 2, 3, 6]).

PROPOSITION 0.1. (i) ([6]) *An* ABD(k,w) *has exactly* $bw/(2k)$ 0*s and* $bw/(2k)$ 1*s in each column. In particular,* $bw/(2k)$ *is an integer.*

(ii) ([1]) *In an* ABD(k,w) *with* $w > 0$ *any given star pattern occurs in an even number of rows. Moreover, among the rows with a given star pattern there are as many with an even number of* 1*s as with an odd number of* 1*s.*

(iii) *For* $w \le 4$ *the only* ABD(k,w) *are the trivial ones with* $w = 0$ *or* $w = k$ *(represented, respectively, by a single row of stars or minus signs only) and the two examples shown above.*

(iv) ([2]) *If* $w > 3$*, then* $k \le w(w-1)/2$.

*Received by the editors February 2, 1997; accepted for publication July 2, 1997; published electronically June 3, 1999.

http://www.siam.org/journals/sicomp/28-6/31662.html

†University of Technology, Den Dolech 2, P.O. Box 513, 5600 MB Eindhoven, the Netherlands (aeb@win.tue.nl).

(v) ([3]) *There is no* ABD$(10, 5)$.

(vi) ([6]) *If* ABD(k_i, w_i) *exist for* $i = 1, 2$, *then there also is an* ABD(k_1k_2, w_1w_2).

(vii) ([1]) *Suppose that* $k \geq w > 0$ *and* $k' \geq w' > 0$ *and* $k' \geq k$ *and* $w'/k' \geq w/k$. *Then if an* ABD(k, w) *exists, and* $2^{w'}w'/(2k')$ *is an integer, then* ABD(k', w') *also exists.*

One may use generating function arguments to get more detailed information on the possible star patterns. See [1].

The purpose of this note is to show that an ABD$(8, 5)$ exists:

```
-0000***  *01*10*0
-0001***  -*1*1*11
-001*0**  **11*001
-**1010*  *10*00*0
*0*1*110  *1*0*001
**01*111  *100**11
-**1110*  -1*0*10*
-*1*00*0  *1**0110
*010**01  *1**1000
-*1*0*11  -1**1*10
*010*1*0  *101*0*1
```

Now the smallest open case is the question of whether an ABD$(12, 6)$ exists.

Acknowledgment. This note was inspired by a letter from Knuth, who asked whether there had been any progress on ABDs since 1976 and in particular whether the existence of an ABD$(8, 5)$ was still open.

REFERENCES

[1] A. E. Brouwer, *On associative block designs*, in Combinatorics (Proc. Fifth Hungarian Colloq., Keszthely, 1976), Vol. I, Colloq. Math. Soc. János Bolyai 18, North-Holland, Amsterdam, 1978, pp. 173–184.

[2] J. A. La Poutré, *A theorem on associative block designs*, Discrete Math. 58 (1986), pp. 205–208.

[3] J. A. La Poutré and J. H. van Lint, *An associative block design* ABD$(10, 5)$ *does not exist*, Utilitas Math., 31 (1987), pp. 219–225.

[4] R. L. Rivest, *On hash-coding algorithms for partial match retrieval*, in Proceedings of the 15th Annual Symposium on Switching and Automata Theory, Long Beach, CA, IEEE Comput. Soc., 1974, pp. 95–103.

[5] R. L. Rivest, *Analysis of Associative Retrieval Algorithms*, Laboratoire de recherche en informatique et automatique, IRIA rapport 54, Institut de Recherche d'Informatique et d'Automatique, Domaine de Voluceau, Rocquencourt, Le Chesnay, France, 1974.

[6] R. L. Rivest, *Partial-match retrieval algorithms*, SIAM J. Comput., 5 (1976), pp. 19–50.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 1972–1997

ON THE ROBUSTNESS OF FUNCTIONAL EQUATIONS*

RONITT RUBINFELD†

Abstract. In this paper, we study the general question of how characteristics of functional equations influence whether or not they are robust. We isolate examples of properties which are necessary for the functional equations to be robust. On the other hand, we show other properties which are sufficient for robustness. We then study a general class of functional equations, which are of the form $\forall x, y \quad F[f(x-y), f(x+y), f(x), f(y)] = 0$, where F is an algebraic function. We give conditions on such functional equations that imply robustness.

Our results have applications to the area of self-testing/correcting programs. We show that self-testers and self-correctors can be found for many functions satisfying robust functional equations, including algebraic functions of trigonometric functions such as $\tan x$, $\frac{1}{1+\cot x}$, $\frac{Ax}{1-Ax}$, $\cosh x$.

Key words. program testing, property testing, functional equations

AMS subject classifications. 68Q40, 68Q60, 68Q25, 13P99

PII. S0097539796298625

1. Introduction.

The mathematical field of functional equations is concerned with the following prototypical problem: given a set of properties (functional equations) over a particular domain, completely characterize the set of functions that satisfy them. For example, the *linearity property* over the integers is $\forall x, y \in \mathcal{Z} \quad f(x+y) - f(x) - f(y) = 0$. The functions mapping from $\mathcal{Z}$ to $\mathcal{Z}$ that satisfy the linearity property, referred to as the *solution set* of the functional equation, is $\mathcal{F} = \{f | f(x) = c \cdot x, c \in \mathcal{Z}\}$. The linearity property is one of the famous, well-studied functional equations referred to as Cauchy's equations and has been studied over many other domains and ranges with various properties (see the text by Aczél [3]). Functional equations are used widely in the study of the various functions that arise in areas such as mathematics, physics, and economics. Several general classes of functional equations have been identified. For example, algebraic addition theorems, of the form

$$\forall x, y \quad F[f(x+y), f(x), f(y)] = 0,$$

where F is any algebraic function, were used as a starting point in the development of the theory of elliptic curves by Weierstrass. Other types of functional equations include difference equations, iteration equations, multivariate functional equations, and systems of functional equations.

In section 2, we present the definition of functional equations given in [3]. For the purposes of this introduction, we define functional equations as follows: let $\mathcal{D}, \mathcal{R}$ be an arbitrary domain and range. Let $\mathcal{T}$ be a range containing 0, and $F : \mathcal{R}^k \times \mathcal{D}^k \to \mathcal{T}$ be a function that is computable via applying a finite number of known functions (in this paper we use $-, +, \times, \backslash$, hyperbolic functions, trigonometric functions, and cth roots for constant c). Let a *neighborhood* over the domain $\mathcal{D}$ be an ordered k-tuple in $\mathcal{D}^k$ and let $\mathcal{N} \subseteq \mathcal{D}^k$. The general form of a functional equation is then

*Received by the editors February 12, 1996; accepted for publication (in revised form) December 1, 1997; published electronically June 3, 1999. A preliminary version of this work has appeared in *Proc.* 35*th IEEE Conference on Foundations of Computer Science*, 1994, pp. 288–299.

http://www.siam.org/journals/sicomp/28-6/29862.html

†Department of Computer Science, Cornell University, Ithaca, NY 14853 (ronitt@cs.cornell.edu).
This research was supported by ONR Young Investigator Award N00014-93-1-0590 and United States–Israel Binational Science Foundation grant 92-00226.

TABLE 1.1
Examples of functions satisfying addition theorems over the reals.

Equation	Solution
$f(x+y)=\frac{f(x)+f(y)}{1-f(x)f(y)}$	$f(x)=\tan Ax$
$f(x+y)=\frac{f(x)f(y)-1}{f(x)+f(y)}$	$f(x)=\cot Ax$
$f(x+y)=\frac{f(x)+f(y)}{1+[f(x)f(y)/a^2]}$	$f(x)=a\tanh Bx$
$f(x+y)=\frac{f(x)f(y)}{f(x)+f(y)}$	$f(x)=C/x$
$f(x+y)=\frac{f(x)+f(y)-2f(x)f(y)}{1-2f(x)f(y)}$	$f(x)=\frac{1}{1+\cot Ax}$
$f(x+y)=\frac{f(x)+f(y)-1}{2f(x)+2f(y)-2f(x)f(y)-1}$	$f(x)=\frac{1}{1+\tan Ax}$
$f(x+y)=\frac{f(x)+f(y)+2f(x)f(y)}{1-f(x)f(y)}$	$f(x)=\frac{Ax}{1-Ax}$
$f(x+y)=\frac{f(x)+f(y)-2f(x)f(y)}{1-f(x)f(y)}$	$f(x)=\frac{-Ax}{1-Ax}$
$f(x+y)=\frac{f(x)+f(y)-2f(x)f(y)\cos a}{1-f(x)f(y)}$	$f(x)=\frac{\sin Ax}{\sin(Ax+a)}$
$f(x+y)=\frac{f(x)+f(y)-2f(x)f(y)\cosh a}{1-f(x)f(y)}$	$f(x)=\frac{\sinh Ax}{\sinh(Ax+a)}$
$f(x+y)=\frac{f(x)+f(y)+2f(x)f(y)\cosh a}{1-f(x)f(y)}$	$f(x)=\frac{-\sinh Ax}{\sinh(Ax+a)}$
$f(x+y)=f(x)f(y)-\sqrt{1-f(x)^2}\sqrt{1-f(y)^2}$	$f(x)=\cos(Ax)$
$f(x+y)=f(x)f(y)+\sqrt{f(x)^2-1}\sqrt{f(y)^2-1}$	$f(x)=\cosh(Ax)$

$\forall(x_1,\ldots,x_k)\in\mathcal{N},\ F[f(x_1),\ldots,f(x_k),x_1,\ldots,x_k]=0$. We denote the functional equation by $(F,\mathcal{N})$ when $\mathcal{D},\mathcal{R},\mathcal{T}$ are understood from the context. A particular solution of a functional equation is a function $f:\mathcal{D}\to\mathcal{R}$ for which F evaluates to 0 on all choices of neighborhoods in $\mathcal{N}$. The general solution, $\mathcal{F}$, is the family of functions that are solutions to the functional equation. Tables 1.1 and 1.2 give several examples of functional equations and their solution sets over the reals [3], [25]. In section 2, we describe the formal definition of *characterizations* as given by Rubinfeld and Sudan in [38], which can be viewed as a generalization of functional equations.

All functional equations involve a "for all" quantifier. Here we are interested in comparing the solution with the functional equation when the "for all" quantifier is replaced by a "for most" quantifier. To illustrate, we give a simplified definition of *robustness*. For a given δ, define $\mathcal{G}\stackrel{\text{def}}{=}\{f|Pr_{(x_1,\ldots,x_k)\in\mathcal{N}}[F[f(x_1),\ldots,f(x_k),x_1,\ldots,x_k]=0]\geq 1-\delta\}$. Clearly $\mathcal{G}$ contains $\mathcal{F}$. However, is it the case that each function in $\mathcal{G}\backslash\mathcal{F}$ is essentially the same (equal on most inputs in $\mathcal{D}$) as some function in $\mathcal{F}$? Slightly more precisely, we say that two functions are ϵ-*close* over $\mathcal{D}$ if $\frac{|\{x\in\mathcal{D}|f(x)\neq g(x)\}|}{|\mathcal{D}|}\leq\epsilon$. For some small constant ϵ, if each function in $\mathcal{G}$ is ϵ-close to some function in $\mathcal{F}$, then in some sense, the "for most" quantifier is sufficient to characterize the same class of functions as the "for all" quantifier, and we say that the functional equation is (ϵ,δ)-

TABLE 1.2
Examples of functions satisfying $F[f(x-y), f(x+y), f(x), f(y)] = 0$ over the reals.

Equation	Solution
$f(x+y)+f(x-y)=2f(x)$	$f(x)=Ax+a$
$f(x+y)+f(x-y)=2f(x)f(y)$	$f(x)=0, \cos Ax, \cosh Ax$
$f(x+y)+f(x-y)=2[f(x)+f(y)]$	$f(x)=Ax^2$
$f(x+y)-f(x-y)=2f(y)$	$f(x)=Ax$
$f(x+y)f(x-y)=f(x)^2$	$f(x)=a$
$f(x+y)-f(x-y)=4\sqrt{f(x)f(y)}$	$f(x)=Ax^2$
$f(x+y)f(x-y)=f(x)^2-f(y)^2$	$f(x)=Ax,\ k\sin Ax,\ k\sinh Ax$

robust. A formal and more general definition due to [38] is given in section 2. Often it is the case that $\mathcal{N}$ and F are defined and are known to be (ϵ,δ)-robust over an infinite *set* $\mathcal{S}$ of domains and corresponding neighborhood sets. For example, the linearity property can be defined for all domains that are groups where for each group G, the corresponding neighborhood set is $\{x, y, x +_G y | x \in G\}$ ($+_G$ is the group operation for G), and the linearity property is $\forall x, y, x +_G y\ f(x +_G y) -_G f(x) -_G f(y) = 0$. The linearity property is known to be $(2\delta,\delta)$-robust when the domain and range are any finite group for any $\delta < \frac{2}{9}$ [26]. We are interested in the case when for all $\epsilon < 1$, there is a constant δ such that $(F,\mathcal{N})$ is (ϵ,δ)-robust over each of the domains in $\mathcal{S}$. In this case, we say that $(F,\mathcal{N})$ is robust over $\mathcal{S}$ (note that robustness is only interesting if $\frac{1}{\delta}$ is much smaller than $|\mathcal{N}|$).

Previous results on robust characterizations. Robustness and related notions are used implicitly in a number of works [19], [9], [29], [10], [7], [6]. In the following sections, we describe the applications of robustness to program testing and to the study of probabilistically checkable proof systems.

There are many characterizations that are known to be robust: the first nontrivial characterization shown to be robust for constant ϵ, δ was the linearity property over finite groups in the work of Blum, Luby, and Rubinfeld [19]. Coppersmith [26] gives a particularly elegant proof of the robustness of the linearity property as well as improves the allowable parameters of ϵ, δ to the following: if $f(x+y)-f(x)-f(y) = 0$ is satisfied for a constant greater than $\frac{7}{9}$ fraction of the choices of x, y in the group G, then there is some function $g(z) = c \cdot z$ such that $f(x) = g(x)$ for at least $\frac{5}{9}$ of the x in G. Coppersmith also gives an example which shows that $\delta = \frac{7}{9}$ is a type of a threshold; i.e., there is a function which satisfies $f(x+y) - f(x) - f(y) = 0$ for $\frac{7}{9}$ fraction of the choices of x, y in the group G, but which does not agree with any linear function on more than $\frac{1}{3}$ of the domain. Bellare et al. [11] show that one can get a tighter result on the range of δ that is useful over domains of the type $GF(2)^n$ where Coppersmith's example does not apply. Robust characterizations of total degree d polynomials are given in [38].[1] Robust characterizations of maximum degree d polynomials are given

[1]The total degree of a polynomial is the maximum over all terms of the total degree of a term. The total degree of a term is the sum of the individual degrees of each variable in the term.

in several works [9], [29], [10], [38].[2] These results apply to polynomials over finite fields $\mathcal{Z}_p$ (p prime) and finite subsets of rational domains. The first formal definition of robustness was given in [38]. Very recently, robust characterizations of functions satisfying linear recurrence relations have been given by Kumar and Sivakumar [33].

Our results. Our goal is to characterize the fundamental characteristics of functional equations that make them robust, in order to gain an understanding of how broadly robustness applies. It happens that the structure of the neighborhoods in $\mathcal{N}$ is very important to whether a characterization is robust. We present a graph theoretic characterization of neighborhood sets $\mathcal{N}$, which is used to quantify the connectivity of $\mathcal{N}$. In Theorem 3.2, we show that high connectivity of $\mathcal{N}$ is necessary for $\mathcal{N}$ to be robust. Since the functional equations which relate inputs that are linear functions of a single variable (e.g., $\forall x,\ f(x) - f(x+1) - 1 = 0$) are known not to have this connectivity property, we can conclude that they are not robust. On the other hand, in Theorem 3.4, we show that when $\mathcal{N} = \mathcal{D}^k$, and the set of solutions to $(\mathcal{F}, \mathcal{N})$ is rich enough, $(\mathcal{F}, \mathcal{N})$ is robust.

We next investigate conditions on the class of functional equations of the general form $\forall x, y\ \ F[f(x-y), f(x+y), f(x), f(y)] = 0$ that imply robustness. We focus on domains that are finite groups and certain types of subsets of infinite groups, such as those of the form $\mathcal{D}_{n,s} = \{\frac{i}{s} \big|\ |i| \leq n\}$ (see the beginning of subsection 2.1) and others that are of use in studying periodic functions. In the case of domains that are finite groups and domains used for studying periodic functions, testing that a function satisfies a functional equation over a domain will involve neighborhood sets that are chosen from the same domain. In the case of domains that are subsets of infinite groups of the form $\mathcal{D}_{n,s} = \{\frac{i}{s} \big|\ |i| \leq n\}$, testing that a function satisfies a functional equation will involve neighborhood sets that are chosen from a larger subset of the same infinite group. Our results apply to ranges that have a group structure. In Theorems 4.1 and 6.4, we show that if the equation can be written as $\forall x, y\ \ f(x+y) = G[f(x), f(y)]$ (a special case of algebraic addition theorems referred to as an *addition theorem*), then it is robust as long as G satisfies $G[a, G[b, c]] = G[G[a, b], c]\ \forall a, b, c$ (which is satisfied by all of our examples of addition theorems). The proofs for all types of domains rely on the same techniques. Since the proofs of the results over finite group domains are simpler to state, we give them first in order to highlight the main ideas. This work leads to self-testers for several families of trigonometric functions including $\tan Ax$, $\frac{1}{1+\cot x}$, $\frac{Ax}{1-Ax}$, $\cosh Ax$, and several examples from [3], [4], [22] given in Table 1.1. A general format for constructing self-testers is given in sections 5 and 6, and a self-tester for the particular example of the cosh function is given in section 6.3. We then give techniques that apply to functions which satisfy other functional equations (the first three examples in Table 1.2), including d'Alembert's equation $\forall x, y\ \ f(x+y) + f(x-y) = 2f(x)f(y)$ in section 4.2. In this case, the range must be a field containing 2.

Robustness and self-testing/correcting. In order to allow a programmer to use programs that are not known to be correct on all inputs, result checkers were introduced by Blum [15] and Blum and Kannan [18], and soon after, the related paradigms of self-testers and self-correctors were introduced by Blum, Luby, and Rubinfeld [19]. (A notion similar to self-correctors was independently proposed by Lipton [34].) The paradigm of self-testers and self-correctors is intended to fit into

[2]The maximum degree of a polynomial is the maximum over all variables of the maximum degree of the variable in any term.

the framework of result checkers, and in fact it is observed that a self-tester and a self-corrector for a function can be combined to give a checker [19]. If a function has a checker, then one can determine whether program P is giving the correct answer on a particular input or whether there is a bug in the program. If a function has a self-corrector, then given a program P for computing the function that is correct on most inputs, one can transform P into a new randomized program that is correct on each input with high probability and is almost as efficient as running P. Self-testers allow one to ascertain that P is correct on a large enough fraction of the inputs so that it is capable of being self-corrected. More formal definitions of self-testers and self-correctors are given in section 5. If a function has both a self-tester and a corresponding self-corrector, then an unreliable program can be used to reliably compute the function.

Problems that can be viewed as linear or low degree polynomial functions, such as matrix multiplication, integer division, sine/cosine, integer multiplication, the mod function, modular multiplication, polynomial multiplication, modular exponentiation, fast Fourier transform and determinant, have been shown to have self-testers and self-correctors [19], [8], [34], [23], [31], [37], [38], [2], [28], [21]. Although many functions can be viewed as linear functions or low degree polynomials over an appropriate group structure, one concern was that these might be the only examples of functions that have self-testers and self-correctors. Using the new robustness results, we show that self-testers and self-correctors can be found for numerical functions that previously did not have self-testers and self-correctors. The techniques used to derive our results seem amenable to further generalization and may apply to an even wider variety of numerical functions.

We concentrate on self-testers which operate by finding properties (such as functional equations) that should be satisfied by any correct program and then *testing that the program satisfies the properties for randomly chosen inputs*. In this work, we study the characteristics of the properties that make them usable for testing. Properties that can be tested more efficiently than computing the function f are particularly interesting for constructing good tests for programs.

The idea of testing programs by verifying that programs satisfy properties known to be satisfied by the functions being computed is not new to the self-testing/correcting approach. For example, matrix multiplication routines have been tested by verifying that the outputs satisfy the distributive property [39]. The work of Cody and Stoltz [25] proposes the use of Taylor series in order to test programs for exponential integrals. These techniques apply to Bessel functions and Dawson's integral. The work of Vainstein [42], [43], [44], [45] suggests the use of *polynomial checks* for testing and correcting programs. In the language previously defined, polynomial checks are those functional equations for which the function F is a polynomial and the neighborhoods are ordered sets of the form $(x, x+a_1, x+a_2, \ldots, x+a_k)$ for fixed constants $a_1, \ldots, a_k$. These functional equations can be used to test functions that are algebraic functions of trigonometric functions. The work of Cody [24] suggests the following test for programs computing the real gamma function over the reals:

$$\text{Pick random } x \text{ and verify that } P(2x) = (2\pi)^{-1/2} 2^{2x-1/2} P(x) P(x+\tfrac{1}{2}).$$

In all of the above cases, it is clear that any correct program for the function must pass these recommended tests. However, none of the works mentioned in this paragraph give any formal evidence that programs that pass these tests should be usable. On the contrary, it is easy to come up with examples of programs that pass the above tests but do not compute the correct function on a large fraction of inputs.

Still, it has been shown that in many cases using properties to test programs is mathematically justified (cf. [19], [37], [31], [38]). Essentially one can show that some of these tests can be used in conjunction with other simple tests in order to determine that a program is correct on most inputs. In order to show that such tests work, the main technique used has been to partition the problem into three tasks: first, find properties that characterize a family of functions, $\mathcal{F}$, containing the function f. For example, one can find functional equations satisfied by specific classes of finite degree rational functions of $x, e^x, \sin x$ using the results of [43], [16]. Second, show that these properties are robust, so that it is possible to efficiently test whether the program is computing a function that is close to some function in $\mathcal{F}$. We call this task *property testing*. Third, find other efficient tests which allow the user to determine whether or not the program is computing the correct function within $\mathcal{F}$. We call this latter task *equality testing*. Equality testing can often be done much more efficiently once it is known that the program is essentially computing some member of $\mathcal{F}$. For example, the function $f(x) = x \bmod R$ is uniquely specified by the properties that (1) f is linear, i.e., $\forall x, y \quad f(x) + f(y) \equiv f(x + y) \bmod R$, (2) f has slope 1, i.e., $\forall x \quad f(x)+1 \equiv f(x+1) \bmod R$. Using the robustness of linearity, if (1) is satisfied for *most* x, y (greater than a $\frac{7}{9}$ fraction), then there is some function $g(x) = cx \bmod R$ such that $f(x) = g(x)$ for most x. If in addition (2) is satisfied for most x then $f(x) = x \bmod R$ for most x. (Note that if R is considered to be part of the input, then it is not enough to test only that property (2) is satisfied for any constant fraction of the $x \in [0..R-1]$.) Thus, it is only necessary to check that the program satisfies the given properties at a relatively small number (in this case, a constant independent of $|x|, |R|$) of randomly selected inputs in order to guarantee that the program usually computes the correct values. This paper concentrates on the task of property testing.

It is shown in [19] that self-correctors exist for any function that is random self-reducible,[3] since if the program is known to be correct on most inputs, then the correct value of the function at any particular input x can be inferred, even though the program may be incorrect on input x. In particular, any function satisfying the linearity property is random self-reducible [19]. On a related note, the use of polynomial checks (or the functional equations that are defined by the polynomial checks) for the correction of programs with few errors is suggested in [43], and Blum et al. [16] build on the work of [43] to give self-correctors for the same functions. Here we observe that efficient self-correctors exist for functions satisfying any one of a *class* of functional equations, namely, those of the general form $\forall x, y \quad F[f(x - y), f(x + y), f(x), f(y)] = 0$, where F is an algebraic function that has the property that given three of $f(x-y), f(x+y), f(x), f(y)$, F can be used to efficiently solve for the remaining one. A similar result was obtained independently by Blum et al. [16], where self-correcting using functional equations is studied in much greater depth.

Organization of paper. In section 2 we present the formal definitions of exact and robust characterizations from [38]. In section 3 we investigate certain general properties of functional equations that influence whether they are robust. In section 4 we present technical theorems showing conditions under which the general functional equation $F[f(x - y), f(x + y), f(x), f(y)] = 0$ is robust on domains that are finite

[3] f is random self-reducible if f can be computed at any particular input x via $f(x) = G[f(y_1), \ldots, f(y_k), y_1, \ldots, y_k]$, where G can be computed asymptotically faster than f and the y_i's are uniformly distributed, although not necessarily independent [19]. This notion of random self-reducibility is somewhat different than other definitions given by [20], [1], [30], where the requirement on G is that it be computable in polynomial time.

groups. In section 5 we present the self-testers and self-correctors based on the general form of the functional equation $\forall x, y \;\; F[f(x-y), f(x+y), f(x), f(y)] = 0$. In section 6 we show how to convert the self-testers and self-correctors shown for finite groups into self-testers and self-correctors that apply to functions over rational domains. The completely specified self-tester and self-corrector for the particular example of the cosh function is described in section 6.3. In section 7 we discuss our conclusions and directions for further research.

2. Functional equations and characterizations. In this section, we give the definitions of functional equations and exact and robust characterizations. We also show a relationship between functional equations and probabilistically checkable proof systems.

2.1. Domains and ranges. Throughout this paper, we focus on the following three kinds of domains: The first are finite subsets of the rationals of the form $\mathcal{D}_{n,s} = \{\frac{i}{s} \big| \; |i| \leq n\}$, where n, s are integers. These domains are not necessarily closed under addition and multiplication. This class includes domains that can be internally represented in a computer, corresponding to fixed point arithmetic, which have been used in previous work on self-testing and self-correcting [31], [37]. The second type of domain that we are interested in are finite groups. Even for functions that are not defined over finite group domains, it is much simpler to first reason about the functional equations that they satisfy over finite group domains since they are closed under addition and then to use the techniques of [31], [37] (described in section 6) for converting results on finite group domains into results on rational domains. The third class of domains are of use when studying periodic functions: $\mathcal{D}_s^b = \{i \cdot s \big| \; i \in \mathcal{Z}\}$, where b/s is an integer, and addition and multiplication is performed mod b. For example, $\mathcal{D}_{2\pi/10}^{2\pi} = \{0, 2\pi/10, 4\pi/10, 6\pi/10, \ldots, 18\pi/10\}$. Note that any function $f : \mathcal{D}_s^b \to \mathcal{R}$ corresponds to a function $\hat{f} : \mathcal{Z}_{b/s} \to \mathcal{R}$ by $\hat{f}(i) = f(i \cdot s \bmod b)$. Thus results on the finite group domains can be immediately applied to this third class of domains. The range of the functions considered can in general be arbitrary. If not specified, the range is assumed to be the reals.

In Tables 1.1 and 1.2, solutions to functional equations over the reals are given. It may happen that the functional equation over the reals characterizes a family of functions that is a proper subset of the functions characterized by the same functional equation over $\mathcal{D}_{p,s}$. In section 5.2 we show that this does not limit the ability to construct self-testers for programs for these functions, due to the equality testing performed by self-testers.

2.2. Functional equations. In the text by Aczél [3, p. 1] , functional equations are defined by first defining a *term*.

DEFINITION 2.1 (term—[3, p. 1]).

1. *The independent variables $x_1, \ldots, x_k$ are terms.*
2. *Given that $A_1, \ldots, A_m$ are terms and that H is a function of m variables, then $H(A_1, \ldots, A_m)$ is also a term.*
3. *There are no other terms.*

DEFINITION 2.2 (functional equation—[3, p. 2]). *A* functional equation *is an equation $A_1 = A_2$ between two terms A_1, A_2 which contains k independent variables $x_1, \ldots, x_k$ and $n \geq 1$ unknown functions $H_1, \ldots, H_n$ of $j_1, \ldots, j_n$ variables, respectively, as well as a finite number of known functions.*

The known functions used in [3] include addition; subtraction; division; multiplication; and exponentiation, trigonometric, and hyperbolic functions. In this paper,

we will also include all functions computable by a Turing machine. Later in [3, p. 3] it is also noted that the functional equation must be identically satisfied for certain values of the variables $(x_1, \ldots, x_k)$ figuring in them, called the domain. (We use the term neighborhood set in this paper.) A *particular solution* of a functional equation is a function that satisfies the equation in the given domain (neighborhood set). The *general solution* is the set of all solutions belonging to the *class of admissible functions*, which can, for example, be defined by the analytic properties (measurability, differentiability, continuity, boundedness), other properties such as computability by a polynomial time Turing machine, by initial and boundary conditions and/or by conditions given in the form of another functional equation.

2.3. Exact and robust characterizations. We now present the definitions of characterizations and robust characterizations given by [38]. $\mathcal{D}$ is used to represent a finite domain. We consider families of functions $\mathcal{F}$ where $f \in \mathcal{F}$ maps elements from domain $\mathcal{D}$ to range $\mathcal{R}$ (we use $\mathcal{R}$ to denote the range of a function and $\Re$ to denote the set of real numbers). $\mathcal{T}$ is a range containing 0. We illustrate these definitions using the example of linear functions. Here $\mathcal{D} = \mathcal{R} = \mathcal{T} = \mathcal{Z}_p$ and the family of linear functions is $\{f_a | a \in \mathcal{Z}_p \text{ where } f_a(x) = a \cdot x\}$.

DEFINITION 2.3 (neighborhoods—[38]). *$N_{\mathcal{D}}$ is a k-*local neighborhood *if it is an ordered tuple of (not necessarily distinct) k points $(x_1, \ldots, x_k)$ from $\mathcal{D}^k$. A k-local collection of neighborhoods $\mathcal{N}_{\mathcal{D}}$ is a (multi)set of k-local neighborhoods. When $\mathcal{D}$ is understood from the context, we drop it from the subscript.*

DEFINITION 2.4 (properties—[38]). *$\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}$ is a k-*local property *if it is a function from $\mathcal{R}^k \times \mathcal{D}^k$ to $\mathcal{T}$. We say that a function $f : \mathcal{D} \to \mathcal{R}$* satisfies a property *$\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}$ over a neighborhood $N_{\mathcal{D}} = (x_1, \ldots, x_k)$ if $\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}(f(x_1), \ldots, f(x_k), x_1, \ldots, x_k) = 0$.[4] When $\mathcal{D}, \mathcal{R}, \mathcal{T}$ are understood from the context, we drop them as subscripts.*

DEFINITION 2.5 (exact characterizations—[38]). *We say that $(\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}, \mathcal{N}_{\mathcal{D}})$ is an* exact characterization *of a family $\mathcal{F}$ of functions if a function $f : D \to R$ satisfies $\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}$ over all neighborhoods $N_{\mathcal{D}} \in \mathcal{N}_{\mathcal{D}}$ exactly when $f \in \mathcal{F}$. The characterization is k-local if the property $\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}$ and the collection $\mathcal{N}_{\mathcal{D}}$ is k-local. When $\mathcal{D}, \mathcal{R}, \mathcal{T}$ are understood from the context, we drop them as subscripts.*

In our example, $\mathcal{R} = \mathcal{T} = \mathcal{D} = \mathcal{D}' = \mathcal{Z}_p$. The collection of neighborhoods is $\mathcal{N} = \{(x, y, x + y) | x, y \in \mathcal{Z}_p\}$. The property $\mathcal{P}$ which for $\{x_1, x_2, x_3\}$ computes $\mathcal{P}(f(x_1), f(x_2), f(x_3), x_1, x_2, x_3) = f(x_1) + f(x_2) - f(x_3)$ is 3-local. $(\mathcal{P}, \mathcal{N})$ is a 3-local characterization of the family of the linear functions $\{f | f(x) = c \cdot x, c \in \mathcal{Z}_p\}$.

DEFINITION 2.6 (robust characterizations—[38]). *Let $\mathcal{D}' \subseteq \mathcal{D}$. Let $\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}$ be a property over a collection of neighborhoods $\mathcal{N}_{\mathcal{D}}$; let $\mathcal{F}$ be such that $(\mathcal{D}, \mathcal{R}, \mathcal{T}, \mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}, \mathcal{N}_{\mathcal{D}})$ is an $(\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}, \mathcal{N}_{\mathcal{D}})$ is an exact characterization of $\mathcal{F}$. We say that the characterization $\mathcal{A} \equiv (\mathcal{D}', \mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}, \mathcal{N}_{\mathcal{D}})$ is an* (ϵ, δ)-robust characterization *of $\mathcal{F}$ in $\mathcal{G}$ if whenever a function $f \in \mathcal{G}$ satisfies $\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}$ on all but δ fraction of the neighborhoods in $\mathcal{N}_{\mathcal{D}}$, it is ϵ-close on domain $\mathcal{D}'$ to some function $g \in \mathcal{F}$.[5] When $\mathcal{D}, \mathcal{D}', \mathcal{R}, \mathcal{T}$ are understood from the context, we drop those parameters.*

We remark that in order for a robust characterization to be useful, membership

[4] This is a slight modification of the definition in [38], where the function f satisfies $\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}$ if $\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}(f(x_1), \ldots, f(x_k), x_1, \ldots, x_k) = 1$ and the range of the function $\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}$ is $\{0, 1\}$ instead of $\mathcal{T}$.

[5] It is convenient for our results in this paper to define $\mathcal{N}_{\mathcal{D}}$ as a multiset and to define robust characterizations in terms of picking neighborhood sets from the uniform distribution on $\mathcal{N}_{\mathcal{D}}$. Alternatively, one can define robust characterizations in terms of a distribution on ordered sets, where $\mathcal{N}_{\mathcal{D}}$ would correspond to the support of the distribution.

in $\mathcal{G}$ should be efficient to test, choosing a random neighborhood in $\mathcal{N}_{\mathcal{D}}$ should be efficient, and $\mathcal{D}'$ should be a fairly large subset of $\mathcal{D}$. All of our results have these properties. In most examples, $\mathcal{G}$ will be the set of all functions; however, we will see examples in which it is useful to have $\mathcal{G}$ be a smaller, efficiently recognizable, set of functions.

To continue with the example of linear functions, a theorem of [19] can be used to say that for any finite group G and any $\delta < \frac{2}{9}$, $(\mathcal{P}_G, \mathcal{N}_G)$ is a $(2\delta, \delta)$-robust characterization of the linear functions mapping G to G.

In order to test if f is close to some member of $\mathcal{F}$, one would need to sample at least $\frac{1}{\delta}$ of the neighborhoods in $\mathcal{N}$ and test if $\mathcal{P}$ holds on these neighborhoods. Thus, $\frac{1}{\delta}$ is referred to as the *efficiency* of the characterization.

We now define what it means for a characterization to be robust over a class.

DEFINITION 2.7. *Let $\mathcal{S} = \{(\mathcal{A}_1, \mathcal{F}_1, \mathcal{G}_1), (\mathcal{A}_2, \mathcal{F}_2, \mathcal{G}_2), \ldots\}$ be such that for all i, $\mathcal{A}_i = (\mathcal{D}_i', \mathcal{P}_{\mathcal{D}_i,\mathcal{R}_i,\mathcal{T}_i}, \mathcal{N}_{\mathcal{D}_i})$ and $(\mathcal{P}_{\mathcal{D}_i,\mathcal{R}_i,\mathcal{T}_i}, \mathcal{N}_{\mathcal{D}_i})$ is an exact characterization of $\mathcal{F}_i$. We say that $(\mathcal{P}, \mathcal{N})$ is* robust over the family $\mathcal{S}$ *if*

1. *there is a function $\mathcal{N}$ which takes as input i and returns a Turing machine M such that M on input a random string chooses a random member $N \in \mathcal{N}_{\mathcal{D}_i}$;*
2. *there is a function $\mathcal{P}$ which takes as input i and returns a Turing machine that on input $N \in \mathcal{N}_{\mathcal{D}_i}$ computes $\mathcal{P}_{\mathcal{D}_i,\mathcal{R}_i,\mathcal{T}_i}(N)$;*
3. *for all $\epsilon < 1$ there is a $\delta < 1$ such that for all i, $\mathcal{A}_i$ is an (ϵ, δ)-robust characterization.*[6]

In order for a robust characterization over a class to be useful, the functions $(\mathcal{P}, \mathcal{N})$ should have a uniform and concise description. In particular, functional equations have a natural interpretation as a concise description of robust characterizations over a class. In this paper, we consider variations of the following two basic types of classes.

In the first type of class, we capture the property that a functional equation is (ϵ, δ)-robust over all domains that have a certain structure, such as finite groups. The functional equation $\mathcal{P}, \mathcal{N}$ is described with a generic group or field operation. Let $\mathcal{D}_i = \mathcal{D}_i' = \mathcal{R}_i = \mathcal{T}_i = G_i$, where G_i is the ith group (for an arbitrary ordering of the groups) with group operator $+_{G_i}$. $\mathcal{P}_{\mathcal{D}_i,\mathcal{R}_i,\mathcal{T}_i}$ and $\mathcal{N}_{\mathcal{D}_i}$ are then the functions obtained by using the group operator $+_{G_i}$. In our linearity example, $\mathcal{P}(i) = \mathcal{P}_{\mathcal{D}_i,\mathcal{R}_i,\mathcal{T}_i}$ is the function $\mathcal{P}(f(x_1), f(x_2), f(x_3), x_1, x_2, x_3) = f(x_1) +_{G_i} f(x_2) -_{G_i} f(x_3)$. The collection of neighborhoods $\mathcal{N} = \{(x, y, x +_{G_i} y) | x, y \in G_i\}$.

In the second type of class, we concentrate on finite subsets of various sizes of an infinite group. The functional equation is defined over a large, possibly infinite domain such as the rationals. However, the robust characterization is defined over a finite subset of the domain. Let $\mathcal{D}_i = \mathcal{D}_{n_i,s}$, $\mathcal{D}_i' = \mathcal{D}_{i,s}$ for $n_i \geq i$ (the exact value of n_i is determined by the robust characterization) and $\mathcal{R} = \mathcal{T} = \Re$. $\mathcal{P}, \mathcal{N}$ always return the same function which maps the rationals to the reals. In our linearity example, $\mathcal{P}$ is the function $\mathcal{P}(f(x_1), f(x_2), f(x_3), x_1, x_2, x_3) = f(x_1) + f(x_2) - f(x_3)$ and $\mathcal{N}$ is a carefully chosen subset of $\{(x, y, x + y) | x \in \mathcal{D}_{n_i,s}, y \in \mathcal{D}_{n_i,s}\}$ (see section 6). Operations $+, -$ are the usual group operations over the reals.

Our results specify the class of domains and ranges over which the functional equation is robust. When $\mathcal{S}$ is understood from the context, we say that $(\mathcal{P}, \mathcal{N})$ is robust.

[6] The only interesting case is when the size of $\mathcal{S}$ is infinite, since otherwise there are always constants ϵ, δ such that all characterizations in the collection are (ϵ, δ)-robust.

2.4. Robustness and probabilistically checkable proofs. A language L in NP has a *probabilistically checkable proof system* if there is a probabilistic polynomial time Turing machine V (the verifier) that has read access to a source of random strings R and to a proof P for the membership of x in L, such that (1) if $x \in L$, there exists a proof P of membership in the language such that V accepts P with probability 1 (where the probability is over the random strings R) and (2) if x is not in L, for all proofs P', V accepts proof P' with probability at most $\frac{1}{4}$ [29]. The linearity test and tests for low total degree polynomial functions that are given in [19], [37], [6] have been used to construct probabilistically checkable proof systems in the recent results of [6], [14], [12] (tests that functions are low degree in each variable are given and used in [9], [29], [7]). Much recent research has been devoted to expanding the range of the robustness parameter δ for which these tests work, as it directly influences the strength of results showing that it is computationally difficult to approximate certain NP-complete problems [36], [11], [12].

Conversely, Sudan [41] has noted that the property of being a probabilistically checkable proof can actually be viewed as an example of a robust functional equation (where the definition of F is generalized to include all polynomial time circuits): in the work of Arora, Lund, Motwani, Sudan, and Szegedy [6], each probabilistically checkable proof P can be viewed as a truth table of a function. If P is n bits long, then P can be thought of as the function $P : [1 \dots n] \to \{0,1\}$; i.e., $P(i)$ is the ith bit of the proof. The protocol followed by V is to choose an r-bit random string y, perform a computation in order to determine a constant number of locations $\sigma_1(y), \dots, \sigma_k(y)$, query the proof at those locations, and then perform another computation on input $(y, P(\sigma_1(y)), \dots, P(\sigma_k(y)))$ in order to determine whether to accept or reject the proof. More formally, let $\mathcal{N} = \{(\sigma_1(y), \dots, \sigma_k(y)) | y \in \{0,1\}^r\}$. The verifier's choice of a random string in $\{0,1\}^r$ determines a choice of a neighborhood $(y_1, \dots, y_k)$ from $\mathcal{N}$ by the computation $y_i = \sigma_i(y)$. The verifier then tests whether a relationship $\forall (y_1, \dots, y_k) \in \mathcal{N},\ F[P(y_1), P(y_2), \dots, P(y_k), y] = 0$ is satisfied, where F is computable by a polynomial time Turing machine and describes the computation of the verifier that determines whether to accept or reject the proof. In [6] it is shown that one can construct an $(F, \mathcal{N})$ that characterizes the set of valid proofs; i.e., valid proofs are exactly those bit strings P for which $F[P(y), P(\sigma_1(y)), \dots, P(\sigma_k(y)), y] = 0$ is satisfied for all random strings y. Furthermore only proofs that are close (equal on most bits) to some valid probabilistically checkable proof are passed with probability $\geq 3/4$.

3. Characterizing robust functional equations. We turn to the general question of how to distinguish functional equations that are robust from those that are not in order to arrive at a better understanding of what makes a functional equation robust. It turns out that the structure of the neighborhood set is a very important determining factor to whether or not the functional equation is robust. To illustrate, the following are three characterizations of the lines $\mathcal{F} = \{f | f : Z_p \to Z_p, f(x) = ax + b \text{ for } a, b \in \mathcal{Z}_p\}$ (this family of lines is different from the one discussed previously) and over the class $\mathcal{S}$ in which $\mathcal{R}_i = \mathcal{T}_i = \mathcal{D}_i = \mathcal{D}'_i = \mathcal{Z}_{p_i}$ where p_i is the ith prime:

1. $\forall x_1, x_2, x_3 \in Z_{p_i}, \frac{f(x_1)-f(x_2)}{x_1-x_2} = \frac{f(x_2)-f(x_3)}{x_2-x_3}$.
2. $\forall x_1, x_2 \in Z_{p_i},\ f(x_1) - 2f(x_2) + f(2x_2 - x_1) = 0$ or, equivalently, $\mathcal{N}_{\mathcal{D}} = \{(x_1, x_2, 2x_2 - x_1) | x_1, x_2 \in Z_{p_i}\}$ and $\forall (x_1, x_2, x_3) \in \mathcal{N}_{\mathcal{D}}, f(x_1) - 2f(x_2) + f(x_3) = 0$.
3. $\forall x_1 \in Z_{p_i},\ f(x_1) - 2f(x_1+1) + f(x_1+2) = 0$ or, equivalently, $\mathcal{N}_{\mathcal{D}} = \{(x_1, x_1+$

$1, x_1 + 2)|x_1 \in Z_{p_i}\}$ and $\forall (x_1, x_2, x_3) \in \mathcal{N}_{\mathcal{D}}, f(x_1) - 2f(x_2) + f(x_3) = 0$.

In all three characterizations, the property is the same (although simplified in the latter two characterizations because of the specially chosen neighborhoods) and ensures that the points $(x_1, f(x_1)), (x_2, f(x_2)), (x_3, f(x_3))$ all lie on a single line. The only difference in the three characterizations is the collection of neighborhoods over which it is defined. However, the choice of neighborhoods heavily influences the robustness of the characterizations. A simple counting argument (similar to the one described later in section 3.2) shows that the first property is (δ, δ)-robust for all $\delta < 1$. The second property is $(2\delta, \delta)$-robust for $\delta < \frac{1}{1082}$ [38]. It is easy to see that for all ϵ, the third property is not (ϵ, δ)-robust over $\mathcal{S}$ for any constant δ. Thus the richness of the neighborhood set influences the robustness as well as the complexity of computing $\mathcal{P}_{\mathcal{D},\mathcal{R},\mathcal{T}}$. Another interesting quantity related to the neighborhood set is the number of random bits required to choose a random element of the neighborhood set (or the logarithm of the size of the neighborhood set). Reducing this quantity, even by a constant factor, while not significantly affecting the range of δ, ϵ achievable for maintaining a robust characterization (and thus not significantly reducing the efficiency of the characterization), has been useful for constructing more efficient probabilistically checkable proofs [14], [12].

We begin by investigating two extreme types of functional equations:

1. A *k-minimal neighborhood set* is one in which $\mathcal{N}_{\mathcal{D}}$ is described by $k-1$ functions $\sigma_1(x), \ldots, \sigma_{k-1}(x)$, where the σ_i's are arbitrary functions mapping $\mathcal{D}$ to $\mathcal{D}$. $\mathcal{N}_{\mathcal{D}}$ is of the form $\{(x, \sigma_1(x), \ldots, \sigma_{k-1}(x))|x \in \mathcal{D}\}$. Since once the first element of the neighborhood is chosen, the other elements are uniquely determined, the cardinality of the neighborhood set is at most $|\mathcal{D}|$. $\mathcal{N}_{\mathcal{D}}$ in the third example uses a 3-minimal neighborhood set. A *minimal functional equation* is one in which the neighborhood set is k-minimal for some constant k.
2. A *k-total neighborhood set* is one in which $\mathcal{N}_{\mathcal{D}} = \mathcal{D}^k$, relating the function at each input x to function values at *all* subsets of $k-1$ other inputs. $\mathcal{N}_{\mathcal{D}}$ in the first example uses a 3-total neighborhood set. A *total functional equation* is one in which the neighborhood set is k-total for some constant k.

We isolate a key combinatorial property of the neighborhood sets of functional equations and show that having this property is a necessary condition for robustness. We apply this combinatorial property to show a general condition under which functional equations with minimal neighborhood sets are not robust. As we will see later, this result implies that certain methods of testing programs used in practice are related to this class and are therefore provably faulty. For example, our techniques apply to the functional equation that is used to test the real gamma function [24, p. 5]. On the other hand, we mention an example, given by Sudan [41], of a minimal equation that is robust. We then show conditions under which k-total equations are always robust.

In the following, we assume that $\mathcal{D} = \mathcal{D}'$.

3.1. Minimal functional equations. One might conjecture that minimal equations cannot be robust, since for most inputs x, the function value at x is related to the function values at very few other inputs. We show a class of minimal equations that are provably not robust. We then describe an example of a minimal robust functional equation.

A combinatorial property of robustness. We first define a graph which captures much of the information in the neighborhood set $\mathcal{N}_{\mathcal{D}} = \{(x, \sigma_1(x, \bar{y}), \ldots,$

$\sigma_k(x,\bar{y}))|x \in \mathcal{D}, \bar{y} \in \mathcal{D}^k\}$: given the functional equation $\forall x \in \mathcal{D}, \bar{y} \in \mathcal{D}^k$, $F[f(x), f(\sigma_1(x,\bar{y})), \ldots, f(\sigma_k(x,\bar{y}))] = 0$, for $\sigma_i : \mathcal{D} \times \mathcal{D}^k \to \mathcal{D}$, we define the undirected multigraph $G_{\mathcal{N}_\mathcal{D}} = (V, E)$, where the vertices correspond to elements of $\mathcal{D}$ and the edges are $E = \{(x, \sigma_j(x,\bar{y}))|x \in \mathcal{D}, \bar{y} \in \mathcal{D}^k, 1 \le j \le k\}$ (there may be more than one edge between u and v if there is more than one $i, \bar{y}$ such that $\sigma_i(u,\bar{y}) = v$).

For example, the graph of $\forall x \in \mathcal{Z},\ f(x)+1-f(x+1) = 0$ corresponds to a path, and the graph of $\forall x, y,\ f(x) + f(y) - f(x+y) = 0$ corresponds to a complete graph with two edges between every pair of nodes.

The following result applies to functional equations defining classes of functions that can be thought of as codewords with very large ($>> \frac{1}{2}$) distance.

DEFINITION 3.1. *An α-separated function family $\mathcal{F}$ over domain $\mathcal{D}$ is one for which $|\mathcal{F}| \ge 2$ and $\forall f_i, f_j \in \mathcal{F},\ Pr_{x\in\mathcal{D}}[f_i(x) = f_j(x)] \le \alpha$.*

For example, for all of the functional equations mentioned in Tables 1.1, 1.2, for all α, there are integers n, s such that the functional equations over the domain $\mathcal{D}_{n,s}$ characterize α-separated function families.

The following theorem shows a relationship between the connectivity of $G_{\mathcal{N}_\mathcal{D}}$ and the robustness of the functional equation $(F, \mathcal{N}_\mathcal{D})$: if $(F, \mathcal{N}_\mathcal{D})$ is (ϵ, δ)-robust, then more than δ/k fraction of the edges in the graph $G_\mathcal{N}$ must be removed in order to separate $G_\mathcal{N}$ into two "large" components, each of size $\ge (\epsilon + \alpha)|V|$.

THEOREM 3.2. *Let $\mathcal{N} = \{(x, \sigma_1(x,\bar{y}), \ldots, \sigma_k(x,\bar{y}))|x \in \mathcal{D}, \bar{y} \in \mathcal{D}^k\}$. Suppose that $\mathcal{F}$, characterized by $(F, \mathcal{N})$, is an α-separated function family. If $G_\mathcal{N}$ has a set of edges E' such that (1) $|E'| \le \frac{\delta}{k}|E|$ and (2) removing E' separates the vertices of $G_\mathcal{N}$ into two components, each of size $\ge (\epsilon+\alpha)|V|$, then $(F, \mathcal{N})$ is not an (ϵ, δ)-robust characterization.*

Proof. Suppose E' separates G_F into sets A, B. Consider the function h which labels vertices in A according to f_1 and vertices in B according to f_2 for some $f_1, f_2 \in \mathcal{F}$. Since $\mathcal{F}$ is α-separated, we have that $\forall f_i \in \mathcal{F}, Pr_{x\in\mathcal{D}}[f_i(x) \ne h(x)] \ge \epsilon$. However, only tests using edges that cross the cut will fail. Since $x, \bar{y}$ are chosen uniformly, the edges are also chosen uniformly. Thus tests will fail with probability $\le \delta$. □

Application to minimal functional equations. It is easy to see that for any minimal equation F of the form $F[f(x), f(\sigma(x))] = 0$, G_F can be separated into two large components by removing very few edges (by Theorem 3.2), and thus for all classes $\mathcal{S}$ in which the domain size is not bounded, the functional equation $(F, \mathcal{N})$ for $\mathcal{N} = \{(x, \sigma(x))|x \in \mathcal{D}\}$ over $\mathcal{S}$ is not robust.

It was shown by Klawe [32] that for any given ϵ, any graph on n nodes whose edges are defined by a constant number of linear functions has a cut containing $o(n)$ edges which separates the graph into two large portions, each containing an ϵ fraction of the nodes. The following corollary applies to functional equations relating points x to points that are linear functions of x.

COROLLARY 3.3. *Given n, s, let $\mathcal{D}_{n,s} = \mathcal{D} = \mathcal{D}'$. $\mathcal{R} = \mathcal{T} = \Re$. Let $\sigma_1, \ldots, \sigma_k$ be any family of linear functions over the rationals of the form $\sigma_i(x) = ax + b$, where a, b are rational, and let $\mathcal{F}$ be an α-separated function family satisfying the equation $F[f(x), f(\sigma_1(x)), f(\sigma_2(x)), \ldots, f(\sigma_k(x))] = 0$. Then there exists a constant $0 < \epsilon < 1$ such that $(F, \mathcal{N})$ is not (ϵ, δ)-robust for any constant δ.*

Thus, if $\mathcal{S}$ is a class such that $\mathcal{D}_{i,s} = \mathcal{D}_i = \mathcal{D}_i'$, then $\mathcal{R}_i = \mathcal{T}_i = \Re$, and $\sigma_1, \ldots, \sigma_k$ are any family of linear functions over the rationals of the form $\sigma_i(x) = ax + b$, where a, b are rational, the functional equation $(F, \mathcal{N})$ for $\mathcal{N} = \{(x, \sigma_1(x), \ldots, \sigma_k(x))|x \in \mathcal{D}\}$ over $\mathcal{S}$ is not robust. A similar result applies for linear functions over finite groups.

This corollary shows that many tests that are used in practice to test programs should be used with more care. For example, in the functional equation

$$\forall x,\ f(2x) - (2\pi)^{-1/2} 2^{2x-1/2} f(x) f(x+1/2) = 0$$

used for testing the real gamma function by [24], all of the σ_i's are linear functions ($\sigma_1(x) = 2x, \sigma_2(x) = x + \frac{1}{2}$). Thus the corollary implies that there exist programs which are very different from any solution to this functional equation, yet pass the test most of the time. The direct use of polynomial checks suggested in [43] also yields functional equations which are not robust due to Corollary 3.3; however, the work of [16] shows how to transform the polynomial checks into more general functional equations for which the negative results in this section do not apply.

A robust minimal functional equation. Previous examples of robust functional equations have always been usable for self-correction as well. This might lead one to think that usability for self-correction might be another necessary condition for robustness. However, this may note be the case: it is not known how to use minimal functional equations for self-correction. Even so, there are minimal functional equations that are robust and can therefore be used to self-test. We describe an example of a minimal functional equation that is robust. The following example was given by Sudan [41].

We say that a graph $G(V, E)$ is an α-expander if for all $S \subseteq V, |S| \leq |V|/2$, the set of nodes that are neighbors of S (not including nodes in S), is of size $\geq \alpha|S|$. Fix constants d and α. Let $G_i(V, E)$ be any degree d α-expander on i nodes, such that the vertices are labelled by elements of the domain D. Let the functional equation be $\forall (u, v) \in E,\ f(u) - f(v) = 0$. Since G_i is connected, the only functions which are solutions to this functional equation are the constant functions. Assume that the functional equation is satisfied for most $(u, v) \in E$. Suppose one deletes all edges for which the functional equation does *not* hold. Since G_i is an expander, there must exist a large connected component in G_i (containing at least a constant fraction of the nodes), even after deleting the edges. The large connected component will correspond to elements of the domain that agree with a single constant function. Let $\mathcal{S}$ be the class corresponding to the above functional equation on $G_0, G_1, \ldots$. Then for all ϵ, there is a δ such that the above functional equation is (ϵ, δ)-robust on $\mathcal{S}$.

3.2. Total functional equations. On the other end of the spectrum, we consider a class of functional equations where there are no restrictions on the way inputs are related, and we show that if some technical conditions are satisfied, then they are always robust.

Given $\mathcal{D}$ and $\mathcal{R}$, let $F[f(x), f(y_1), \ldots, f(y_{k-1}), x, y_1, \ldots, y_{k-1}] = 0,\ \forall x, y_1, \ldots, y_k$ be a k-total functional equation characterizing functions $f : \mathcal{D} \to \mathcal{R}$. Assume further that F can be solved for $f(x)$, namely, $f(x) = G[f(y_1), \ldots,\ f(y_{k-1}),\ x, y_1, \ldots, y_{k-1}]$ $\forall y_1, \ldots, y_{k-1}$ (because of the totality of the equation, F and G depend on $x, y_1, \ldots, y_{k-1}$ as well as the function values at those points). We say that the solution $\mathcal{F}$ to equation F is $(k-1)$-*complete* if $\forall((y_1, w_1), \ldots, (y_{k-1}, w_{k-1})) \in (\mathcal{D}, \mathcal{R})^{k-1},\ \exists f \in \mathcal{F}$ such that $f(y_i) = w_i$ for all $1 \leq i \leq k-1$. An example of a $(k-1)$-complete function family is the family of degree $(k-1)$ polynomials. The following theorem, which says that k-total functional equations that characterize $k-1$-complete function families are necessarily robust, can be viewed as a generalization of a known theorem for degree $(k-1)$ univariate polynomials (cf. [40]).

THEOREM 3.4. *Let $\mathcal{N} = \mathcal{D}^k$. Suppose the k-total functional equation $F[f(x), f(y_1), \ldots, f(y_{k-1}), x, y_1, \ldots, y_{k-1}] \equiv f(x) - G[f(y_1), \ldots, f(y_{k-1}), x,\ y_1, \ldots,\ y_{k-1}] = 0$*

$\forall x, y_1, \ldots, y_{k-1}$ *has a* $k-1$*-complete solution* $\mathcal{F}$*. Then* $(F, \mathcal{N})$ *is* (δ, δ)*-robust* $\forall 0 < \delta < 1$.

Proof. Suppose f satisfies $Pr_{x,\bar{y}}[f(x) = G[f(y_1), \ldots, f(y_{k-1}), x, y_1, \ldots, y_{k-1}]] \geq 1 - \delta$. Then there exists $z_1, \ldots, z_{k-1}$, a particular setting of the y_i's, such that the test works for $\geq 1 - \delta$ of the x's. Let $g(x)$ be the function in $\mathcal{F}$ determined by the values of the program at $z_1, \ldots, z_{k-1}$. Then $Pr_x[g(x) = f(x)] \geq 1 - \delta$. Thus $\forall x, y_1, \ldots, y_{k-1}\ g(x) - G[g(y_1), \ldots, g(y_{k-1}), x, y_1, \ldots, y_{k-1}] = 0$. □

Any function f that satisfies such a total functional equation F can be computed, given the value of the function at any fixed k locations, as efficiently as evaluating the functional equation G. Thus, if F and G have the same complexity, the functional equation is not useful for self-testing, since it does not have the "little-oh property" described in [18]. However, it is possible that more efficient self-testers can be constructed by looking at a smaller, carefully chosen, set of neighborhoods $\mathcal{N}$ and showing that the functional equation is still robust over $\mathcal{N}$. Given $\vec{\sigma} = (\sigma_1, \ldots, \sigma_{k-1})$, such that $\sigma_i : \mathcal{D} \times \mathcal{D}^{k-1} \to \mathcal{D}$, suppose functional equations $F[f(x), f(y_1), \ldots, f(y_{k-1}), x, y_1, \ldots, y_{k-1}] = 0\ \ \forall x, y_1, \ldots, y_{k-1}$, and $F[f(x), f(\sigma_1(x, \bar{z})), \ldots, f(\sigma_{k-1}(x, \bar{z})), x, \sigma_1(x, \bar{z}), \ldots, \sigma_{k-1}(x, \bar{z})] = 0\ \ \forall x, \bar{z}$ both have the same complete solution $\mathcal{F}$. Due to the structure of the σ's, it might be the case that F is easier to compute on those tuples defined by the σ's (for example, efficient polynomial degree tests have been constructed by only performing tests on points that are evenly spaced: $\sigma_i(x, z) = x + iz$ [38]). If F is also robust over random choices of $x, \bar{z}$, then a more efficient tester can be constructed.

We use a bound on the runtime of the program being tested to devise a tester. The works of Blum et al. [17] and Micali [35] also construct checkers based on bounds on the runtime of the program being checked but use very different methods. Let the distribution $\mathcal{V}_{\vec{\sigma}}$ be the distribution defined by picking $x \in \mathcal{D}, \bar{z} \in \mathcal{D}^{k-1}$ randomly, and outputting $(x, \sigma_1(x, \bar{z}), \ldots, \sigma_{k-1}(x, \bar{z}))$. Let $\mathcal{U}$ be the distribution defined by picking $x, y_1, \ldots, y_{k-1} \in \mathcal{D}$ and outputting $(x, y_1, \ldots, y_{k-1})$. If the σ's look "random enough," we have the following theorem showing a sense in which F is robust over random choices of $x, \bar{z}$. This theorem implies that it is enough to test points related by the σ's.

THEOREM 3.5. *Let* $Time(c) \equiv \{f | f$ *is computable on inputs of length* n *in time* $n^c\}$*. Let* c *be an arbitrary constant, and fix* $0 \leq \epsilon \leq 1$ *and* $0 \leq \delta \leq 1$*. Let* E_1 *denote the functional equation* $F[f(x), f(y_1), \ldots, f(y_{k-1}), x, y_1, \ldots, y_{k-1}] = 0\ \forall x \in \mathcal{D}, \bar{y} \in \mathcal{D}^{k-1}$*, and* E_2 *denote the functional equation* $F[f(x), f(\sigma_1(x, \bar{z})), \ldots, f(\sigma_{k-1}(x, \bar{z})), x, \sigma_1(x, \bar{z}), \ldots, \sigma_{k-1}(x, \bar{z})] = 0\ \forall x \in \mathcal{D}, \bar{z} \in \mathcal{D}^{k-1}$*. Assume that*

1. E_1 *and* E_2 *have the same complete solution* $\mathcal{F}$*,*
2. F *can be computed by a circuit of size* $(\log |\mathcal{D}|)^c$*,*
3. *no circuit of size* $\leq (k+1)(\log |\mathcal{D}|)^c$ *can distinguish inputs from* $\mathcal{V}_{\vec{\sigma}}$ *and* $\mathcal{U}$ *with more than* δ *advantage,*
4. E_1 *is* $(\epsilon, 2\delta)$*-robust.*

Then E_2 *can be used to test all programs running in time* $(\log |\mathcal{D}|)^c$*: if* $Pr_{x,R}[F[P(x), P(\sigma_1(x, \bar{z})), \ldots, P(\sigma_{k-1}(x, \bar{z})), x, \sigma_1(x, \bar{z}), \ldots, \sigma_{k-1}(x, \bar{z})] = 0] \geq 1 - \delta$*, and* P *runs in* $\leq (\log |\mathcal{D}|)^c$ *steps, then there is a* $f \in \mathcal{F}$ *such that* $Pr_x[P(x) = f(x)] \geq 1 - \epsilon$*. Thus,* $(\mathcal{F}, \mathcal{D}^k)$ *is an* (ϵ, δ)*-robust characterization of* $\mathcal{F}$ *in* $Time(c)$*.*

Proof. Suppose that there exists some program P running in time $(\log |\mathcal{D}|)^c$ which is wrong at $\geq \epsilon$ inputs in $\mathcal{D}$ but passes the tester with probability $> 1 - \delta$. We use it to construct a program $\mathcal{A}$ of size $\leq (k+1)(\log |\mathcal{D}|)^c$ that can distinguish between outputs from distributions $\mathcal{V}$ and $\mathcal{U}$ with more than δ advantage (which contradicts (3)). $\mathcal{A}$ re-

ceives $w, z_1, \ldots, z_{k-1}$, and tests whether $F[P(x),P(z_1),\ldots,P(z_{k-1}),x,z_1,\ldots,z_{k-1}] = 0$. $\mathcal{A}$ outputs 1 if P passes the test and 0 if P fails. By Theorem 3.4, $Pr_{x,y_1,\ldots,y_{k-1}\in\mathcal{U}}[F[P(x),\ P(y_1),\ \ldots,\ P(y_{k-1}),\ x,\ y_1,\ \ldots,\ y_{k-1}] \neq 0] \geq 2\delta$, and by the assumption, $Pr_{x,y_1,\ldots,y_{k-1}\in\mathcal{V}}[F[P(x),\ P(y_1),\ldots,\ P(y_{k-1}),\ x,\ y_1,\ldots,y_{k-1}] \neq 0] < \delta$. □

4. Robustness of $\forall x, y,\ F[f(x-y), f(x+y), f(x), f(y)] = 0$. We study conditions under which any member of the general class of functional equation $\forall x,\ y\ F[f(x-y), f(x+y), f(x), f(y)] = 0$ is robust. We show that addition theorems $\forall x, y\ f(x+y) = G[f(x), f(y)]$ for which G satisfies $G[a, G[b,c]] = G[G[a,b],c]\ \forall a, b, c$ (which all of our examples satisfy) are robust over the class $\mathcal{S}$, such that the domains in $\mathcal{S}$ are finite groups, and then we give a technique which applies to a number of functional equations that are not addition theorems. Our techniques apply to all functional equations in Table 1.1 as well as the first three functional equations in Table 1.2. We conjecture that all functional equations in this class are robust.

All results can be extended to rational domains of the form $\mathcal{D}_{p,s} = \{\frac{i}{s} \big| |i| \leq p\}$ using standard techniques from [31], [37]. We give an example of such an extension in section 6. Our only assumption on $\mathcal{R}$ in subsection 4.1 is that it is a (possibly infinite) group. In subsection 4.2 we assume that $\mathcal{R}$ is a field.

4.1. Addition theorems. We show that any addition property $\forall x, y\ \ f(x+y) = G[f(x), f(y)]$ is $(2\delta, \delta)$-robust for $\delta < \frac{1}{8}$ and G that satisfies $G[a, G[b,c]] = G[G[a,b],c]\ \forall a, b, c$ (we do not attempt to optimize the relationship between ϵ and δ in our proofs of (ϵ, δ)-robustness—see [13], [26], and [11] for techniques for improving this relationship). Therefore, knowing that $f(x+y) = G[f(x), f(y)]$ holds at more than a $\frac{7}{8}$ fraction of the (x, y) pairs is enough to conclude that f agrees with some solution of the addition theorem G on at least $\frac{3}{4}$ fraction of the inputs. One can verify that G satisfies $G[a, G[b,c]] = G[G[a,b],c]\ \forall a, b, c$ in all of the examples given in Table 1.1.

At the end of this subsection, we consider the requirement that $G[a, G[b,c]] = G[G[a,b],c]\ \forall a, b, c$. We show that that if the domain is a subset of a field, such that rational functions are defined (a function $f(x,y) = p(x,y)/q(x,y)$ where p, q are polynomials), then we can make a general claim for any constant degree rational function G that is based on the number of zeros that a rational function can have. Similar results that apply to algebraic functions can be proven for domains over which algebraic functions are defined (see [46]).

We now show that any additional theorem satisfying $\forall a, b, c\ \ G[a, G[b,c]] = G[G[a,b],c]$ is robust. This proof follows an outline similar to Coppersmith's version of the proof of robustness of the linearity test which is described in [19]. However, the inner manipulations are different. Hence, whereas Coppersmith's proof works for any $\delta \leq \frac{2}{9}$, here we require $\delta \leq \frac{1}{8}$.

THEOREM 4.1. *Let $\mathcal{D} = \mathcal{D}'$ be a finite group and $\mathcal{R} = \mathcal{T}$ a group. Let $\mathcal{N}^{\text{add}} = \{(x, y, x+y) | x, y \in \mathcal{D}\}$. Let G be such that G satisfies $\forall a, b, c \in \mathcal{R}\ \ G[a, G[b,c]] = G[G[a,b],c]$. Let $F(x_1, x_2, x_3) = f(x_3) - G[f(x_1), f(x_2)]$ on neighborhoods $(x_1, x_2, x_3) \in \mathcal{N}^{\text{add}}$. Then for all $\delta < \frac{1}{8}$, $(F, \mathcal{N}^{\text{add}})$ is $(2\delta, \delta)$-robust. Letting $\mathcal{S}$ be a class such that $\mathcal{D}_i = \mathcal{D}_i'$ are finite groups and $\mathcal{R}_i = \mathcal{T}_i$ are groups, then since for all $\epsilon < \frac{1}{4}$, $(F, \mathcal{N}^{\text{add}})$ is $(\epsilon, \frac{\epsilon}{2})$-robust, $(F, \mathcal{N}^{\text{add}})$ is robust over $\mathcal{S}$.*

Proof of Theorem 4.1. To prove the theorem, we will show that if $Pr_{x,y\in_R\mathcal{D}}[f(x+y) = G[f(x), f(y)]] \geq 1 - \delta$, for $\delta < \frac{1}{8}$, then there exists a function g such that (1) $Pr_{x\in_R\mathcal{D}}[f(x) = g(x)] \geq 1 - 2\delta$ and (2) $\forall x, y\ \ g(x+y) = G[g(x), g(y)]$. □

Define $g(x)$ to be $\text{maj}_{z\in\mathcal{D}}\{G(f(x-z), f(z))\}$, where maj of a set is the function

that picks the element occurring most often (choosing arbitrarily in the case of ties). We first show that g is 2δ-close to f.

LEMMA 4.2. *g and f agree on more than $1-2\delta$ fraction of the inputs from $\mathcal{D}$.*

Proof. Consider the set of elements x such that $\Pr_z[f(x) = G[f(x-z), f(z)]] < \frac{1}{2}$. If the fraction of such elements is more than 2δ, then it contradicts the condition that $\Pr_{x,y}[f(x+y) = 0G[f(x),\ f(y)\ ]\ ] \geq 1 - \delta$. For all remaining elements, $f(x) = g(x)$. □

Next we show a sense in which g is well-defined.

LEMMA 4.3. *For all x, $\Pr_z[g(x) = G[f(x-z), f(z)]] \geq 1 - 2\delta$.*

Proof.[7]

$$\begin{aligned}\Pr_{y,z}\big[G[f(x-y), f(y)]\big] &= G[G[f(x-y-z), f(z)], f(y)] \\ &= G[f(x-y-z), G[f(z), f(y)]] \\ &= G[f(x-(y+z)), f(y+z)]] \qquad \geq 1-2\delta.\end{aligned}$$

The first and third equality hold with probability $1-\delta$ by our assumption on f and since $x-y, y, z, x-y-z, z+y$ are all uniformly distributed in $\mathcal{D}$. The second equality always holds since $G[a, G[b,c]] = G[G[a,b],c]\ \forall a,b,c$.

The lemma now follows from the well-known fact that the probability that the same object is drawn twice from a set in two independent trials lower bounds the probability of drawing the most likely object in one trial: suppose the objects are ordered so that p_i is the probability of drawing object i. Without loss of generality $p_1 \geq p_2 \geq \cdots$. Then the probability of drawing the same object twice is $\sum_i p_i^2 \leq \sum_i p_1 p_i = p_1$. □

Finally, we prove that g satisfies the addition theorem everywhere.

LEMMA 4.4. *For all x, y, $g(x+y) = G[g(x), g(y)]$.*

Proof.

$$\begin{aligned}\Pr_{u,v}\big[G[g(x), g(y)]\big] &= G[G[f(u), f(x-u)], G[f(v), f(y-v)]] \\ &= G[f(u), G[f(x-u), G[f(v), f(y-v)]]] \\ &= G[f(u), G[G[f(x-u), f(v)], f(y-v)]] \\ &= G[f(u), G[f(x-u+v), f(y-v)]] \\ &= G[f(u), f(x+y-u)] \\ &= g(x+y)\big] \\ &> 1 - 8\delta > 0.\end{aligned}$$

By Lemma 4.3, the first equality holds with probability $1-4\delta$ and the last equality holds with probability $1-2\delta$. By the assumption on f, the fourth and fifth equalities each hold with probability $1-\delta$. The other equalities always hold, since $G[a, G[b,c]] = G[G[a,b],c]\ \forall a,b,c$. Since the statement is independent of u, v and holds with positive probability, it must hold with probability 1. This also proves Theorem 4.1. □

4.1.1. Addition theorems that satisfy $G[a, G[b,c]] = G[G[a,b],c]\ \forall a,b,c$. If the domain is a large enough subset of a field, such that rational functions are defined (a function $f(x,y) = p(x,y)/q(x,y)$ where p, q are polynomials), and if G is a rational function such that the numerator has bounded degree, then one can show that G satisfies $G[a, G[b,c]] = G[G[a,b],c]\ \forall a,b,c$.

[7] For conciseness, we use a somewhat nonstandard notation: for random variables a, b, c, we reason about the probability that $a = c$ by using an intermediate variable b, using $Pr[a=c] \geq Pr[a=b=c] \geq 1 - Pr[a \neq b] - Pr[b \neq c]$.

THEOREM 4.5. *Let G be a constant degree rational function such that the degree in each variable of the numerator of the rational function $H(a,b,c) \equiv G[a,G[b,c]] - G[G[a,b],c]$ is bounded by N. Assume that G is such that one of the solutions f to the functional equation $\forall x,y \;\; f(x+y) = G[f(x),f(y)]$ takes on at least $(N+1)^3$ values (in particular, $|\mathcal{R}| > (N+1)^3$). Then G satisfies $G[a,G[b,c]] = G[G[a,b],c]$ $\forall a,b,c \in \mathcal{R}$.*

Proof. Since H is a rational function, it will suffice to show that H evaluates to 0 on many inputs and therefore must be identically 0. The inputs for which we show that H evaluates to 0 will correspond to outputs of functions f that satisfy the addition theorem at all x,y.

Given any function f satisfying $\forall x,y \;\; f(x+y) = G[f(x),f(y)]$, since $\mathcal{D}$ is associative we have that $f(x+y+z) = G[f(x),f(y+z)] = G[f(x+y),f(z)]$, and so $G[f(x),G[f(y),f(z)]] = G[G[f(x),f(y)],f(z)]$.

Suppose that there exists a solution f of $f(x+y) = G[f(x),f(y)]$ that takes on at least N^3 distinct values $V = \{v_1,\ldots,v_{N^3}\}$. Since $\forall a,b,c \in V,\; H(a,b,c) = 0$, we know that $H(a,b,c) \equiv 0$ [47]. □

4.2. d'Alembert's equation and others. In this section, we show that the robustness of functional equations of the form $\forall x,y \;\; F[f(x-y),f(x+y),f(x),f(y)] = 0$, is not limited to addition theorems by showing that when the domain $\mathcal{D}$ is a finite group, and the range $\mathcal{R} = \mathcal{T}$ is a field containing 2, d'Alembert's equation $\forall x,y \;\; f(x+y) + f(x-y) = 2f(x)f(y)$ is a robust property on $\mathcal{G} = \{f | Pr_{x\in\mathcal{D}}[f(x) = 0] \leq \frac{1}{20}\}$. Since membership in $\mathcal{G}$ is easy to test, these robustness results lead to self-testers as described later. The techniques in this section can also be used to show that the equations $\forall x,y \;\; f(x+y)+f(x-y) = 2[f(x)+f(y)]$ and $\forall x,y \;\; f(x+y)+f(x-y) = 2f(x)$ are robust over $\mathcal{G}$.

This result does not allow us to test functions that are in $\mathcal{F}$ but not in $\mathcal{G}$ such as the 0-function. For carefully chosen domains, other functions that are solutions to these functional equations (see Table 1.2) can also take the value 0 on more than $\frac{1}{20}$ fraction of the domain: for example, $\cos x$ takes the value 0 on half of the domain $\mathcal{D} = \{i \cdot \pi/2 | 0 \leq i \leq 3\}$. The result can still be used to construct self-testers for functions satisfying d'Alembert's equation. We discuss this further in section 5.

For the following three robustness results, let $\mathcal{N}^{\text{d'Alembert}} = \{(x,y,x+y,x-y) | x,y \in \mathcal{D}\}$.

THEOREM 4.6. *Let $F(x_1,x_2,x_3,x_4) = 2f(x_1)\cdot f(x_2) - f(x_3) - f(x_4)$. Let $\mathcal{F}$ be the function family characterized by $(F,\mathcal{N}^{\text{d'Alembert}})$. Then for $\delta \geq \frac{1}{80}$, $(F,\mathcal{N}^{\text{d'Alembert}})$ is a $(4\delta,\delta)$-robust characterization of $\mathcal{F}$ in $\mathcal{G}$. In particular, if $Pr_{x,y}[f(x+y)+f(x-y) = 2f(x)f(y)] \geq 1-\delta \geq \frac{79}{80}$ and $f \in \mathcal{G}$, then the function $g(x) \equiv \text{maj}_{y\in\mathcal{D},\; f(y)\neq 0}\{\frac{f(x+y)+f(x-y)}{2f(y)}\}$ satisfies (1) $Pr_x[f(x) = g(x)] \geq 1-4\delta$ and (2) $\forall x,y \;\; g(x+y) + g(x-y) = 2g(x)g(y)$.*

THEOREM 4.7. *Let $F(x_1,x_2,x_3,x_4) = 2f(x_1)+2f(x_2)-f(x_3)-f(x_4)$. Let $\mathcal{F}$ be the function family characterized by $(F,\mathcal{N}^{\text{d'Alembert}})$. Then for $\delta \geq \frac{1}{80}$, $(F,\mathcal{N}^{\text{d'Alembert}})$ is a $(4\delta,\delta)$-robust characterization of $\mathcal{F}$ in $\mathcal{G}$. In particular, if $Pr_{x,y}[f(x+y) + f(x-y) = 2(f(x)+f(y))] \geq 1-\delta \geq \frac{79}{80}$ and $f \in \mathcal{G}$, then the function $g(x) \equiv \text{maj}_{y\in\mathcal{D},\; f(y)\neq 0}\{\frac{f(x+y)+f(x-y)}{2} - f(y)\}$ satisfies (1) $Pr_x[f(x) = g(x)] \geq 1-4\delta$ and (2) $\forall x,y \;\; g(x+y) + g(x-y) = 2(g(x)+g(y))$.*

THEOREM 4.8. *Let $F(x_1,x_2,x_3,x_4) = 2f(x_1) - f(x_3) - f(x_4)$. Let $\mathcal{F}$ be the function family characterized by $(F,\mathcal{N}^{\text{d'Alembert}})$. Then for $\delta \geq \frac{1}{80}$, $(F,\mathcal{N}^{\text{d'Alembert}})$ is a $(4\delta,\delta)$-robust characterization of $\mathcal{F}$ in $\mathcal{G}$. In particular, if $Pr_{x,y}[f(x+y)+f(x-y) = 2f(x)] \geq 1-\delta \geq \frac{79}{80}$ and $f \in \mathcal{G}$, then the function $g(x) \equiv \text{maj}_{y\in\mathcal{D},\; f(y)\neq 0}\{\frac{f(x+y)+f(x-y)}{2}\}$*

satisfies (1) $Pr_x[f(x) = g(x)] \geq 1 - 4\delta$ *and* (2) $\forall x, y \;\; g(x+y) + g(x-y) = 2g(x)$.

The proofs in this section are similar in flavor to the proofs of the robustness of the addition theorems, but since the functional equation is defined on inputs that are related in different ways, we have to take advantage of different aspects of the structure of their relationship in order to get the desired results. The proofs of all three theorems follow the same outline. In the following, we give the proof of Theorem 4.6.

Proof of Theorem 4.6. Using techniques identical to those in Lemma 4.2, we have the following lemma.

LEMMA 4.9. *g and f agree on more than $1 - 4\delta$ fraction of the inputs from $\mathcal{D}$.*

LEMMA 4.10. *For all x, $\Pr_y[g(x) = \frac{f(x+y)+f(x-y)}{2f(y)}] \geq 1-\delta'$ where $\delta' = 4\delta+2\cdot\frac{1}{20}$.*

Proof. $\Pr_{y,z}[f(y) \neq 0$ *and* $f(z) \neq 0$ *and*

$$\begin{aligned} 2f(z)(f(x+y) + f(x-y)) & \\ &= (f(x+y+z) + f(x+y-z)) \\ &\quad +(f(x-y-z) + f(x-y+z)) \\ &= (f(x+y+z) + f(x-y+z)) \\ &\quad +(f(x-y-z) + f(x+y-z)) \\ &= 2f(y)(f(x+z) + f(x-z))] \\ &\geq 1 - 4\delta - 2 \cdot \tfrac{1}{20}. \end{aligned}$$

$f(y) = 0$ or $f(z) = 0$ with probability at most $2 \cdot \frac{1}{20}$. The first and third equalities each hold with probability $1 - 2\delta$ by our assumption on f and since all the references to f are uniformly distributed in $\mathcal{D}$. The second equality always holds. If $f(y), f(z)$ are both nonzero and all equalities hold, then $\frac{f(x+y)+f(x-y)}{2f(y)} = \frac{f(x+z)+f(x-z)}{2f(z)}$. The lemma now follows from the fact that the probability that the same object is drawn twice from a set in two independent trials lower bounds the probability of drawing the most likely object in one trial. □

Finally, we can prove that g satisfies d'Alembert's equation everywhere.

LEMMA 4.11. *For all x, y, $2g(x)g(y) = g(x+y) + g(x-y)$.*

Proof.

$$\begin{aligned} \Pr_z \big[f(z) \neq 0 \text{ and } & f(z) \cdot (g(x+y) + g(x-y)) \\ &= \tfrac{f(x+y+z)+f(x+y-z)+f(x-y+z)+f(x-y-z)}{2} \\ &= \tfrac{2g(x)f(y+z)+2g(x)f(y-z)}{2} \\ &= 2g(x) \cdot g(y) \cdot f(z)\big] \\ &> 1 - 5\delta' - \tfrac{1}{20} > 0. \end{aligned}$$

$f(z) = 0$ with probability at most $\frac{1}{20}$. By Lemma 4.10, the first, second, and third equalities hold with probability $1 - 2\delta'$, $1 - 2\delta'$, and $1 - \delta'$, respectively. If $f(z) \neq 0$ and all equalities hold, then $2g(x)g(y) = g(x+y) + g(x-y)$. Since the statement is independent of z and holds with positive probability, it must hold with probability 1. □

5. Self-testing/correcting from functional equations. We give informal definitions of self-testers and self-correctors. Formal definitions are given in [19]. An (ϵ_1, ϵ_2)-*self-tester* $(0 \leq \epsilon_1 < \epsilon_2)$ for f on D must fail any program that is not $(1-\epsilon_2)$-close to f on D and must pass any program that is $(1 - \epsilon_1)$-close to f on D (the behavior of the tester is not specified for programs that are $(1 - \epsilon_2)$-close but not $(1 - \epsilon_1)$-close to f). The tester should satisfy these conditions with error probability at most β, where β is a confidence parameter input by the user. For simplicity, we

assume that $\epsilon_1 = 0$, and we drop that parameter from our claims. An *ϵ-self-corrector* for f on D is an algorithm C that uses program P as a black box, such that for every $x \in D$ and β, $\Pr[C^P(x) = f(x)] \geq 1 - \beta$ for *every* P which is $(1-\epsilon)$-close to f on D. Furthermore, all require only a small multiplicative overhead over the running time of P and are different, simpler, and faster than any correct program for f in a precise sense defined in [18]. Checkers can be constructed by finding both a self-tester and a self-corrector for the function [19].

In this section, we give self-correctors and self-testers that are based on the class of functional equations of the form $F[f(x-y), f(x+y), f(x), f(y)] = 0$. We will use the robustness theorems proved earlier for the self-testers but not for the self-correctors. We refer to the function computed by the program as P, and the correct function as $f : \mathcal{D} \to \mathcal{R}$. For purposes of exposition, we assume that $\mathcal{D}$ is a finite group. All results can be extended to rational domains of the form $\mathcal{D}_{p,s} = \{\frac{i}{s} \big| |i| \leq p\}$ using known techniques from [31], [37] (see section 6). We assume that $\mathcal{R}$ is an (possibly infinite) abelian group.

5.1. Self-correctors. The following self-corrector works for any function satisfying $\forall x, y \quad f(x) = G[f(x-y), f(x+y), f(y)]$. This includes functions satisfying an addition theorem of the form $f(x+y) = G[f(x), f(y)]$, since letting $z = x + y$, $f(z) = G[f(z-y), f(y)]$. Self-correctors for functions that are not solvable for $f(x)$, but are solvable for another of $f(x-y), f(x+y), f(y)$ can be similarly constructed.

Program self-correct(x, β)

$N \leftarrow O(\ln(1/\beta))$
Do for $m = 1, \ldots, N$
  Pick $y \in_R \mathcal{D}$
  $answer_m \leftarrow G[P(x-y), P(x+y), P(y)]$
Output the most common answer in $\{answer_m : m = 1, \ldots, N\}$

THEOREM 5.1. *Given $\mathcal{D}$ a finite group, and P and f functions over domain $\mathcal{D}$. If P is $\frac{1}{12}$-close to f over $\mathcal{D}$, then $\forall x$, Pr*[Self-Correct$(x, \beta) = f(x)] \geq 1 - \beta$.

The proof of this theorem follows the format in [19] and is based on the fact that since calls to P are made on uniformly distributed inputs in $\mathcal{D}$, at each iteration, all calls are answered correctly by P with probability at least $\frac{3}{4}$.

The existence of an ϵ-self-corrector for a class of functions $\mathcal{F}$ trivially implies that for any two functions $f_1, f_2 \in \mathcal{F}$, the quantity $Pr_x[f_1(x) \neq f_2(x)]$, or the *distance* between f_1 and f_2, must be large. Thus, the existence of self-correctors for $\mathcal{F}$ implies that the functions in $\mathcal{F}$ can be thought of as a collection of codewords with large distance.

5.2. Self-testers. In this section we show self-testers based on robust functional equations of the form $F[f(x-y), f(x+y), f(x), f(y)] = 0$. In all of our examples only a constant number of additions and multiplications are required to perform a test. Furthermore, only a constant number of tests need to be performed. It often happens that more than one functional equation can be used to specify a function family; the user can determine which of the robust functional equations is best to use for testing based on criteria such as efficiency and ease of programming.

When a family of functions satisfies the property, equality testing must be done to determine that the program is computing the correct function within the family. Although equality testing is often easier than the original testing task, it may still be inefficient, as in the case of multivariate polynomials [38]. For the functions considered in this paper, the problem of equality testing can be solved efficiently. We assume that

the function values are given at a constant number of inputs, such that these values in conjuction with the property F are enough to completely specify the function. For example, for functions satisfying addition theorems, the function values at 0 and all generators of the group suffice to completely specify the function. In particular, if the group is cyclic and generated by 1, only $f(0)$ and $f(1)$ are required since $f(x+1) = G[f(x), f(1)]$. Similarly, over $\mathcal{D}_{p,s}$ it is enough to specify the function at $0, \frac{1}{s}, \frac{-1}{s}$. It is often the case that there are certain inputs at which the function is much easier to compute and that these inputs can be used for the equality testing.

It may happen that the functional equation over the reals characterizes a different family of functions than the same functional equation over $\mathcal{D}_{p,s}$. For example, suppose we are given $\mathcal{F}$, a set of functions that is a solution to the functional equation over the reals. The set of functions that we are interested in testing is $\mathcal{F}' = \{g \mid g : \mathcal{D}_{p,s} \to \mathcal{R}, \exists f \in \mathcal{F}$ such that $\forall x \in \mathcal{D}_{p,s}, f(x) = g(x)\}$, the set of functions that are restrictions of functions in $\mathcal{F}$ to the domain $\mathcal{D}_{p,s}$. Consider also $\mathcal{F}''$, the solutions to the functional equation over $\mathcal{D}_{p,s}$. Then, since functions in $\mathcal{F}''$ must satisfy a subset of the constraints satisfied by $\mathcal{F}'$, $\mathcal{F}' \subseteq \mathcal{F}''$. It can happen that $\mathcal{F}'$ is a proper subset of $\mathcal{F}''$. Nevertheless, to test that a program purporting to compute function $f \in \mathcal{F}'$ is correct, the property test determines whether the program agrees on most inputs with some function g in $\mathcal{F}''$, and the equality test will then determine that $g = f$ as long as it is given the correct values of f at the inputs required for the equality test.

We concentrate on functions that can be tested by testers of the form given below. In the following, δ_0 is the maximum δ for which the functional equation is robust (see Theorems 4.1, 4.6, 4.7, 4.8), and the function values specifying f are given as a list $(x_i, y_i), 0 \le i \le c$ where $y_i = f(x_i)$.

Program self-test$((x_0, y_0), (x_1, y_1), \ldots, (x_c, y_c), \delta_0, \beta)$

$N \leftarrow O(\max\{\frac{1}{\delta_0}, 24\} \ln(2/\beta))$
Do for $m = 1, \ldots, N$ {*Property Test*}
  Pick $u, v \in_R \mathcal{D}$
  if $F[P(u-v), P(u+v), P(u), P(v)] \neq 0$ output FAIL and halt
Do for $i = 1, \ldots, c$ {*Equality Test*}
  If self-correct$(x_i, \beta/c) \neq y_i$ output FAIL and halt
Output PASS

THEOREM 5.2. *Given domain $\mathcal{D}$ a finite group, range $\mathcal{R}$ an abelian group, and functions P and f mapping $\mathcal{D}$ to $\mathcal{R}$. If F is $(2\delta, \delta)$-robust over domain $\mathcal{D}$, and if P is not $(\frac{1}{12})$-close to f on $\mathcal{D}$, then Pr[Self-Test$(x, \beta) =$ FAIL$] \ge 1 - \beta$. If $P \equiv f$, then Self-Test outputs PASS. Thus, Self-Test is a $\frac{1}{12}$-self-testing program for f on $\mathcal{D}$.*

The proof of the theorem is based on the robustness of F, which tells us that if there is no function g such that (1) g is usually equal to P and (2) g satisfies the property everywhere, then P is reasonably likely to fail the test. Furthermore, if there is such a function g, the equality tests are likely to fail unless $g(x_0) = f(x_0), \ldots, g(x_c) = f(x_c)$ which ensures that $g \equiv f$. Thus P fails unless it is usually equal to f. It is easy to see that by altering the choice of N, one can construct ϵ-self-testers for any $\epsilon < \frac{1}{12}$.

The above self-tester is not sufficient for testing functions using d'Alembert's equation, since we have proved its robustness only under the condition that the function P is 0 on $\le \frac{1}{20}$ of the inputs. To fix this, one may use an algorithm that depends on the fraction of inputs on which f takes the value 0. The solution to d'Alembert's equation over $\Re$ is all functions of the form $0, \cos Ax, \cosh Ax$. The 0 function is trivial to test. The $\cosh Ax$ functions are never 0, so a program purporting to compute $\cosh Ax$ can

be first tested to ensure that it does not output 0 on more than $\frac{1}{20}$ of the domain, and then the above tester may be used. The $\cos Ax$ functions are 0 only at odd multiples of $\frac{\pi}{(2A)}$. If the odd multiples of $\frac{\pi}{(2A)}$ constitute at most $\frac{1}{20}$ of the domain, then a similar procedure to the one for $\cosh Ax$ may be used. Otherwise one can use a different functional equation for which $\cos Ax$ is a solution. Alternatively, suppose the program can be modified to compute over a larger domain $\hat{\mathcal{D}}$ which contains $\mathcal{D}$ such that $f(x) = 0$ on at most $\frac{1}{20}$ of $\hat{\mathcal{D}}$. (For example, if $f(x) = \cos x, \mathcal{D} = \{i \cdot \pi/2 | 0 \leq i \leq p-1\}$, then choose $\hat{\mathcal{D}} = \{i \cdot \pi/20 | 0 \leq i \leq 10p-1\}$.) Then one can test the program over $\hat{\mathcal{D}}$. If the program passes the test, it is possible to test the program on $\mathcal{D}$ by using the self-corrector based on d'Alembert's equation to correctly compute f at any input in $\mathcal{D} \subseteq \hat{\mathcal{D}}$.

6. Extensions to rational domains. In this section, we show the self-testers and self-correctors that result from extending the results in section 5 to rational domains. We consider rational domains of the form $\mathcal{D}_{n,s} = \{\frac{i}{s} \big| \ |i| \leq n\}$.

The theorems follow the same outline as in the finite fields case, but certain additional technical details must be addressed. These technical details are similar to those used in [31], [38].

6.1. Self-correctors. The following self-corrector works for any function satisfying $\forall x, y \ \ f(x) = G[f(x-y), f(x+y), f(y)]$. Self-correctors for functions that are not solvable for $f(x)$, but are solvable for another of $f(x-y), f(x+y), f(y)$ can be similarly constructed.

As in [31], we assume that the program has been tested over a larger domain $\mathcal{D}_{m,s}$ in order to self-correct over the domain $\mathcal{D}_{n,s}$ (this requires the more general definitions of self-correcting given in [31]). It suffices that $m > 12n$.

Program self-correct(x, β)

$N \leftarrow O(\ln(1/\beta))$
Do for $i = 1, \ldots, N$
  Pick $y \in_R \mathcal{D}_{m,s}$
  $answer_i \leftarrow G[P(x-y), P(x+y), P(y)]$
Output the most common answer in $\{answer_i : i = 1, \ldots, N\}$

THEOREM 6.1. *Let m, n be such that $m > 12n$. If P is $(\frac{1}{24})$-close to f over $\mathcal{D}_{m,s}$, then $\forall x \in \mathcal{D}_{n,s}$, $Pr[\text{Self-Correct}(x, \beta) = f(x)] \geq 1 - \beta$.*

The proof of this theorem follows the format in [31].

Proof of Theorem 6.1. By the assumption on P, $P(y)$ is correct with probability at least $1 - \frac{1}{24}$. Two bad events can happen when picking $x + y$: either $x + y$ is not in $\mathcal{D}_{m,s}$ in which case we know nothing about the probability that $P(x+y)$ is correct, or $x + y$ is in $\mathcal{D}_{m,s}$ but happens to be one of the inputs for which P is incorrect. By our choice of m, the first bad situation happens with probability $\leq \frac{1}{24}$. The second bad situation happens with probability $\leq \frac{1}{24}$. If neither of these happens, then $P(x+y) = f(x+y)$. The same argument can be made for $P(x-y)$. Thus, at each iteration, $answer_i = f(x)$ with probability at least $1 - 2\frac{1}{24} - 3\frac{1}{24} > \frac{3}{4}$. □

6.2. Self-testers. The following is a self-tester for addition theorems over the rational domain $\mathcal{D}_{m,s}$. We test the program over a larger domain $\mathcal{D}_{p,s}$ in order to certify that it is usually correct over $\mathcal{D}_{m,s}$. It suffices that $p > 11m$. As in section 5.2, we assume the function values specifying f are given as a list of pairs $(x_i, y_i), 0 \leq i \leq c$ where $y_i = f(x_i)$. In addition we assume that $x_i \in \mathcal{D}_{n,s}$ for all i.

The self-tester is based on finding a neighborhood $\mathcal{N}^{\text{add}'}{}_{\mathcal{D}_{p,s}}$ such that $(\mathcal{D}_{m,s}, F_{\mathcal{D}_{p,s},\Re,\Re}, \mathcal{N}^{\text{add}'}{}_{\mathcal{D}_{p,s}})$ is an $(\epsilon, \epsilon/2)$-robust characterization.

Program self-test$((x_0, y_0), (x_1, y_1), \ldots, (x_c, y_c), n, m, \delta_0, \beta)$

$N \leftarrow O(\max\{\frac{4}{\delta_0}, 48\} \ln(2/\beta))$
{*Property Test*}
Do for $m = 1, \ldots, N$
  Choose $i \in \{1, 2, 3, 4\}$
  If $i = 1$ then $\{x_1 \leftarrow x, x_2 \leftarrow x - y, x_3 \leftarrow y\}$
   Pick $x \in_R \mathcal{D}_{m,s}$ and $y \in_R \mathcal{D}_{p,s}$
     if $P(x) \neq G[P(x - y), P(y)]$ output FAIL and halt
  Else if $i = 2$ then $\{x_1 \leftarrow x, x_2 \leftarrow x - y, x_3 \leftarrow y\}$
   Pick $x, y \in_R \mathcal{D}_{p,s}$
     if $P(x) \neq G[P(x - y), P(y)]$ output FAIL and halt
  Else if $i = 3$ then $\{x_1 \leftarrow x + y, x_2 \leftarrow x, x_3 \leftarrow y\}$
   Pick $x, y \in_R \mathcal{D}_{p,s}$
     if $P(x + y) \neq G[P(x), P(y)]$ output FAIL and halt
  Else $\{i = 4\}$ $\{x_1 \leftarrow x, x_2 \leftarrow y, x_3 \leftarrow x - y\}$
   Pick $x, y \in_R \mathcal{D}_{p,s}$
     if $P(x) \neq G[P(y), P(x - y)]$ output FAIL and halt
{*Equality Test*}
Do for $i = 1, \ldots, c$
  If self-correct$(x_i, \beta/c) \neq y_i$ output FAIL and halt
Output PASS

We have the following theorem.

THEOREM 6.2. *Let p, m, n be such that $p > 11m$ and $m > 12n$. Let the function values (x_i, y_i) specifying f be such that $x_i \in \mathcal{D}_{n,s}$. If P is not $\frac{1}{24}$-close to f on $\mathcal{D}_{m,s}$, then Pr[Self-Test(x, β) = FAIL] $\geq 1 - \beta$. If $P \equiv f$, then Self-Test outputs PASS. Thus, Self-Test is a $\frac{1}{24}$-self-testing program for f on $\mathcal{D}_{m,s}$.*

In order to show the above theorem, we need the following to show that the addition theorems are robust properties over rational domains.

LEMMA 6.3. *If (1) $Pr_{x \in \mathcal{D}_{m,s}, y \in \mathcal{D}_{p,s}}[P(x + y) = G[P(x), P(y)]] \geq 1 - \delta$, (2) $Pr_{x,y \in \mathcal{D}_{p,s}}[P(x) = G[P(x-y), P(y)]] \geq 1-\delta$, (3) $Pr_{x,y \in \mathcal{D}_{p,s}}[P(x+y) = G[P(x), P(y)]] \geq 1 - \delta$, (4) $Pr_{x,y \in \mathcal{D}_{p,s}}[P(x) = G[P(y), P(x - y)]] \geq 1 - \delta$, for $\delta < \frac{1}{48}$, then there exists a function g such that*

1. $Pr_{x \in \mathcal{D}_{m,s}}[P(x) = g(x)] \geq 1 - 2\delta = 1 - \frac{1}{24}$,
2. $\forall x, y \in \mathcal{D}_{m,s}$ $g(x + y) = G[g(x), g(y)]$.

Let $\mathcal{N}^{\text{add}'}$ be the multiset such that picking random $(x_1, x_2, x_3) \in \mathcal{N}^{\text{add}'}$ is the same as picking inputs from the above distribution. If the functional equation is satisfied with probability at least $1 - \frac{\delta}{4}$ when neighborhoods are chosen from $\mathcal{N}^{\text{add}'}$, then each of the four conditions of the theorem are met. Thus we have the following theorem.

THEOREM 6.4. *Let $\mathcal{R} = \mathcal{T}$ be a group. Let G be such that G satisfies $\forall a, b, c \in \mathcal{R}$ $G[a, G[b, c]] = G[G[a, b], c]$. Let $F(x, y) = P(x + y) - G[P(x), P(y)]$. Then for all $\delta < \frac{1}{48}$, $(\mathcal{D}_{m,s}, F_{\mathcal{D}_{p,s},\mathcal{R},\mathcal{T}}, \mathcal{N}^{\text{add}'}{}_{\mathcal{D}_{p,s}})$ is $(2\delta, \frac{\delta}{4})$-robust.*

Proof of Lemma 6.3. Define $g(x)$ to be $\text{maj}_{z \in \mathcal{D}_{p,s}}\{G(P(x - z), P(z))\}$.

The Lemma 6.5 follows from the first condition on P and a counting argument.

LEMMA 6.5. *g and P agree on more than $1 - 2\delta$ fraction of the inputs from $\mathcal{D}_{m,s}$.*

For the following lemmas, set $\gamma = \frac{m}{2p}$.

LEMMA 6.6. *For all* $x \in \mathcal{D}_{2m,s}$, $\Pr_{z\in\mathcal{D}_{p,s}}[g(x) = G[P(x-z), P(z)]] \geq 1-\delta'$ *where* $\delta' = 2\delta + 2\gamma$.

Proof. For $x \in \mathcal{D}_{2m,s}$, $Pr_{y\in\mathcal{D}_{p,s}}[x + y \in \mathcal{D}_{p,s}] \geq 1 - \gamma$. Thus,

$$\begin{aligned}\Pr_{y,w\in\mathcal{D}_{p,s}}&\left[G[P(x-y), P(y)]\right.\\ &= G[G[P(x-w), P(w-y)], P(y)]\\ &= G[P(x-w), G[P(w-y), P(y)]]\\ &= G[P(x-w), P(w)]]\\ &\geq 1 - 2\delta - 2\gamma.\end{aligned}$$

By the fourth condition on P, the first equality holds with probability $1 - \delta - 2\gamma$. By the second condition on P, the third equality holds with probability $1 - \delta$. The second equality always holds.

The lemma now follows from the observation that the probability that the same object is drawn twice from a set in two independent trials lower bounds the probability of drawing the most likely object in one trial. □

Finally, we prove that g satisfies the addition theorems everywhere.

LEMMA 6.7. *For all* $x, y \in \mathcal{D}_{m,s}$, $g(x + y) = G[g(x), g(y)]$.

Proof.

$$\begin{aligned}\Pr_{u,v\in\mathcal{D}_{p,s}}&\left[G[g(x), g(y)]\right.\\ &= G[G[P(u), P(x-u)], G[P(v), P(y-v)]]\\ &= G[P(u), G[P(x-u), G[P(v), P(y-v)]]]\\ &= G[P(u), G[G[P(x-u), P(v)], P(y-v)]]\\ &= G[P(u), G[P(x-u+v), P(y-v)]]\\ &= G[P(u), P(x+y-u)]\\ &= g(x+y)\big]\\ &\geq 1 - 3\delta' - 2\delta - 3\gamma = 1 - 8\delta - 9\gamma > 0.\end{aligned}$$

By Lemma 6.6, the first equality holds with probability $1 - 2\delta'$ and the last equality holds with probability $1-\delta'$ (since $x+y \in \mathcal{D}_{2m,s}$). By the third assumption on P, the fourth equality holds with probability $1-\delta-\gamma$. By the second assumption on P, the fifth equality holds with probability $1-\delta-2\gamma$. The other equalities always hold, due to the structure of G.

Since the statement is independent of u, v and holds with positive probability, it must hold with probability 1. This also proves Lemma 6.3. □

6.3. An example: Testing the cosh function. In this subsection we will illustrate how to apply the above techniques to construct a self-tester and self-corrector for a particular function, namely, the cosh function, over a given domain. Suppose that one would like to reliably use a program that purports to compute the cosh function over the domain $\mathcal{D}_{2^k,2^k} = \{\frac{i}{2^k}\big|\ |i| \leq 2^k\}$ (the numbers between –1 and 1 with k bits of precision). Assume that the correct values of $\cosh 0, \cosh\frac{-1}{2^k}, \cosh\frac{1}{2^k}$ are given and that the program purports to compute cosh over the larger domain $\mathcal{D}_{2^{k+8},2^k} = \{\frac{i}{2^k}\big||i| \leq 2^{k+8}\}$. Recall that cosh is one of the solutions to the functional equation $\forall x, y f(x + y) = f(x)f(y) + \sqrt{f(x)^2 - 1}\sqrt{f(y)^2 - 1}$. Furthermore, $\cosh(x)$ is the only solution to the functional equation that agrees with the given values of $\cosh 0, \cosh\frac{-1}{2^k}, \cosh\frac{1}{2^k}$, since $f(0), f(\frac{1}{2}^k), f(\frac{-1}{2^k})$ determine the values of f over the

whole domain $\mathcal{D}_{2^{k+8},2^k}$ via $f(\frac{i+1}{2^k}) = f(\frac{1}{2^k})f(\frac{i}{2^k}) + \sqrt{f(1/2^k)^2 - 1}\sqrt{f(i/2^k)^2 - 1}$ and $f(\frac{i-1}{2^k}) = f(\frac{-1}{2^k})f(\frac{i}{2^k}) + \sqrt{f(-1/2^k)^2 - 1}\sqrt{f(i/2^k)^2 - 1}$.[8]

One should first test the program over the domain $\mathcal{D}_{2^{k+4},2^k} = \{\frac{i}{2^k} \big|\ |i| \leq 2^{k+4}\}$, using the tester given in subsection 6.2 (with $\delta_0 = \frac{1}{48}, s = 2^k, m = 2^{k+4}, p = 2^{k+8}, n = 2^k$ so that $p > 11m$ and $m > 12n$). By Theorem 6.2 (which in turn uses Theorem 6.4), if the program is always correct on domain $\mathrm{q}\mathcal{D}_{2^{k+8},2^k} = \{\frac{i}{2^k} \big|\ |i| \leq 2^{k+8}\}$ the tester will output PASS, and if the program is incorrect on greater than $\frac{1}{24}$ fraction of the domain $\mathcal{D}_{2^{k+4},2^k} = \{\frac{i}{2^k} \big|\ |i| \leq 2^{k+4}\}$, then the tester will output FAIL. If the program is incorrect on less than $\frac{1}{24}$ fraction of the domain $\mathcal{D}_{2^{k+4},2^k}$, one can use the self-corrector in subsection 6.1 (with $s = 2^k, m = 2^{k+4}, n = 2^k$ so that $m > 12n$) in order to compute the correct value of $\cosh x$ for all $x \in \mathcal{D}_{2^k,2^k} = \{\frac{i}{2^k} \big|\ |i| \leq 2^k\}$. The correctness of the self-corrector is guaranteed by Theorem 6.1.

7. Conclusions and directions for further research. We have studied the question of when functions characterized by functional equations of the form $\forall x, y$ $F[f(x-y), f(x+y), f(x), f(y)] = 0$ are robust. However, we still do not have a complete answer to this question. Even for addition theorems $\forall x, y \quad f(x+y) = G[f(x), f(y)]$, we do not know what happens when G does not satisfy $G[a, G[b, c]] = G[G[a, b], c]$ $\forall a, b, c$. More generally, many other general types of functional equations have been identified, including those on multivariate functions and systems of functional equations, but we do not know which ones are robust. Given a functional equation, is there an (efficient) algorithm to determine whether or not it is robust? Is it the case that any property that leads to a self-corrector is robust?

Systems of functional equations can be used to define more than one unknown function by their joint properties. For example, Pexider's equations are $f(x+y) = g(x) + h(y)$, $f(x+y) = g(x)h(y)$, $f(xy) = g(x) + h(y)$, and $f(xy) = g(x)h(y)$, which are generalizations of Cauchy's original functional equations. These equations have applications to the library setting [19], where programs for several functions can be used to self-test and self-correct each other, as long as none of the answers are a priori assumed to be correct. The library setting has been used to find checkers that are significantly more efficient for functions such as determinant and rank. Are there any other examples of functions where their mutual properties lead to more efficient testers?

It is important to find methods to extend all robustness results to the case of real-valued computation as in [31], [5], [27]. One point of difficulty is that in real-valued computation, none of the functional equations will be satisfied exactly, even when the program is giving very good approximations to the correct answers. Thus, the area of functional inequalities, which is the investigation of which families of functions satisfy inequalities such as $|f(x+y) - f(x) - f(y)| \leq \epsilon$, directly applies to this setting. Much of the work in [31], [5], [27] has been in relating the class of functions that are solutions to functional inequalities to the class of functions that are solutions of the corresponding functional equations. Several functional inequalities have been shown to be robust in [31], [5], [27]. Other related results used for testing matrix multiplication, linear system solution, matrix inversion, and determinant computation are in [5].

Acknowledgments. We wish to thank Mike Luby, for initial conversations that eventually led to the idea behind this work, and Nati Linial for directing us to the

[8] Self-testers and self-correctors for $\cosh x$ can be constructed via other functional equations that $\cosh x$ satisfies as long as they are shown to be robust and equality testing is possible.

area of functional equations as well as to many of the references. We thank Madhu Sudan for many interesting conversations, comments, and insights on this work. We also thank Richard Zippel for his technical advice on the zero testing of algebraic functions and for guiding us through the literature regarding the scope of the theorems in this paper. We thank S. Ravi Kumar and Funda Ergün for discussions regarding the definitions in this paper and comments on the write-up. We thank S. Ravi Kumar and D. Sivakumar for pointing out an error. We thank Lucian Bebchuk, Ran Canetti, Oded Goldreich, Diane Hernek, Sandy Irani, Mike Luby, Dana Ron, and especially the anonymous referee for greatly improving the presentation of this paper.

REFERENCES

[1] M. Abadi, J. Feigenbaum, and J. Kilian, *On hiding information from an oracle*, J. Comput. System Sci., 39 (1989), pp. 29–50.

[2] L. Adleman, M. Huang, and K. Kompella, *Efficient checkers for number-theoretic computations*, Inform. and Comput., 121(1995), pp. 93–102.

[3] J. Aczél, *Lectures on Functional Equations and Applications*, Academic Press, New York, 1966.

[4] J. Aczél and J. Dhombres, *Functional Equations in Several Variables*, Cambridge University Press, Cambridge, UK, 1989.

[5] S. Ar, M. Blum, B. Codenotti, and P. Gemmell, *Checking approximate computations over the reals*, in Proc. 25th Annual ACM Symp. Theory of Computing, 1993, pp. 786–795.

[6] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and the intractability of approximation problems*, in Proc. 33rd IEEE Symp. Foundations of Computer Science, 1992, pp. 14–23.

[7] S. Arora and S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, in Proc. 33rd Annual IEEE Symp. Foundations of Computer Science, 1992, pp. 2–13.

[8] D. Beaver and J. Feigenbaum, *Hiding instances in multioracle queries*, in Proc. 7th Annual Symp. Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 415, Springer-Verlag, 1990, pp. 37–48.

[9] L. Babai, L. Fortnow, and C. Lund, *Non-deterministic exponential time has two-prover interactive protocols*, Comput. Complexity, 1 (1991), pp. 3–40.

[10] L. Babai, L. Fortnow, L. Levin, and M. Szegedy, *Checking computations in polylogarithmic time*, in Proc. 23rd Annual ACM Symposium on Theory of Computing, 1991, pp. 21–31.

[11] M. Bellare, D. Coppersmith, J. Hastad, M. Kiwi, and M. Sudan, *Linearity testing in characteristic two*, Proc. 36th Annual Symp. Foundations of Computer Science, 1995, pp. 434–441.

[12] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and nonapproximability–Toward tight results*, SIAM J. Comput., 27 (1998), pp. 804–915.

[13] M. Bellare, S. Goldwasser, C. Lund, and A. Russell, *Efficient probabilistically checkable proofs*, in Proc. 25th Annual ACM Symp. Theory of Computing, 1993, pp. 294–304.

[14] M. Bellare and M. Sudan, *Improved non-approximability results*, in Proc. ACM Symp. Theory of Computing, 1994, pp. 184–193.

[15] M. Blum, *Designing Programs to Check Their Work*, Tech. report TR-88-009, International Computer Science Institute, University of California at Berkeley, Berkeley, CA, 1988.

[16] M. Blum, B. Codenotti, P. Gemmell, and T. Shahoumian, *Self-Correcting for Function Fields of Finite Transcendental Degree*, Procc. 22nd Internat. Colloquium ICALP 95, Lecture Notes in Comput. Sci. 944, 1995, pp. 547–557.

[17] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor, *Checking the correctness of memories*, Proc. 32nd Annual Symp. Foundations of Computer Science, 1991, pp. 90–99.

[18] M. Blum and S. Kannan, *Program correctness checking...and the design of programs that check their work*, in Proc. 21st Annual ACM Symp. on Theory of Computing, 1989, pp. 86–97.

[19] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595.

[20] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-*

random bits, SIAM J. Comput., 13 (1984), pp. 850–864.
[21] M. Blum and H. Wasserman, *Program result-checking: A theory of testing meets a test of theory*, in Proc. 35th FOCS, 1994, pp. 382–392.
[22] E. Castillo and M. R. Ruiz-Cobo, *Functional Equations and Modelling in Science and Engineering*, Marcel Dekker, New York, 1992.
[23] R. Cleve and M. Luby, *A Note on Self-Testing/Correcting Methods for Trigonometric Functions*, Tech. report TR-90-032, International Computer Science Institute, University of California at Berkeley, Berkeley, CA, 1990.
[24] W. J. Cody, *Performance evaluation of programs related to the real gamma function*, ACM Trans. Math. Software, 17 (1991), pp. 46–54.
[25] W. J. Cody and L. Stoltz, *The use of Taylor series to test accuracy of function programs*, ACM Trans. Math. Software, 17 (1991), pp. 55–63.
[26] D. Coppersmith, untitled manuscript, December 1989 (result described in [19]).
[27] F. Ergün, S. R. Kumar, and R. Rubinfeld, *Approximate checking of polynomials and functional equations*, Proc. 37th IEEE Symp. Foundations of Computer Science, 1996, pp. 292–303.
[28] F. Ergün, S. R. Kumar, and D. Sivakumar, *Self-testing without the generator bottleneck*, SIAM J. Comput., to appear.
[29] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy, *Approximating clique is almost NP-complete*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, 1991, pp. 2–12.
[30] J. Feigenbaum and L. Fortnow, *Random-self-reducibility of complete sets*, SIAM J. Comput., 22 (1993), pp. 994–1005.
[31] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson, *Self-testing/correcting for polynomials and for approximate functions*, in Proc. 23rd Annual ACM Symposium on Theory of Computing, 1991, pp. 32–42.
[32] M. Klawe, *Limitations on explicit constructions of expanding graphs*, SIAM J. Comput., 13 (1984), pp. 156–166.
[33] S. Ravi Kumar and D. Sivakumar, *Efficient self-testing/self-correction of linear recurrences*, Proc. 37th IEEE Foundations of Computer Science, 1996, pp. 602–611.
[34] R. Lipton, *New directions in testing*, in Distributed Computing and Cryptography, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 2, AMS, Providence, RI, 1991, pp. 191–202.
[35] S. Micali, *Computationally-sound proofs*, Proc. 35th Annual Symp. Foundations of Computer Science, 1994, pp. 436–453.
[36] A. Polischuk and D. Spielman, *Nearly linear size holographic proofs*, Proc. 26th ACM Symp. on Theory of Computing, 1994, pp. 194–203.
[37] R. Rubinfeld and M. Sudan, *Testing polynomial functions efficiently and over rational domains*, in Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms, SIAM, Philadelphia, 1992, pp. 23–43.
[38] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.
[39] N. Schriver, personal communication, February 1990.
[40] M. Sudan, *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*, Ph.D. thesis, University of California, Berkeley, CA, 1992.
[41] M. Sudan, personal communication, 1994.
[42] F. Vainstein, *Error detection and correction in numerical computations by algebraic methods*, Proc. 9th Internat. Symp. AAECC-9, Lecture Notes in Comput. Sci. 539, Springer-Verlag, New York, 1991.
[43] F. Vainstein, *Algebraic Methods in Hardware/Software Testing*, Ph.D. thesis, Boston University, Boston, MA, 1993.
[44] F. Vainstein, *Low redundancy polynomial checks for numerical computation*, Appl. Algebra Engrg., Comm. Comput., 7 (1996), pp. 439–447.
[45] F. Vainstein, *Self checking design technique for numerical computations*, J. VLSI Design, 5 (1995), pp. 385–392.
[46] R. Zippel, *Zero testing of algebraic functions*, Inform. Process. Lett., 61 (1997), pp. 63–67.
[47] R. Zippel, *Effective Polynomial Computation*, Kluwer Academic Publishers, Norwell, MA, 1993.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 1998–2029

NEW RESULTS ON THE OLD k-OPT ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM*

BARUN CHANDRA†, HOWARD KARLOFF‡, AND CRAIG TOVEY§

Abstract. Local search with k-change neighborhoods is perhaps the oldest and most widely used heuristic method for the traveling salesman problem, yet almost no theoretical performance guarantees for it were previously known. This paper develops several results, some worst-case and some probabilistic, on the performance of 2- and k-opt local search for the traveling salesman problem, with respect to both the quality of the solution and the speed with which it is obtained.

Key words. graph algorithms, analysis of algorithms, explicit machine computation and programs (in optimization heading), explicit machine computation and programs (in computer science heading)

AMS subject classifications. 05C85, 68Q25, 49-04, 68-04

PII. S0097539793251244

1. Introduction.

Local search with k-change neighborhoods is perhaps the oldest and most widely used heuristic method for the traveling salesman problem (TSP) [12, 16]. Given a graph $G = (V, E)$ and a tour T of G ("tour" is synonymous with "Hamiltonian cycle"), a tour T' is said to be obtained from T by an *improving k-change* if T' is shorter than T, and T' is obtained by removing k edges from T and adding a disjoint set of k edges. The k-opt algorithm starts with an arbitrary initial tour and incrementally improves on this tour by making successive improving k'-changes for any $k' \leq k$, terminating when no such improving changes can be made. This paper develops several results, some worst-case and some probabilistic, on the performance of 2- and k-opt algorithms for the TSP, with respect to the two principal criteria, quality and speed. Regarding quality: how good is a locally optimal solution? The only results on this question that we are aware of are due to Grover [6] and Lueker [15]. Grover proves that for any (symmetric) TSP instance, any 2-optimal tour has length at most the average of all tour lengths. (This result also was credited to Edelberg [15, p. 7] but without reference.) Lueker gives a construction for which this bound is tight when tour lengths differ. Our results regarding solution quality are as follows:

- For TSPs satisfying the triangle inequality (i.e., the distances are those in an n-point metric space), the worst-case performance ratio of 2-opt is at most $4\sqrt{n}$ for all n and at least $\frac{1}{4}\sqrt{n}$ for infinitely many n. The k-opt algorithm can have a performance ratio that is at least $\frac{1}{4}n^{\frac{1}{2k}}$ for infinitely many n.
- For TSPs embedded in the normed space $\mathbb{R}^m$, the worst-case performance ratio of 2-opt, and hence k-opt, is $O(\log n)$. If the points are embedded in $\mathbb{R}^2$ and the distances are Euclidean, then there is a $c > 0$ such that the worst-case performance ratio of 2-opt is at least $c \cdot \frac{\log n}{\log \log n}$ for infinitely many n.

*Received by the editors June 29, 1993; accepted for publication (in revised form) May 14, 1997; published electronically June 23, 1999.
http://www.siam.org/journals/sicomp/28-6/25124.html

†Department of Computer Science, University of New Haven, West Haven, CT 06516 (barun@charger.newhaven.edu).

‡College of Computing, Georgia Tech, Atlanta, GA 30332-0280 (howard@cc.gatech.edu). This author was supported in part by NSF grant CCR 9107349.

§School of Industrial and Systems Engineering and College of Computing, Georgia Tech, Atlanta, GA 30332-0280 (ctovey@isye.gatech.edu). This author was supported in part by NSF grant DDM-9215467.

- For all norms on $\mathbb{R}^m$, there exists a constant c such that any 2-optimal tour on any TSP instance in the unit hypercube (with norm-induced distances) has length less than $cn^{1-1/m}$. A corollary is that for random instances in the hypercube, i.e., points are sampled independently and identically distributed (i.i.d.) from the uniform distribution on the hypercube, with high probability, the performance ratio of 2-opt, even if it makes the worst possible sequence of improvements, is $O(1)$. Furthermore, the expected value of the performance ratio is also $O(1)$.

Regarding speed: how many iterations does local search require? There seem to be three previous results on this question. The first two of these are worst-case. Lueker [15] constructs a TSP instance for which there exists an exponentially long sequence of improving 2-changes. Johnson, Papadimitriou, and Yannakakis [7] and Krentel [10] prove the existence of instances with exponentially long improving sequences with respect to k-changes, for all sufficiently large k. (In fact, [7] and [10] prove something much stronger: there are an integer k, instances of the TSP, and initial tours such that, starting with the initial tour, *every* sequence of moves each of which is an improving k'-change (for some $k' \leq k$) and which terminates with a k-optimal tour has exponential length. Furthermore, Krentel [11] claims that $k = 8$ suffices.)

Here we extend Lueker's construction for all $k > 2$, giving explicit instances with exponentially long improving sequences.

The third previously known result is probabilistic. Kern [9] shows that for random Euclidean instances on the unit square, the probability is at least $1 - c/n$ that the expected number of iterations required by 2-opt is $O(n^{16})$, where c is a constant independent of the number of points n. It was not known if the expected number of iterations is polynomial. Our main probabilistic results regarding speed are:

- For random Euclidean instances in the unit square, the expected number of iterations required by 2-opt is $O(n^{10} \log n)$.
- For random L_1 instances in the unit hypercube, the expected number of iterations required by 2-opt is $O(n^6 \log n)$.

Taken together, our results provide the first theoretical proof of the quality of 2-opt as a heuristic for random TSP instances in the unit square. In particular, the expected time is polynomial, and the expected worst-case performance ratio is bounded by a constant.

2. Preliminaries. We begin by stating some definitions and notation that will be used throughout the paper.

A *metric space* (V, d) is a nonempty set V of points and a function $d : V \times V \to \mathbb{R}$, called *distance*, satisfying the following properties for all $x, y, z \in V$:

(i) $d(x, y) \geq 0$ and $d(x, y) = 0$ if and only if $x = y$;
(ii) $d(x, y) = d(y, x)$;
(iii) $d(x, z) \leq d(x, y) + d(y, z)$.

A *norm* N on $\mathbb{R}^m$ is a function $||\cdot|| : \mathbb{R}^m \to \mathbb{R}$ satisfying the following properties:

(i) For all $x \in \mathbb{R}^m$, $\|x\| \geq 0$, and $\|x\| = 0$ if and only if $x = 0$;
(ii) $\|cx\| = |c| \cdot \|x\|$ for every $c \in \mathbb{R}$ and $x \in \mathbb{R}^m$;
(iii) $\|x + y\| \leq \|x\| + \|y\|$ for every $x, y \in \mathbb{R}^m$.

A norm N induces a distance function defined as $d_N(x, y) = \|x - y\|$. If the norm of an m-dimensional vector $x = (x_1, x_2, \ldots, x_m)$ is defined as $\sqrt[p]{|x_1^p| + |x_2^p| + \cdots + |x_m^p|}$, where p is a positive integer, then the corresponding metric is called the L_p metric. The L_2 metric is called the *Euclidean metric*.

A *geometric graph* is given by a finite nonempty set V of points in $\mathbb{R}^m$ and a norm

N on $\mathbb{R}^m$. The graph is a complete weighted graph on V with the weight of edge $\{x, y\}$ being $d_N(x, y) = ||x - y||$. When the metric considered is the Euclidean metric, the graph is called a *Euclidean graph.* Given a weighted graph $G = (V, E)$ we refer to the weight or length of an edge $e \in E$ by $wt(e)$. Given a collection of edges $E' \subseteq E$, the weight $wt(E')$ of E' is the sum of the weights of the edges in E'; if the edges in E' form a tour T', we also refer to $wt(T')$ as the *length* of the tour. We denote an optimal tour by $OPT(G)$. Since we work only in complete graphs in a metric space (so given V, $G = (V, E)$ is completely determined), we also abuse notation slightly and refer to $OPT(V)$.

Given a weighted graph $G = (V, E)$ and a tour T of G, a tour T' is said to be obtained from T by an *improving k-change* if T' is shorter than T, and T' is obtained by removing a set of k edges from T and adding a disjoint set of k edges. A tour T is said to be *k-optimal* if for all $k' \leq k$, no improving k'-change can be made to T. The *k-opt algorithm* starts with an arbitrary initial tour T_0 and incrementally improves on this tour by finding $T_1, T_2, \ldots, T_z$, where T_{i+1} is obtained from T_i by an improving k'-change for some $k' \leq k$, and T_z is k-optimal.

The k-opt algorithm can start from many different initial tours, and even starting from the same initial tour, k-opt can end up in many different k-optimal tours. All the upper bounds in this paper are proved for the worst possible outcome of k-opt.

3. Bounds on performance ratios in metric spaces.

3.1. An upper bound for 2-opt. In this subsection we show that if the points are chosen from a metric space, then the worst-case performance ratio of 2-opt is bounded by $4\sqrt{n}$.

As a preliminary, we first prove that the performance ratio of k-opt cannot be bounded by a function of n if the triangle inequality is not imposed.

THEOREM 3.1. *For all $k \geq 2$, for all $n \geq 2k + 8$, for all $M > 0$, there exists a complete weighted graph G on n vertices, with strictly positive weights, containing a k-optimal tour T' such that $wt(T')/OPT(G) > M$.*

Proof. We prove the result for all k even and $n \geq 2k + 6$. The result will follow for k odd since k-optimality implies $(k - 1)$-optimality. The idea of the construction is to take a pair of Hamiltonian cycles in G which differ by a $(k + 1)$-change. We set the weights of all edges in these cycles to ϵ; all other edges in G are given very large weight. For one special edge in the first cycle, we change the weight to 1. This keeps the first cycle k-optimal but now its weight is many times that of the second cycle.

The graph G has n vertices denoted $1, 2, \ldots, n$. Its edge weights are as follows.

1. $wt(1, 2) = 1$.
2. $wt(i, i + 1) = \epsilon$ for all $i > 1$, and $wt(n, 1) = \epsilon$.
3. $wt(k + 3, 2k + 4) = \epsilon$.
4. $wt(j, 2k + 4 - j) = \epsilon$ for all $1 \leq j \leq k$.
5. All other edges have weight kn.

The two tours, the optimal T and the k-optimal T', differ only in the order in which they visit $1, 2, 3, \ldots, 2k+4$. Both tours visit $2k+4, 2k+5, 2k+6, \ldots, n-2, n-1, n, 1$ in that order. T' starts with $1, 2, 3, 4, \ldots, 2k + 3, 2k + 4$, whereas T starts with $1; 2k+3, 2k+2, 2, 3; 2k+1, 2k, 4, 5; 2k-1, 2k-2, 6, 7; \ldots; k+5, k+4, k, k+1; k+2, k+3, 2k+4$. (The semicolons appear only to help the reader.) This tour has weight $n\epsilon$. For $k = 2$, T is $1; 7, 6, 2, 3; 4, 5, 8; 9, 10, 11, 12, \ldots, n$. When $k = 8$ and $n = 26$, the optimal tour T is $1; 19, 18, 2, 3; 17, 16, 4, 5; 15, 14, 6, 7; 13, 12, 8, 9; 10, 11, 20; 21, 22, 23, 24, 25, 26$. See Figure 1.

The tour T' is $1, 2, 3, \ldots, n$ with weight $1+(n-1)\epsilon$. To verify that T' is k-optimal,

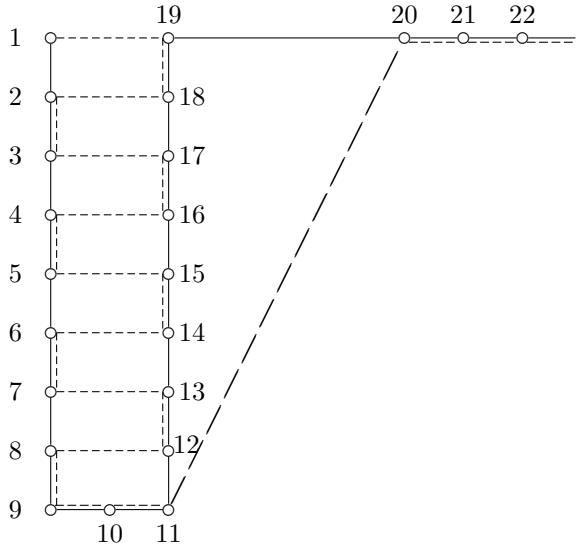


FIG. 1. *Part of the optimal tour T (dashed edges) and part of the k-optimal tour T' (solid edges) for $k = 8$.*

observe that when $\epsilon < 1$, any tour which can be obtained from T' by an improving k'-change, $k' \leq k$, must have weight $n\epsilon$, but T is the unique Hamiltonian cycle in G consisting entirely of ϵ-cost edges and T cannot be obtained from T' by an improving k' change (T' has $k+1$ edges not in T). If we set $\epsilon = 1/((M+1)n)$, the performance ratio will exceed M, as desired. □

The performance ratio can be bounded by a function of n if the triangle inequality is imposed. We will prove that if the points are chosen from a metric space, then the worst-case performance ratio of 2-opt is bounded by $4\sqrt{n}$.

Let M be any arbitrary metric space with a distance function d. Let V be a set of points in M, and let $n = |V|$. Let $OPT(V)$ be an optimal tour on V and let $T(V)$ be *any* 2-opt tour. We first state a simple fact which follows from the triangle inequality.

FACT 3.2. $V' \subseteq V \Rightarrow wt(OPT(V')) \leq wt(OPT(V))$.

LEMMA 3.3. *For any $k \in \{1, 2, \ldots, n\}$, let $E_k = \{$edges $e \in T(V) | wt(e) > \frac{2 \cdot wt(OPT(V))}{\sqrt{k}}\}$. Then $|E_k| < k$.*

Proof. Suppose otherwise; so for some k, $r := |E_k| \geq k$. Orient the edges of $T(V)$ in a consistent manner, i.e., so that the directed edges form a directed Hamiltonian cycle. Consider the directed edges (with the same orientation) of E_k, $(t_1, h_1), (t_2, h_2), \ldots, (t_r, h_r)$, where the t_i's are the tails and the h_i's are the heads of these directed edges.

We first see that not too many tails can be clustered very closely together. Consider any sphere of radius $\frac{wt(OPT(V))}{\sqrt{k}}$ around some point in the metric space. We show that the number of tails (of edges from E_k) in this sphere is less than $\sqrt{k}$.

Suppose otherwise, so that the tails $t_{i_1}, t_{i_2}, \ldots, t_{i_p}$ all lie in the sphere for some $p \geq \sqrt{k}$. Let $h_{i_1}, h_{i_2}, \ldots, h_{i_p}$ be the corresponding heads. For any $u \neq v$, $d(t_{i_u}, t_{i_v}) \leq \frac{2 \cdot wt(OPT(V))}{\sqrt{k}}$, since t_{i_u} and t_{i_v} lie in the sphere. This implies that $d(h_{i_u}, h_{i_v}) \geq \frac{2 \cdot wt(OPT(V))}{\sqrt{k}}$, since otherwise we get a shorter valid tour $(T(V) \cup \{(t_{i_u}, t_{i_v}), (h_{i_u}, h_{i_v})\})$ $-\{(t_{i_u}, h_{i_u}), (t_{i_v}, h_{i_v})\}$ with a 2-change operation. But since, by supposition, we have

$p \geq \sqrt{k}$ heads and these heads are pairwise at a distance at least $\frac{2 \cdot wt(OPT(V))}{\sqrt{k}}$ apart, the optimal tour on these heads is of length at least $2 \cdot wt(OPT(V))$, which contradicts Fact 3.2.

Now we show that a large number of tails have to be at a large distance apart. Pick any arbitrary tail t_i and consider the sphere of radius $\frac{wt(OPT(V))}{\sqrt{k}}$ centered around t_i. "Kill" all the tails within this sphere. By the above argument, fewer than $\sqrt{k}$ tails can have been killed. Now pick any remaining "live" tail and kill all tails in the sphere centered at this tail. Repeat this process until all tails have been killed. Since there are at least k tails and in a single iteration we kill fewer than $\sqrt{k}$ tails, this process can be repeated more than $\sqrt{k}$ times. Clearly, the tails at the center of the spheres are at a distance greater than $\frac{wt(OPT(V))}{\sqrt{k}}$ apart from each other, and there are greater than $\sqrt{k}$ of them; therefore, the optimal tour on the tails is of length greater than $wt(OPT(V))$, which contradicts Fact 3.2. □

THEOREM 3.4. $\frac{wt(T(V))}{wt(OPT(V))} \leq 4\sqrt{n}$.

Proof. Note that Lemma 3.3 implies that the weight of the kth largest edge is at most $\frac{2 \cdot wt(OPT(V))}{\sqrt{k}}$. Hence

$$\begin{aligned} wt(T(V)) &= \sum_{k=1}^{n} wt(k\text{th largest edge}) \\ &\leq \sum_{k=1}^{n} \frac{2 \cdot wt(OPT(V))}{\sqrt{k}} \\ &= 2 \cdot wt(OPT(V)) \sum_{k=1}^{n} \frac{1}{\sqrt{k}} \\ &\leq 2 \cdot wt(OPT(V)) \int_{x=0}^{n} \frac{1}{\sqrt{x}} \\ &= 4 \cdot wt(OPT(V)) \cdot \sqrt{n}. \end{aligned}$$ □

3.2. Lower bounds for 2-opt and *k*-opt. Next we show that the upper bound for 2-opt's performance ratio given by Theorem 3.4 is tight to within a factor of 16. We also provide lower bounds for k-opt.

THEOREM 3.5. *For any $k \geq 2$, for infinitely many values of n, there exists a complete weighted n-node graph $G_{k,n}$ with positive edge weights satisfying the triangle inequality, and a k-optimal tour $T_{k,n}$ of $G_{k,n}$, such that $\frac{wt(T_{k,n})}{wt(OPT(G_{k,n}))} \geq \frac{1}{4} \cdot n^{1/2k}$ if $k \geq 3$, and $\frac{wt(T_{k,n})}{wt(OPT(G_{k,n}))} \geq \frac{1}{4}\sqrt{n}$ if $k = 2$.*

Define the *girth* of a graph as the number of edges in its smallest cycle, provided it is not a forest.

LEMMA 3.6. *Suppose there exists a connected unweighted graph $G_{k,n,m}$, with n vertices and m edges, having girth at least $2k$, in which every vertex has even degree. Then there is an m-vertex complete weighted graph G_1 with positive edge weights satisfying the triangle inequality and a k-optimal tour T of G_1 such that $\frac{wt(T)}{wt(OPT(G_1))} \geq \frac{m}{2n}$.*

Proof. Assume that we are given $G_{k,n,m} = G = (V, E)$. Since G is connected and every vertex has even degree, G has an Eulerian tour ET.

Using G and ET, we construct a complete weighted graph $G_1 = (V_1, E_1)$ and a tour T for G_1. Let $V(G) = \{x_1, x_2, \ldots, x_n\}$. We think of each vertex x_i in G as a

"supervertex" corresponding to $deg_G(x_i)/2$ vertices in G_1, so

$$V_1 = \{x_{1,1}, \ldots, x_{1,deg_G(x_1)/2}, \ldots, x_{n,1}, \ldots, x_{n,deg_G(x_n)/2}\}.$$

The number of vertices in G_1 is $(deg_G(x_1) + \cdots + deg_G(x_n))/2 = m$.

Let $d_G(x_i, x_j)$ be the length of the shortest path from x_i to x_j in G. The edge weights of G_1 are as follows:

1. for all $i, s, t, s \neq t$, $wt(x_{i,s}, x_{i,t}) = \epsilon$, where $\epsilon = \frac{1}{n^2}$;
2. for all i, j, s, t, $i \neq j$, $wt(x_{i,s}, x_{j,t}) = d_G(x_i, x_j)$.

By inspection, it is easy to see that the edge weights of G_1 satisfy the triangle inequality.

The tour T on G_1 is constructed as follows. Suppose that the rth vertex of the Eulerian tour ET of G is vertex x_i. Suppose that this is the lth time ET has entered and exited vertex x_i, $1 \leq l \leq deg(x_i)/2$. Then the rth vertex of tour T of G_1 is $x_{i,l}$. Since ET enters and exits each vertex x_i of G exactly $deg_G(x_i)/2$ times and there are precisely $deg_G(x_i)/2$ vertices in each supervertex, this procedure gives us a tour T. Note that for all $\{x_i, x_j\} \in E$ there is a unique pair s, t such that $\{x_{i,s}, x_{j,t}\} \in T$.

Since the weight of the minimum spanning tree of G_1 is at most $(n-1) + (n \cdot n \cdot \epsilon) = n$ and edge weights satisfy the triangle inequality, $wt(OPT(G_1)) \leq 2n$. In the tour T, there are m edges each of weight 1, and so $wt(T) = m$. Hence we get $\frac{wt(T)}{wt(OPT(G_1))} \geq \frac{m}{2n}$, so all we need to prove Lemma 3.6 is the following.

CLAIM 3.7. *T is k-optimal.*

Proof. If not, then there is a tour T' of G_1 which is obtained from T by a k'-change operation, $k' \leq k$, such that $wt(T') < wt(T)$.

A *closed walk* is a walk which begins and ends at the same vertex, repeated edges and vertices allowed. A *simple closed walk* is a closed walk with no repeated edges.

CLAIM 3.8. *Viewing T and T' as sets of edges, there are sets $C \subseteq T - T'$ and $C' \subseteq T' - T$ such that $C \cup C'$ is the edge set of a simple closed walk, $|C| = |C'| \leq k$, $wt(C) > wt(C')$, and every vertex in V_1 is incident to the same number* (0, 1, *or* 2) *of edges of C as C'.*

Proof. Let Δ denote symmetric difference. Since for all $v \in V_1$, $deg_{T \Delta T'} v$ is either 0, 2, or 4, $T \Delta T'$ can be partitioned into a collection of vertex-disjoint simple closed walks $P_1, P_2, \ldots, P_s$. Further, since $wt(T) > wt(T')$, at least one of the P_i, say, P_1, has to satisfy $wt(P_1 \cap T) > wt(P_1 \cap T')$. Since $|P_1 \cap T| = |P_1 \cap T'|$, it is easy to verify that $C = P_1 \cap T$ and $C' = P_1 \cap T'$ have the desired properties. □

Let C, C' be as in Claim 3.8. Let $G_2 = (V, E_2)$ be a weighted multigraph with the following edges. Between every pair of distinct vertices there will be one edge of positive integral weight and zero or one edge of weight -1. Specifically, between x_i and x_j, $i \neq j$, there is an edge in E_2 of weight $d_G(x_i, x_j)$, which is a positive integer. For that x_i and x_j, if there are s, t such that $\{x_{i,s}, x_{j,t}\} \in C$, then ($s$ and t are unique and) in addition to the edge of positive weight between x_i and x_j, there is an edge in E_2 between x_i and x_j of weight $-d_G(x_i, x_j)$. A crucial fact is that $-d_G(x_i, x_j) = -1$ in this case, because every edge in T is of weight 1. Let us denote the set of edges of positive weight in E_2 as D' and let us denote the set of edges of weight -1 as D. Each edge in C gives rise to exactly one edge in D, so $|D| = |C| \leq k$.

Note that there is an obvious correspondence between the edges of G_1 and the edges of positive weight in G_2: the vertices inside a supervertex in G_1 are merged into a single vertex in G_2, with the intrasupervertex edges in G_1 "disappearing."

Edges from C', like arbitrary edges of G_1, are either of weight ϵ or of positive integral weight. An edge in C' of positive integral weight is an edge $\{x_{i,s}, x_{j,t}\}$ for some

$i \neq j$ having weight $d_G(x_i, x_j)$ and is said to correspond to the edge in D' between x_i and x_j of weight $d_G(x_i, x_j)$. An edge in C' of weight ϵ is said to correspond to nothing. An edge in C is of unit length and is an edge $\{x_{i,s}, x_{j,t}\}$ such that $d_G(x_i, x_j) = 1$, i.e., $\{x_i, x_j\} \in E$. Such an edge is said to correspond to the edge in D (of weight -1) between x_i and x_j. (Several edges of C' may correspond to the same edge in D'. However, different edges in C correspond to different edges in D.) With this correspondence, the simple closed walk in G_1 which uses each edge in $C \cup C'$ exactly once corresponds to a closed walk P in G_2. (Edges of weight ϵ are not needed and do not appear.) P need not be simple since edges in D' may have several "preimages" in C'. Since each edge in $C \cup C'$ is traversed exactly once and different edges in C correspond to different edges in D, it follows that no edges in D are traversed twice. Each, in fact, is traversed exactly once by P. The weight of P, which is the sum of the weights of the edges it traverses, counting multiplicities, is at most $wt(C') - wt(C) < 0$.

Let $G_3 = (V, E_3)$ be a weighted multigraph obtained by replacing each edge from G by two edges, one of weight $+1$ and one of weight -1. An edge of G_2 of positive integral weight is an edge between some x_i and x_j with $i \neq j$. Such an edge has weight $d_G(x_i, x_j)$ and is said to correspond to some fixed (shortest) path in G_3 between x_i and x_j consisting of $d_G(x_i, x_j)$ edges of weight $+1$. An edge of G_2 of weight -1 between, say, x_i and x_j, is said to correspond to the identical edge in G_3. (There will be many more negative edges in G_3 than there are in G_2, since C is small.) With this correspondence, the closed walk P in G_2 corresponds to a closed walk W in G_3 of the same weight. Edges of weight $+1$ may be traversed many times, but no edge of weight -1 can be traversed even twice, since no edge of weight -1 is traversed twice by P. Let the edges of weight $+1$ in W be edges $c_1, c_2, \ldots, c_r$ occurring in W $m_1, m_2, \ldots, m_r$ times, respectively. The number of edges of weight $+1$ in W, including multiplicities, of course, equals $\sum_{j=1}^{r} m_j$. Since $wt(W) = wt(P) < 0$ and $wt(W) = (\sum_{j=1}^{r} m_j) + |D|(-1)$, we have $m_1 + m_2 + \cdots + m_r < |D|$. Also, the number of edges in W is $(\sum_{j=1}^{r} m_j) + |D| < 2|D| \leq 2k$.

One of the following must be true:

- For every edge in W of weight -1, there is another edge in W with the same endpoints. But since W never has two negative edges with the same endpoints, this other edge must have weight $+1$. We infer that $wt(W) \geq 0$, a contradiction.
- There is some edge in W of weight -1 such that there is no other edge in W with the same endpoints. But since W is a closed walk, this implies that there is some set $S \subseteq W$ of edges, $|S| > 2$, such that S is the edge set of a cycle (without repeated vertices, of course). Since $S \subseteq W$ and W has fewer than $2k$ edges, S also has fewer than $2k$ edges. But then there is a cycle in G corresponding to S, and this cycle has fewer than $2k$ edges since S has fewer than $2k$ edges, a contradiction since the girth of G is at least $2k$. □

LEMMA 3.9. *For all $k \geq 2$, for infinitely many n the graphs $G_{k,n,m}$ of Lemma 3.6 exist with $\frac{m}{2n} \geq \frac{m^{1/2k}}{4}$.*

Proof. To prove that these graphs exist for infinitely many n, it suffices to show that for any n_0, there exists such a graph $G_{k,n,m}$ with $n > n_0$.

We first present an extremal graph-theoretic lemma [4, Theorem 1.4′, Chapter III].

FACT 3.10. *Let q, δ, and g be positive integers such that $q \geq \frac{(\delta-1)^{g-1}-1}{\delta-2}$. Then there exists a δ-regular graph (all vertices have degree δ) having $2q$ vertices and girth at least g.*

Let $p \geq n_0$ be a positive integer. Let $q = (2p)^{2k-1}$, $\delta = 2p$, and $g = 2k$. The parameters q, δ, g satisfy the hypothesis of Fact 3.10; let G' be the graph from Fact 3.10. G' has $2q$ vertices, girth at least $2k$, and is $(2p)$-regular. Let G be the largest connected component of G'. We claim that G has the desired properties.

Clearly, G is connected, every vertex has even degree, and the girth is at least $2k$. Let $n = |V(G)|$. Since $p \geq n_0$ and G is $2p$-regular, we get $n > 2p > n_0$. Let $m = |E(G)|$. Since G is $2p$-regular, $m = pn \leq p(2q) = 2p(2p)^{2k-1} = (2p)^{2k}$, which implies that $\frac{m}{2n} = \frac{p}{2} \geq \frac{m^{1/2k}}{4}$. This completes the proof of Lemma 3.9. □

LEMMA 3.11. *For infinitely many n the graphs $G_{2,n,m}$ of Lemma 3.6 exist with $\frac{m}{2n} = \frac{\sqrt{m}}{4}$.*

Proof. We will prove the result for all values of n which are multiples of 4. Let $p = n/4$. Let $G = K_{2p,2p}$, i.e., a complete bipartite graph with $2p$ vertices on each side. G is connected, every vertex has even degree, and G has no cycles of length 3. G has exactly $m = 4p^2$ edges so $m/n = p = \sqrt{m}/2$. □

Theorem 3.5 now follows from Lemmas 3.6, 3.9, and 3.11.

4. Bounds on performance ratios for geometric graphs. In the previous section we found that the triangle inequality by itself ensures a $\Theta(\sqrt{n})$ worst-case performance ratio. Now we put stronger conditions on the distances, requiring them to be induced by a norm on $\mathbb{R}^m$, and show that the worst-case performance ratio is between $c \log n / \log \log n$ and $O(\log n)$.

4.1. The upper bound. We find an upper bound on the performance ratio of any 2-optimal tour for geometric graphs, under any norm and in any dimension. A large portion of this subsection is based on concepts presented in [5].

We begin by stating a well-known property about norms and introducing a few definitions. Consider any positive integer $m \geq 2$ and any norm N on $\mathbb{R}^m$. Let $d_N(x, y)$ denote the distance between x and y in the metric generated by N. Let $d(x, y)$ denote the Euclidean distance between x and y. By the well-known comparability of norms [14, p. 132], there exist $l_N, u_N > 0$ such that for every x and y,

$$l_N \cdot d_N(x, y) \leq d(x, y) \leq u_N \cdot d_N(x, y). \tag{1}$$

In this section we use the concept of angles. As usual, angles are defined by the inner product and the Euclidean metric. The angle between a and b in $\mathbb{R}^m$ is

$$\arccos \frac{a \cdot b}{||a||_2 ||b||_2},$$

which we take to be in the interval $[0, \pi]$.

Consider any norm N on $\mathbb{R}^m$. For l_N and u_N satisfying (1), let $\theta_N = \arctan(\frac{l_N}{4u_N})$. Define the angle between directed line segments $\overrightarrow{ux}$ and $\overrightarrow{vy}$ to be the angle between the vectors $x-u$ and $y-v$. ($\overrightarrow{ab}$ denotes a line segment directed from a to b.) Two directed line segments $\overrightarrow{ux}$ and $\overrightarrow{vy}$ are said to be *similar-directional* (with respect to N, l_N, and u_N) if the angle between them is at most θ_N. More intuitively, similar-directional means that the two directed line segments point in almost the same direction, since the angle θ_N is small. Note that for $\theta_N < \pi/2$, if $\overrightarrow{ux}$ and $\overrightarrow{vy}$ are similar-directional, then $\overrightarrow{ux}$ and $\overrightarrow{yv}$ are *not* similar-directional. N, l_N, and u_N will be implicit when we write *similar-directional* instead of *similar-directional with respect to N, l_N, and u_N.*

Let V be a finite nonempty set of points in $\mathbb{R}^m$ with norm N. Let G be the geometric graph induced by V. Let T' be a 2-optimal tour of G with (directed) edge set E' (2-optimality is with respect to distances on the metric induced by N).

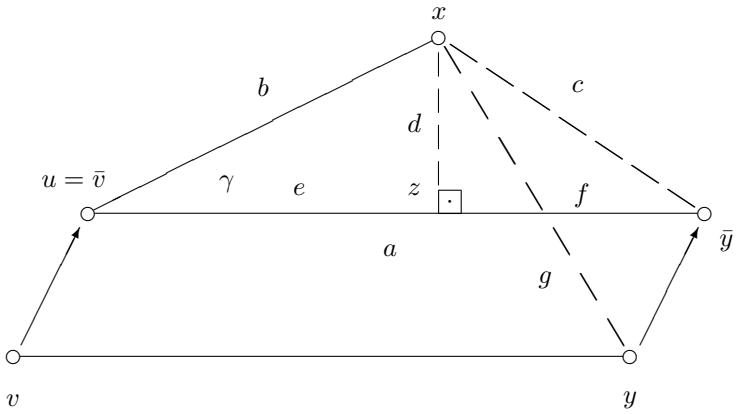


FIG. 2. *Illustration for Lemma* 4.1 (*Case* 1).

Build a set $\boldsymbol{E'}$ of directed line segments in $\mathbb{R}^m$ corresponding to E' as follows. Suppose the tour $T' = (v_1, v_2, \ldots, v_n, v_1)$. Then $\boldsymbol{E'} = \{\overrightarrow{v_1v_2}, \overrightarrow{v_2v_3}, \ldots, \overrightarrow{v_{n-1}v_n}, \overrightarrow{v_nv_1}\}$. Every vertex is the tail of exactly one line segment in $\boldsymbol{E'}$ and the head of exactly one line segment in $\boldsymbol{E'}$.

We now present an important technical lemma. Intuitively, this lemma says that, if there are two similar-directional directed line segments $\overrightarrow{ux}$ and $\overrightarrow{vy}$ in $\boldsymbol{E'}$, then u and v must be separated by a distance greater than half the length of the shorter segment. Consequently, the originating points of two similar-directional segments cannot be too close together.

LEMMA 4.1. *Let $G = (V, E)$ be a geometric graph in $\mathbb{R}^m$ under norm N. Let T' be a 2-optimal tour of G having (directed) edge set E'. Let $\overrightarrow{ux}$ and $\overrightarrow{vy}$ be any two similar-directional segments in $\boldsymbol{E'}$. If $d_N(u, x) \le d_N(v, y)$, then $d_N(u, v) > \frac{1}{2} \cdot d_N(u, x)$.*

Proof. Let $l_N, u_N > 0$ be the constants defined in (1) and $\theta_N = \arctan \frac{l_N}{4u_N}$. Let γ be the angle between directed segments $\overrightarrow{ux}$ and $\overrightarrow{vy}$. To prove the lemma we assume that

$$d_N(u, x) \le d_N(v, y), \qquad \gamma \le \theta_N, \qquad \text{and} \qquad d_N(u, v) \le \frac{1}{2} \cdot d_N(u, x) \tag{2}$$

and we derive a contradiction.

Note that since $l_N \le u_N$, $\tan \theta_N = \frac{l_N}{4u_N} < 1$. Therefore, if $\gamma \ge \pi/4$, $\gamma > \theta_N$. Thus, we may assume that $\gamma < \pi/4$.

Consider the configuration obtained by translating $\overrightarrow{vy}$ in space such that v coincides with u. Let $\bar{v} = u$ be the translate of v, and let $\bar{y}$ be the translate of y. Points $u = \bar{v}, x, \bar{y}$ lie in a 2-dimensional plane. The situation is illustrated by Figures 2 and 3.

Throughout this proof we use primed lower-case letters, $a', b', c', \ldots$ to denote distances in the N metric, while unprimed lower-case letters denote distances in the Euclidean metric. For example, if $a' = d_N(x, y)$, then $a = d(x, y)$, and vice versa.

Let $a' = d_N(v, y)$, $b' = d_N(u, x)$, $c' = d_N(x, \bar{y})$, and $g' = d_N(x, y)$. (Recall that $a = d(v, y)$, $b = d(u, x)$, $c = d(x, \bar{y})$, and $g = d(x, y)$ are the corresponding Euclidean distances.) Using this notation, (2) implies that

$$b' \le a' \qquad \text{and} \qquad d_N(u, v) \le \frac{1}{2} b'. \tag{3}$$

CLAIM 4.2. $g' \ge a'$.

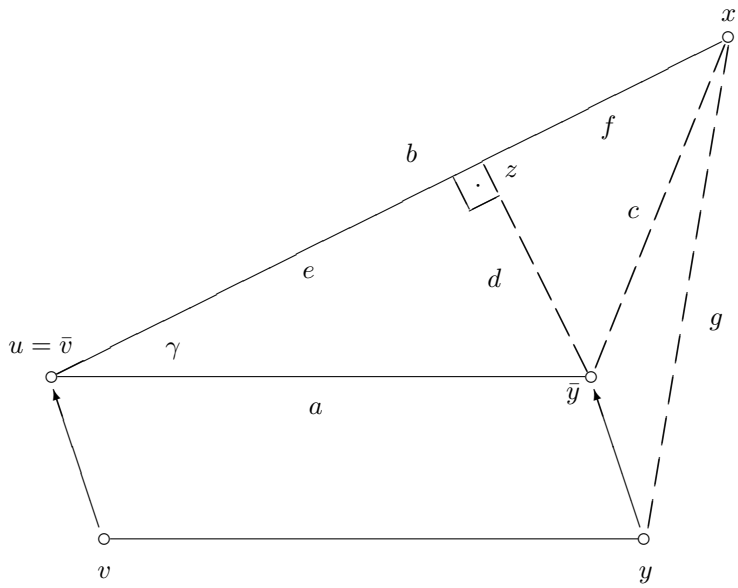


FIG. 3. *Illustration for Lemma* 4.1 (*Case* 2).

Proof. Suppose otherwise. We first see that $|\{u, x, v, y\}| = 4$. Clearly, $u \neq x$ and $v \neq y$. Clearly, $u \neq v$, since otherwise u is the tail of two line segments in E', $\overrightarrow{ux}$ and $\overrightarrow{uy}$. Similarly, $x \neq y$. If $u = y$, then $d_N(u, v) = d_N(v, y)$ and $d_N(u, v) \leq \frac{1}{2} d_N(u, x) \leq \frac{1}{2} d_N(v, y)$, which is a contradiction. If $v = x$, then $d_N(u, v) = d_N(u, x)$ and $d_N(u, v) \leq \frac{1}{2} d_N(u, x)$, which is a contradiction. Hence, $|\{u, x, v, y\}| = 4$.

By assumption, $d_N(u, v) \leq \frac{1}{2} d_N(u, x) < d_N(u, x)$, and $g' < a'$, so $d_N(u, v) + d_N(x, y) < d_N(u, x) + d_N(v, y)$. Also, (u, v) is not in E', because if it were, either $\overrightarrow{uv} \in \boldsymbol{E'}$ or $\overrightarrow{vu} \in \boldsymbol{E'}$. But if $\overrightarrow{uv} \in \boldsymbol{E'}$, then the vertex u is the tail of two line segments in E', namely, $\overrightarrow{ux}$ and $\overrightarrow{uv}$, and if $\overrightarrow{vu} \in \boldsymbol{E'}$, then the vertex v is the tail of two line segments in E', namely, $\overrightarrow{vy}$ and $\overrightarrow{vu}$. Similarly, $(x, y) \notin E'$. However, now we can interchange two edges from the tour T', (u, x) and (v, y), with the two edges (v, u) and (y, x) which are not in the tour, to get a smaller valid tour, which contradicts the 2-optimality of T'. □

We now consider two cases: $a \geq b$ and $a < b$.

The case in which $a \geq b$ is illustrated in Figure 2 where z is on the line containing $\bar{v}$ and $\bar{y}$ and xz is perpendicular to that line, $d' = d_N(x, z)$, $e' = d_N(\bar{v}, z)$, and $f' = d_N(z, \bar{y})$. Since $\gamma < \pi/4$ and $a \geq b$, z does belong to the segment $\bar{v}\bar{y}$.

The case in which $a < b$ is illustrated in Figure 3, where z is on the line containing u and x and $\bar{y}z$ is perpendicular to that line, $d' = d_N(\bar{y}, z)$, $e' = d_N(u, z)$, and $f' = d_N(z, x)$. Since $\gamma < \pi/4$ and $a < b$, z does belong to the segment ux.

Case 1. $a \geq b$ (a and b are Euclidean distances). See Figure 2.

Using (3) and the triangle inequality several times, we obtain

$$g' \leq c' + d_N(y, \bar{y}) = c' + d_N(u, v) \leq c' + \frac{1}{2} b' \leq d' + f' + \frac{1}{2} b',$$

implying

$$d' \geq g' - \frac{1}{2} b' - f' \geq a' - f' - \frac{1}{2} b' = e' - \frac{1}{2} b' \geq e' - \frac{1}{2}(d' + e'),$$

which implies

$$d'\left(1+\frac{1}{2}\right) \geq e'\left(1-\frac{1}{2}\right).$$

Using (1) we have

$$\frac{1-\frac{1}{2}}{1+\frac{1}{2}} \leq \frac{d'}{e'} \leq \frac{u_N d}{l_N e} = \frac{u_N}{l_N}\tan\gamma,$$

implying

$$\tan\gamma \geq \frac{l_N(1-\frac{1}{2})}{u_N(1+\frac{1}{2})} = \frac{l_N}{3u_N} > \frac{l_N}{4u_N} = \tan\theta_N.$$

Since $\tan\gamma > \tan\theta_N$, we have $\gamma > \theta_N$, a contradiction.

Case 2. $a < b$ (a and b are Euclidean distances). See Figure 3.

Using (3) and the triangle inequality several times, we obtain

$$g' \leq c' + d_N(y,\bar{y}) = c' + d_N(u,v) \leq c' + \frac{1}{2}b' \leq d' + f' + \frac{1}{2}b',$$

implying

$$d' \geq g' - \frac{1}{2}b' - f' \geq b' - f' - \frac{1}{2}b' = e' - \frac{1}{2}b' \geq e' - \frac{1}{2}a' \geq e' - \frac{1}{2}(d'+e')$$

(the second inequality follows from $g' \geq a' \geq b'$), which implies

$$d'\left(1+\frac{1}{2}\right) \geq e'\left(1-\frac{1}{2}\right).$$

As in Case 1, we obtain $\gamma > \theta_N$, a contradiction.

This completes the proof of Lemma 4.1. □

We now analyze the weight of the tour T'. In $\mathbb{R}^m$, for any angle $\alpha > 0$, consider a cover of $\mathbb{R}^m$ by some finite number $B(d,\alpha)$ of circular (overlapping) cones, all having the same origin P, such that two distinct points different from P in the same cone form, at P, an angle at most α. We use in Theorem 4.3 the well-known fact that $B(d,\alpha)$ is finite for every $\alpha > 0$ and every d. This *covering problem* has been extensively studied. We mention the following upper bound due to Rogers [17]:

$$B(m,\alpha) \text{ is } O\left(m^{3/2}\left(\log\frac{m}{\sin(\alpha/2)}\right)\left(\frac{1}{\sin(\alpha/2)}\right)^m\right).$$

THEOREM 4.3. *Fix m and a norm N on $\mathbb{R}^m$. Let $G = (V,E)$ be an n-vertex geometric graph in $\mathbb{R}^m$ under norm N. Let OPT be the weight of the optimal tour on G. Let T' be any 2-optimal tour of G. Then the weight of T' is $O(\log n)\cdot OPT$. (The constant implicit in the big O depends on m and N.)*

Proof. Let $\theta_N = \arctan(\frac{l_N}{4u_N})$, where the constants $l_N, u_N > 0$ are according to (1). At some arbitrary point P in $\mathbb{R}^m$, we cover the space by a constant number of circular cones $C_1, C_2, \ldots, C_{B(m,\theta_N)}$, such that every two line segments containing P and lying within the same cone subtend, at P, an angle at most θ_N. As noted, $B(m,\theta_N)$ depends only on d and N.

Call these the *original cones*. Construct $B(m,\theta_N)$ congruent cones around each of the n vertices of G by translating each original cone so that its origin shifts from

P to that vertex. Hence, corresponding to each vertex of G are $B(m,\theta_N)$ cones, one cone corresponding to each of the original cones $C_1, C_2, \ldots, C_{B(m,\theta_N)}$. Let C_{ji} be the cone with its origin at vertex j that is a translate of original cone C_i.

Let E' be the edge set of the 2-optimal tour T'. Let $\boldsymbol{E'}$ be the set of directed line segments corresponding to E'. Let $\boldsymbol{E'_i}$ be the set of directed line segments in $\boldsymbol{E'}$ that appear in $\cup_j C_{ji}$. We claim that the sum of the weights of the segments in $\boldsymbol{E'_i}$ is bounded by $O(\log n) \cdot wt(OPT)$, for $1 \le i \le B(m,\theta_N)$. Since the sets $\boldsymbol{E'_i}$ cover the set $\boldsymbol{E'}$ and the number of cones is a constant, proving this claim is enough to prove the lemma.

Clearly, all the directed line segments in $\boldsymbol{E'_i}$ are similar-directional. Hence, by Lemma 4.1, if $\overrightarrow{u_1v_1}$ and $\overrightarrow{u_2v_2}$ are two directed line segments in $\boldsymbol{E'_i}$ and if the former one is shorter, then $d_N(u_1,u_2) > \frac{1}{2} d_N(u_1,v_1)$.

Let T be an optimal tour on G, so $wt(T) = OPT$. We are now going to account for the length of the line segments in $\boldsymbol{E'_i}$, using the length of the edges in T. For integer l, call an edge in $\boldsymbol{E'_i}$ *long* if its length is at least OPT/l. Consider a walk along the edges of T starting from an arbitrary vertex and ending at the same vertex. As we walk along the path, we encounter the originating points of the long line segments in $\boldsymbol{E'_i}$. Let the order of the long line segments encountered from $\boldsymbol{E'_i}$ be $\overrightarrow{e_1}, \ldots, \overrightarrow{e_q}$.

We claim that for all $l \ge 2$, $\boldsymbol{E'_i}$ contains fewer than $2l$ long line segments. Consider the first two long directed line segments encountered, $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$. By Lemma 4.1, the distance between the originating points of $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$ is longer than $\frac{1}{2}OPT/l$. Hence the distance along T between the originating points of $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$ is also longer than $\frac{1}{2}OPT/l$. Continuing along T from the originating point of $\overrightarrow{e_2}$ to the originating point of $\overrightarrow{e_3}$, and so on to the portion of T from the originating point of $\overrightarrow{e_q}$ back to the originating point of $\overrightarrow{e_1}$, we find that OPT, the total length of T, is longer than $q[OPT/(2l)]$. This is only possible if the number q of long line segments satisfies $q < 2l$.

This means, if $l = 2^j$, $j \ge 1$, that the sum of the lengths of all the line segments in $\boldsymbol{E'_i}$ of length in $[OPT/2^j, OPT/2^{j-1}]$ is at most $(2 \cdot 2^j)[OPT/2^{j-1}] = 4 \cdot OPT$. Now setting $l = 2^j$, $j = 1, \ldots, \lceil \lg n \rceil$, the sum of the lengths of line segments in $\boldsymbol{E'_i}$ is less than

$$OPT + \sum_{j=1}^{\lceil \lg n \rceil} 4 \cdot OPT \le (5 + \lg n) OPT.$$

Since $B(m,\theta_N)$ is constant (dependent only on m and N), we conclude that $wt(\boldsymbol{E'})$ is $O(\log n) \cdot OPT$. □

4.2. A lower bound for 2-opt under L_2.

THEOREM 4.4. *There exists a constant $c > 0$ such that for infinitely many values of n, there exists an n-node graph G_n embedded in the Euclidean plane under the L_2 metric and a 2-optimal tour T_n of G_n such that $\frac{wt(T_n)}{wt(OPT(G_n))} \ge c \cdot \frac{\log n}{\log \log n}$.*

We will prove the result for those values of n which satisfy $n = 2(1 + p^2 + p^4 + p^6 + \cdots + p^{2p}) + 2p + p^{2p} + 1$ for any positive odd integer $p \ge 3$. Note that $p > c' \frac{\log n}{\log \log n}$ for some $c' > 0$.

We exhibit a set of n vertices V (all lying on the $n \times n$ grid in the Euclidean plane) such that $wt(OPT(V))$ is at most $18 \cdot p^{2p}$. We then construct a 2-optimal tour T on V of weight at least $2p \cdot p^{2p} \ge 2c' \frac{\log n}{\log \log n} \cdot p^{2p}$. Hence, we will get that $\frac{wt(T)}{wt(OPT(V))} \ge \frac{c'}{9} \cdot \frac{\log n}{\log \log n}$.

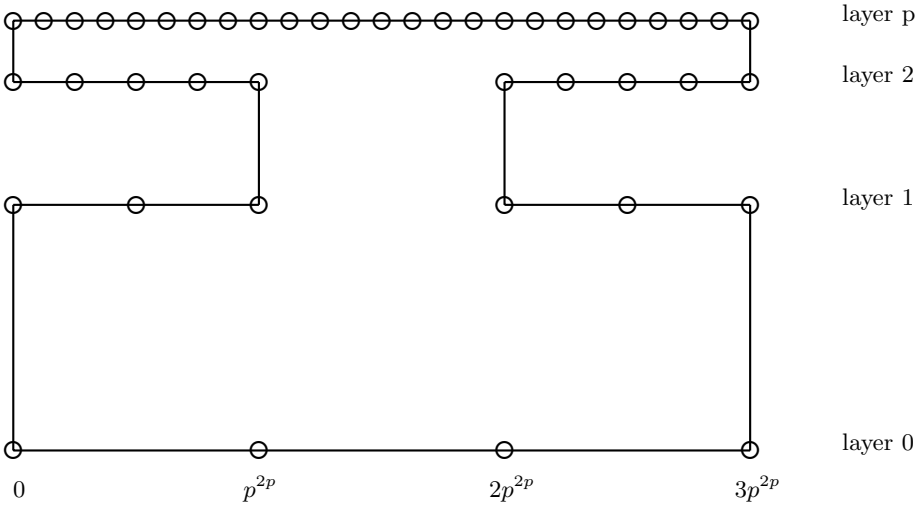


FIG. 4. *The tour T.*

Our construction is a modification of a construction by Alon and Azar [1]. (Also see Bentley and Saxe [3].)

We construct V in three parts, V_1, V_2, and V_3. The vertices in V_1 are in $p+1$ layers, where each layer is a set of equally spaced points on a horizontal line of length p^{2p}. The coordinates of the points in level i, $0 \le i \le p$, are (ja_i, b_i), where $a_i = p^{2p-2i}$ and $0 \le j \le p^{2p}/a_i$, and b_i will be defined later. Thus $a_0 = p^{2p}$, $a_1 = p^{2p-2}, \ldots$, and $a_p = 1$. Hence in layer 0 there are only two points, in layer 1 there are p^2+1, in layer i there are $p^{2i}+1 = \frac{p^{2p}}{a_i}+1$ points, up to layer p, which contains $p^{2p}+1$ points. Let $b_0 = 0$. The vertical distance between layer i and layer $i+1$ (i.e., $b_{i+1} - b_i$) is $c_i = p^{2p-1-2i}$, for all i. Note that $p \cdot a_{i+1} = p \cdot p^{2p-2i-2} = c_i = \frac{p^{2p-2i}}{p} = \frac{a_i}{p}$.

V_2 is a copy of V_1 shifted to the right. For every vertex in V_1 with coordinates (e, f), there is a vertex in V_2 with coordinates $(e+2p^{2p}, f)$. These are the only vertices in V_2.

Finally, we fill in the gaps in the topmost layer to get V_3. Since $a_p = 1$, let $V_3 = \{(j, b_p) | p^{2p} < j < 2 \cdot p^{2p}\}$. The set of all the vertices is $V = V_1 \cup V_2 \cup V_3$. Note that $|V| = n$.

CLAIM 4.5. $wt(OPT(V)) \le 18p^{2p}$.

Proof. Since $wt(OPT(V))$ is no more than twice the weight of the optimal spanning tree, it suffices to show that there is a spanning tree of weight at most $9p^{2p}$. Consider the spanning tree built as follows: for every point in every layer, other than the bottom layer, draw a vertical line to the point directly above it in the next higher layer. Also draw the horizontal line in the topmost layer (layer number p). The total length of this tree is at most

$$3p^{2p} + 2\sum_{i=0}^{p-1} c_i \left(\frac{p^{2p}}{a_i} + 1\right) \le 3p^{2p}\left(1 + \sum_{i=0}^{p-1} \frac{2c_i}{a_i}\right) = 3p^{2p}\left(1 + p\frac{2}{p}\right) = 9p^{2p}. \qquad \square$$

Define the tour T on V to be as shown in Figure 4. Note that since $p+1$ is even, we can always construct this tour.

CLAIM 4.6. $wt(T(V)) > 2p \cdot p^{2p}$.

Proof. Consider just the horizontal edges in T. Each layer has horizontal edges whose combined weight is at least $2p^{2p}$ and there are $p+1$ layers. ∎

CLAIM 4.7. *T is 2-optimal.*

We first present some simple notation. For any point A, let A_x, A_y be its x and y coordinates. We use AB to refer to both the edge (line segment) and its length. $(AB)_x$ is the length of the projection of AB onto the x-axis; i.e., $(AB)_x = |A_x - B_x|$. We define $(AB)_y$ similarly. Note that $AB \geq (AB)_x, (AB)_y$. We say that two edges AB and CD, which are either both vertical or both horizontal, *overlap* if the following holds: let the projection of AB onto the infinite line containing CD be $A'B'$. Then $A'B' \cap CD$ consists of more than a single point.

We next state and prove a simple geometric lemma.

LEMMA 4.8. *Let EF and GH be horizontal line segments in the Euclidean plane, $G_x \leq E_x < F_x \leq H_x$. Let $EF = 1$, $GH = q^2$, $q \geq 1$, so $G_x \leq E_x \leq H_x - 1$. Let the vertical distance between EF and GH be z. If $z \geq q$, then* $\min\{EG+FH, EH+FG\} \geq EF + GH$.

Proof. Let $z \geq q$. Clearly, $EH + FG > EG + FH$, so all we need to prove is that $EG + FH \geq EF + GH$. For $0 \leq a \leq q^2 - 1$, define $f(a) = \sqrt{a^2+q^2} + \sqrt{(q^2-1-a)^2+q^2}$. Let $a = E_x - G_x$. Then $H_x - F_x = (q^2-1) - a$. $EG + FH = \sqrt{a^2+z^2} + \sqrt{(q^2-1-a)^2+z^2} \geq \sqrt{a^2+q^2} + \sqrt{(q^2-1-a)^2+q^2} = f(a)$. Since $EF + GH = q^2 + 1$, in order to show that $EG + FH \geq EF + GH$, it suffices to show, for $0 \leq a \leq q^2 - 1$, that $f(a) \geq q^2 + 1$.

We will show that the minimum value of $f(a)$ in the interval $[0, q^2-1]$ occurs at $a = (q^2-1)/2$. This suffices since $f(\frac{q^2-1}{2}) = q^2 + 1$.

$$\begin{aligned} f'(a) &= \frac{a}{\sqrt{a^2+q^2}} - \frac{(q^2-1-a)}{\sqrt{(q^2-1-a)^2+q^2}} \\ &= \frac{1}{\sqrt{1+(\frac{q}{a})^2}} - \frac{1}{\sqrt{1+(\frac{q}{q^2-1-a})^2}}. \end{aligned}$$

In the interval $[0, \frac{q^2-1}{2})$, $a < q^2 - 1 - a$, and hence $f'(a) < 0$. In the interval $(\frac{q^2-1}{2}, q^2-1]$, $a > q^2 - 1 - a$, and hence $f'(a) > 0$. Hence, the minimum value of $f(a)$ in the interval $[0, q^2-1]$ occurs at $a = (q^2-1)/2$. ∎

Proof of Claim 4.7. Suppose otherwise. Thus, in a single 2-change operation, from T we can get another tour T' such that $wt(T') < wt(T)$. Label the four vertices involved as A, B, C, D so that $E(T) - E(T') = \{AB, CD\}$, $E(T') - E(T) = \{AC, BD\}$, and $AC + BD < AB + CD$. Note that the vertices A, B, C, D have to be distinct, since in a single 2-change operation we cannot replace two edges out of one vertex.

Since all edges in $E(T)$ are either horizontal or vertical, there are three cases.

Case 1: AB and CD are both vertical edges. If AB and CD overlap, then they are the two vertical edges which face each other. But then $(AC)_x + (BD)_x \geq 2p^{2p} \geq AB + CD$. If AB and CD don't overlap, assume without loss of generality that $A_y, B_y \leq C_y < D_y$. If $A_y < B_y$, then $(AC)_y \geq AB$ and $(BD)_y \geq CD$. If $A_y > B_y$, then $(BD)_y \geq AB + CD$.

Case 2: One of AB or CD is horizontal and the other is vertical. Assume without loss of generality that AB is horizontal and CD is vertical. By construction, and since A, B, C, D are all distinct, exactly one of the following subcases has to be true.

Subcase (i): Either $C_y = A_y = B_y$ or $D_y = A_y = B_y$. Then, by construction, $A_x, B_x < C_x = D_x$ or $A_x, B_x > C_x = D_x$. If $C_y = A_y$, then $(BD)_y = CD$ and, by construction, $(AC)_x \geq AB$. Similarly, if $D_y = B_y$, then $(AC)_y = CD$ and $(BD)_x \geq AB$.

Subcase (ii): $C_y, D_y > A_y = B_y$. Since AB is horizontal ($A_x \neq B_x$) and CD is vertical ($C_x = D_x$), either $C_x \neq A_x$ or $D_x \neq B_x$. If $C_x \neq A_x$, then, by construction, $(AC)_x \geq AB$ and, by construction, $(BD)_y > CD$. Similarly, if $D_x \neq B_x$, then $(BD)_x \geq AB$ and $(AC)_y > CD$.

Subcase (iii): $C_y, D_y < A_y = B_y$. If $C_y < D_y$, then $(AC)_y = (AD)_y + CD$ and, by construction, $(AD)_y > AB$, implying $(AC)_y > AB + CD$. Similarly, if $D_y < C_y$, then $(BD)_y > AB + CD$.

Case 3: AB and CD are both horizontal edges.

Subcase (i): AB and CD are nonoverlapping. Assume without loss of generality that $A_x, B_x \leq C_x < D_x$. If $A_x < B_x$, then $(AC)_x \geq AB$ and $(BD)_x \geq CD$. If $B_x < A_x$, then $(BD)_x \geq AB + CD$.

Subcase (ii): AB and CD are overlapping. Assume without loss of generality that AB is the smaller, higher edge and that $C_x \leq D_x$, so $C_x \leq A_x, B_x \leq D_x$. Suppose AB is l levels above CD, $l \geq 1$. Then $CD = p^{2l} \cdot AB$. The difference in height between them is $AB \cdot (p + p^3 + \cdots + p^{2l-1}) \geq AB \cdot p^l$. Scaling all three quantities so that $AB = 1$, we see that the hypotheses of Lemma 4.8 are satisfied, and hence $AC + BD \geq AB + CD$. ∎

5. Bounds on the length of 2-optimal tours in the unit hypercube. In this section we show that for every m and every norm on $\mathbb{R}^m$ there is a $O(n^{1-1/m})$ upper bound on the length of any 2-optimal tour on n points in the m-dimensional unit hypercube. (The constant implicit in the big O depends on m and the norm.)

Notation: an arc is an ordered pair (h, t), $h, t \in \mathbb{R}^m$. The Euclidean norm of v is denoted $||v||_2$, the Euclidean distance between h and t is denoted $d_2(h, t) := ||h - t||_2$, and the (directed) line segment between them is denoted $\overline{ht}$. The *orientation* of an arc (h, t) is the (Euclidean) unit-length vector $(h - t)/||h - t||_2$, if $h \neq t$. The difference between two orientations r and s is the angle between them as defined in section 4, i.e., $\arccos(r \cdot s)$. Thus the orientations of (h, t) and (t, h) differ by π.

Given a norm N on $\mathbb{R}^m$, we can define a metric d_N by $d_N(x, y) := ||x - y||_N$ for all $x, y \in \mathbb{R}^m$.

THEOREM 5.1. *For any dimension $m \geq 2$, for any norm N on $\mathbb{R}^m$, there exists a constant $c_{m,N}$ such that, for any set S of n points in $[0, 1]^m$, any tour on S which is 2-optimal with respect to the metric d_N has length (defined by d_N) less than $c_{m,N} \cdot n^{1-1/m}$.*

Proof. Choose m and N. As in section 4.1, by the comparability of norms, there exists a constant $K_N \geq 1$ such that $d_2(x, y)/K_N \leq d_N(x, y) \leq K_N \cdot d_2(x, y)$ for all x, y. For a given $\epsilon > 0$, define the *long arcs* in T as those of length at least ϵ in d_N. For each long arc (h, t), define the *heart* of the arc as the interior of the hypercylinder of Euclidean radius $r = (1/K_N)\epsilon/8$ and length $d_N(h, t)/2$ in the metric d_N, with the height oriented parallel to $\overline{ht}$ and the center of the hypercylinder at the midpoint of segment $\overline{ht}$. In the 2-dimensional case the heart is a rectangle of Euclidean width $(1/K_N)\epsilon/4$.

We say that arc (h^1, t^1) *attacks* arc (h^2, t^2) if and only if the line segment $\overline{h^1 t^1}$ intersects the heart of $\overline{h^2 t^2}$. Note that attacking is not a symmetric relation.

At times it will be convenient to refer to the heart of a segment, or to say that

a line segment attacks another, even if the segment's endpoints are not tour points. The intended meaning is obvious.

Let $0 < \theta < \pi/2$ be an angle whose value will be chosen later. A *family* of arcs is any collection of long arcs in $\mathbb{R}^m$ from a 2-optimal tour, whose orientations differ pairwise by at most θ.

LEMMA 5.2. *No arc attacks another arc in the same family.*

Proof. We prove the lemma in three steps. First, it suffices to consider the case where arcs have length exactly ϵ (in norm N). Second, if two arcs are parallel, of N-length ϵ, and one attacks the other, then they violate 2-optimality by at least $\epsilon/4$. Third, if two arcs are oriented within θ and one attacks the other, and both are of N-length ϵ, then they can be made parallel while still attacking, while changing things by less than $\epsilon/4$.

Step 1 begins with a simple geometric definition.

DEFINITION 5.3. *If the line segment $\overline{HT}$ contains the line segment $\overline{ht}$, and their orientations are consistent (so $d(H,h) \le d(H,t)$), then we say $\overline{HT}$ is an* extension *of $\overline{ht}$.*

For any four points h^1, t^1, h^2, t^2, define the function

$$G_N(h^1,t^1,h^2,t^2) := d_N(h^1,t^1) + d_N(h^2,t^2) - d_N(h^1,h^2) - d_N(t^1,t^2).$$

The function G_N measures the decrease in tour length if arcs (h^1,t^1) and (h^2,t^2) are removed in a 2-change operation. The two arcs cannot both be in a 2-optimal tour if G_N is strictly positive. (If G_N is positive, we can swap out arcs (h^1,t^1), (h^2,t^2) and swap in either arcs (h^1,h^2), (t^1,t^2) or arcs (h^2,h^1), (t^2,t^1). The tour remains connected.) The following lemma implies that if a pair of arcs has positive G_N value, then so does any pair of extensions of these arcs.

LEMMA 5.4. *Let $\overline{h^it^i}$ for $i = 1,2$ be two directed line segments. Now let $\overline{H^iT^i}$, $i = 1,2$, be extensions of $\overline{h^it^i}$, $i = 1,2$, respectively. Then $G_N(h^1,t^1,h^2,t^2) \le G_N(H^1,T^1,H^2,T^2)$.*

Proof. Observe how G_N changes as the shorter segments are stretched by extending the endpoints in turn. Because H^1, h^1, and t^1 are collinear and any norm scales, $d_N(h^1,t^1) = d_N(H^1,t^1) - d_N(H^1,h^1)$. By the triangle inequality, $d_N(H^1,h^2) - d_N(h^1,h^2) \le d_N(H^1,h^1) = d_N(H^1,t^1) - d_N(h^1,t^1)$. Thus extending h^1 to H^1 cannot decrease G_N. By a symmetric argument the other components of G_N are nondecreasing as the segments are extended and Lemma 5.4 follows. ∎

Suppose long arc (H^1,T^1) attacks long arc (H^2,T^2). Obviously, the segment $\overline{H^1T^1}$ is an extension of some segment $\overline{h^1t^1}$ that has length $d_N(h^1,t^1) = \epsilon$ and that also attacks (H^2,T^2). Now consider all segments that are of N-length ϵ and can be extended to $\overline{H^2T^2}$. The union of the hearts of these segments is a hypercylinder of Euclidean radius $(1/K_N)\epsilon/8$ and N-length $d_N(H^2,T^2) - \epsilon/2 \ge d_N(H^2,T^2)/2$, with the same center as that of the heart of (H^2,T^2), so it contains the heart of (H^2,T^2). Therefore, at least one of these segments is attacked by $\overline{h^1t^1}$. Denote an attacked segment by $\overline{h^2t^2}$.

By Lemma 5.4, if $G_N(h^1,t^1,h^2,t^2) > 0$, then $G_N(H^1,T^1,H^2,T^2) > 0$. To prove our lemma it therefore suffices to consider the case $d_N(h^1,t^1) = d_N(h^2,t^2) = \epsilon$. This completes step 1 of the proof.

For step 2, we consider the case of arc $\overline{h^1t^1}$, which attacks $\overline{h^2t^2}$ and is parallel to it, with both of N-length ϵ. Thus $d_N(h^1,t^1) = d_N(h^2,t^2) = \epsilon$. We have

$$G_N(h^1,t^1,h^2,t^2) = d_N(h^1,t^1) + d_N(h^2,t^2) - d_N(h^1,h^2) - d_N(t^1,t^2).$$

Let P be a point on the infinite line containing h^1 and t^1 so that $\overline{h^1P}$ is perpendicular to $\overline{Ph^2}$. We have

$$d_N(h^1, h^2) \leq d_N(h^1, P) + d_N(P, h^2).$$

Now

$$d_N(P, h^2) \leq K_N d_2(P, h^2) \leq K_N r = \frac{1}{8}\epsilon.$$

Now

$$d_N(h^1, P) \leq \frac{3}{4} d_N(h^1, t^1) = \frac{3}{4}\epsilon.$$

So

$$d_N(h^1, h^2) \leq \frac{3}{4}\epsilon + \frac{1}{8}\epsilon = \frac{7}{8}\epsilon.$$

Similarly, let Q be a point on the infinite line containing h^2 and t^2 such that $\overline{Qt^2}$ is perpendicular to $\overline{Qt^1}$. Then

$$d_N(t^1, t^2) \leq d_N(t^1, Q) + d_N(Q, t^2) \leq K_N d_2(t^1, Q) + \frac{3}{4} d_N(h^2, t^2) \leq K_N r + \frac{3}{4}\epsilon = \frac{7}{8}\epsilon.$$

Therefore,

$$G_N(h^1, t^1, h^2, t^2) \geq 2\epsilon - \frac{7}{8}\epsilon - \frac{7}{8}\epsilon = \frac{\epsilon}{4}.$$

For step 3, suppose that two segments $\overline{h^1t^1}, \overline{h^2t^2}$ of N-length ϵ have orientation differing by at most θ. Without loss of generality assume that the first attacks the second. Let R be a point on the segment $\overline{h^1t^1}$ which lies in the heart of $\overline{h^2t^2}$. Holding R fixed, rotate the segment $\overline{h^1t^1}$ so that it becomes parallel to $\overline{h^2t^2}$. Let the angle by which the segment is rotated be ξ; note that $0 \leq \xi \leq \theta$. Let $\tilde{h}^1$ and $\tilde{t}^1$ be the points that h^1 and t^1 are rotated into, respectively. We have

$$G_N(h^1, t^1, h^2, t^2) = d_N(h^1, t^1) + d_N(h^2, t^2) - d_N(h^1, h^2) - d_N(t^1, t^2),$$

$$G_N(\tilde{h}^1, \tilde{t}^1, h^2, t^2) = d_N(\tilde{h}^1, \tilde{t}^1) + d_N(h^2, t^2) - d_N(\tilde{h}^1, h^2) - d_N(\tilde{t}^1, t^2),$$

and therefore

$$|G_N(h^1, t^1, h^2, t^2) - G_N(\tilde{h}^1, \tilde{t}^1, h^2, t^2)|$$

$$\leq |d_N(h^1, t^1) - d_N(\tilde{h}^1, \tilde{t}^1)| + |d_N(h^1, h^2) - d_N(\tilde{h}^1, h^2)| + |d_N(t^1, t^2) - d_N(\tilde{t}^1, t^2)|.$$

Consider first $|d_N(h^1, t^1) - d_N(\tilde{h}^1, \tilde{t}^1)|$. By the triangle inequality,

$$-d_N(\tilde{h}^1, h^1) - d_N(t^1, \tilde{t}^1) \leq d_N(\tilde{h}^1, \tilde{t}^1) - d_N(h^1, t^1) \leq d_N(\tilde{h}^1, h^1) + d_N(t^1, \tilde{t}^1).$$

Thus

$$|d_N(\tilde{h}^1, \tilde{t}^1) - d_N(h^1, t^1)| \leq d_N(\tilde{h}^1, h^1) + d_N(t^1, \tilde{t}^1) \leq K_N[d_2(\tilde{h}^1, h^1) + d_2(t^1, \tilde{t}^1)].$$

Now

$$|d_N(h^1,h^2)-d_N(\tilde{h}^1,h^2)|\le d_N(h^1,\tilde{h}^1)\le K_N d_2(h^1,\tilde{h}^1)$$

and

$$|d_N(t^1,t^2)-d_N(\tilde{t}^1,t^2)|\le d_N(t^1,\tilde{t}^1)\le K_N d_2(t^1,\tilde{t}^1).$$

Therefore,

$$|G_N(h^1,t^1,h^2,t^2)-G_N(\tilde{h}^1,\tilde{t}^1,h^2,t^2)|\le K_N d_2(\tilde{h}^1,h^1)+K_N d_2(t^1,\tilde{t}^1)+K_N d_2(h^1,\tilde{h}^1)$$
$$+K_N d_2(t^1,\tilde{t}^1)$$

$$=2K_N[d_2(h^1,\tilde{h}^1)+d_2(t^1,\tilde{t}^1)]$$
$$=2\sqrt{2}K_N[(\sqrt{1-\cos\xi})(d_2(R,h^1)+d_2(R,t^1))]\quad(\text{bythelawofcosines})$$
$$=2\sqrt{2}K_N(\sqrt{1-\cos\xi})d_2(h^1,t^1)$$
$$\le 2\sqrt{2}K_N(\sqrt{1-\cos\xi})[K_N d_N(h^1,t^1)]$$
$$=2\sqrt{2}K_N^2(\sqrt{1-\cos\xi})\epsilon$$
$$\le[2\sqrt{2}K_N^2\sqrt{1-\cos\theta}]\epsilon.$$

Now choose $0<\theta<\pi/2$ such that $2\sqrt{2}K_N^2\sqrt{1-\cos\theta}<1/4$. This completes the third and final step of the proof of Lemma 5.2. □

DEFINITION 5.5. *The* soul *of an arc $\overline{ht}$ is the hypercylinder defined as is the heart of the arc but with Euclidean radius $(1/K_N)\epsilon/16$ and N-length $d_N(h,t)/4$, which is half that of the heart.*

LEMMA 5.6. *If the souls of two long arcs intersect, then they attack each other.*

Proof. Let p be a point of intersection of the souls of (h^1,t^1) and (h^2,t^2). Let p^i denote the point on segment $\overline{h^it^i}$ of minimum Euclidean distance to p. Then, on the one hand, p^i is in the soul of (h^i,t^i) and $d_2(p,p^i)<(1/K_N)\epsilon/16$ for $i=1,2$. By the triangle inequality we have $d_2(p^1,p^2)<(1/K_N)\epsilon/8$.

On the other hand, the heart of (h^1,t^1) has Euclidean radius $(1/K_N)\epsilon/8$, and it extends (in metric d_N) $d_N(h^1,t^1)/8\ge\epsilon/8$ beyond the soul along the arc in both directions as well. Therefore, for any point q of the arc that is in the soul, its open Euclidean ball of radius $(1/K_N)\epsilon/8$ (the set of points at Euclidean distance less than $(1/K_N)\epsilon/8$ from q) is completely contained in the heart.

Taking $q=p^1$, it follows that p^2 is in the heart of (h^1,t^1). Therefore, (h^2,t^2) attacks (h^1,t^1). Taking $q=p^2$, it follows that p^1 is in the heart of (h^2,t^2) and (h^1,t^1) attacks (h^2,t^2). □

Any soul is contained in the slightly larger than unit hypercube of (Euclidean) side length $1+2(1/K_N)\epsilon/16$ and volume at most $k_1=(9/8)^m$ (if $\epsilon\le 1$). By Lemmas 5.2 and 5.6 the sum of the volumes of the souls in a family F is at most k_1.

Now the volume of the soul of arc (h,t) is $k_2\epsilon^{m-1}d_N(h,t)$ for some constant $k_2=k_2(N)$. So

$$k_2\epsilon^{m-1}\sum_{(h,t)\in F}d_N(h,t)\le k_1$$

and hence

$$\sum_{(h,t)\in F} d_N(h,t) \le k_1 \epsilon^{1-m}/k_2.$$

This bounds the sum of the lengths of long arcs in a single family.

For every possible orientation $||u||_2 = 1$, define a corresponding set $F_u = \{v \mid ||v||_2 = 1, v \cdot u > \cos(\theta/2)\}$. Notice that for any T the set F_u induces a family of arcs of T. The set of all F_u is an open cover of the compact unit sphere $\{u| \, ||u||_2 = 1\}$. Extract a finite subcover of cardinality k_3. Note that k_3 is independent of T and n. However, since θ depends on N (and m), so does k_3.

For all T the subcover provides a finite collection of families whose union is the set of all long arcs in the tour T. Therefore, the sum of the lengths of all long arcs in T is bounded by $k_3[k_1\epsilon^{1-m}/k_2]$.

The total N-length of all short arcs is obviously bounded by $n\epsilon$. Choose $\epsilon = n^{-1/m}$. The sum of the N-lengths of all arcs in T is less than $n\epsilon + \frac{k_1k_3}{k_2}\epsilon^{1-m} = n^{1-1/m} + \frac{k_1k_3}{k_2}n^{-(1-m)/m} = c_{m,N}n^{1-1/m}$ with $c_{m,N} = 1 + k_1k_3/k_2$, and Theorem 5.1 is proved. □

For the 2-dimensional case we can use a region larger than the soul and change a few other details to get an explicit bound on tour length.

COROLLARY 5.7. *In the 2-dimensional Euclidean case, the length of any 2-optimal tour is less than* $8\sqrt{51}\sqrt{n} + 459$ *if* $n \ge 816$.

Proof. Define the *left heart* of a long arc as that half of the arc's heart which lies strictly to the left of the arc when we are walking from tail to head. In this Euclidean case, the radius of the heart is $\epsilon/8$ and its length is $d_2(h,t)/2$. Also, θ is any positive angle with cosine exceeding 127/128.

LEMMA 5.8. *The left hearts of arcs in a family are mutually disjoint.*

Proof. Suppose that two long arcs (h^1,t^1) and (h^2,t^2) have intersecting left hearts. Suppose further that the two arcs' orientations differ by η where $0 \le \eta \le \theta$. We show that one of the arcs attacks the other and that the result follows from Lemma 5.2.

Let q be a point of intersection of the left hearts. Drop a perpendicular from q to segment $\overline{h^it^i}$ at intersection point $q^i, i = 1, 2$. Consider without loss of generality the case $d_2(q,q^1) \le d_2(q,q^2)$. We prove that in this case arc (h^1,t^1) attacks (h^2,t^2).

Extend the line segment $\overline{qq^2}$ to point q^{22} so that q^2 is the midpoint of the other points: $q^2 = (q + q^{22})/2$. Observe that the entire segment $\overline{qq^{22}}$ is within the heart (not necessarily the left heart) of (h^2,t^2). Therefore, all we have to do to prove that (h^2,t^2) is attacked is to verify that the segment $\overline{h^1t^1}$ intersects this segment $\overline{qq^{22}}$.

Let p denote the point of intersection of the (infinite) lines h^1t^1 and qq^{22}. The intersection p is sure to exist because $\eta < \pi/2$. First, we show that p is "between" q and q^{22} or simply that $d_2(q,p) \le d_2(q,q^{22})$. Since $\eta \le \theta$, we have

$$\frac{d_2(q,q^1)}{d_2(q,p)} = \cos\eta \ge 1/2.$$

Hence $d_2(q,p) \le 2d_2(q,q^1) \le 2d_2(q,q^2) = d_2(q,q^{22})$ as desired.

Second, we show that p is between h^1 and t^1. Now

$$\frac{d_2(q^1,p)}{d_2(q,q^1)} = \tan\eta \le \tan\theta \le \sqrt{3}.$$

Also, q is in the left heart of (h^1,t^1) and q^1 lies on that arc. Therefore, $d_2(q,q^1) \le \epsilon/8$. Finally, recall that the arc (h^1,t^1) is long, whence $d_2(h^1,t^1) \ge \epsilon$. Putting these

inequalities together, we find that q^1 and p are near each other:

$$d_2(q^1,p) \le \sqrt{3}d_2(q,q^1) \le \epsilon\sqrt{3}/8 < \epsilon/4 \le d_2(h^1,t^1)/4.$$

Since q^1 is in the middle half of the arc, $d_2(q^1,p) \le d_2(h^1,t^1)/4$ implies that p lies in the arc (h^1,t^1). Therefore, p is the desired point of intersection of the two segments, (h^2,t^2) is attacked, and the lemma is proved. □

Any left heart is contained in the square of side $1+\epsilon/4$ and area at most $1+9\epsilon/16$ (since $\epsilon \le 1$). By Lemma 5.8 the sum of the areas of the left hearts in a family F is at most $1+9\epsilon/16$.

The area of the left heart of arc (h,t) is $\epsilon d_2(h,t)/16$. Thus

$$(\epsilon/16)\sum_{(h,t)\in F} d_2(h,t) \le 1+9\epsilon/16$$

and hence

$$\sum_{(h,t)\in F} d_2(h,t) \le 16/\epsilon + 9.$$

The number of families needed to cover the circle is $\lceil \frac{2\pi}{\arccos\,(127/128)} \rceil = 51$. Therefore, the sum of the lengths of all long arcs in T is bounded by $51(16/\epsilon+9) = 816/\epsilon + 459$.

Choose $\epsilon = \sqrt{816}/\sqrt{n}$, which is at most 1 if $n \ge 816$. The sum of the lengths of all arcs in T is less than $\sqrt{816}\sqrt{n}$ (for the short arcs) $+(\sqrt{816}\sqrt{n}+459) = 8\sqrt{51}\sqrt{n}+459$ and Corollary 5.7 is proved. □

6. Expected value of the performance ratio in the unit hypercube. In this section we combine Theorem 5.1 with well-known distributional properties of optimal tour lengths to show that the expected performance ratio is bounded by a constant.

Let S_n be any set of n points in the m-dimensional unit hypercube $[0,1]^m \subset \mathbb{R}^m$. Let $OPT(S_n)$ be an optimal tour (under norm N) on S_n, and let $T(S_n)$ be a 2-optimal tour (under norm N). Let I_n be n points picked i.i.d. from the m-dimensional unit hypercube under the uniform distribution.

As an immediate corollary to Theorem 5.1 and a lower bound of $\Omega(n^{(m-1)/m})$ on $E[OPT(I_n)]$ [8], we infer that there exists a constant $\gamma_{m,N}$ such that $\frac{E[wt(T(I_n))]}{E[OPT(I_n)]} \le \gamma_{m,N}$ for all n; the ratio of the expected values is bounded. We now show that $\frac{wt(T(I_n))}{wt(OPT(I_n))}$ is $O(1)$ with high probability and that the expected value of this ratio is $O(1)$.

The following is easily obtainable from [8, Lemma 3, p. 190]: There exist constants $F_N > 0$ and $0 < \rho < 1$ such that for all $n > 1$,

$$P\left[wt(OPT(I_n)) \le F_N \cdot n^{\frac{m-1}{m}}\right] \le \rho^n.$$

From this and Theorem 5.1 we get the following theorem.

THEOREM 6.1.

$$P\left[wt(T(I_n)) \ge \frac{c_{m,N}}{F_N} \cdot wt(OPT(I_n))\right] \le \rho^n.$$

COROLLARY 6.2. *For all m and all norms N on $\mathbb{R}^m$ there exists a constant $c'_{m,N}$ such that*

$$E\left[\frac{wt(T(I_n))}{wt(OPT(I_n))}\right] \le c'_{m,N},$$

where $T(I_n)$ (respectively, $OPT(I_n)$) is the length of the longest 2-optimal tour (respectively, the shortest tour) on the points I_n with respect to N.

Proof. We first note that for any set of points S_n, $\frac{wt(T(S_n))}{wt(OPT(S_n))} \le n$; this follows, since if the diameter (under norm N) of S_n is D, then $wt(T(S_n)) \le nD$ and $wt(OPT(S_n)) \ge D$.

Let n_0 be such that for all $n \ge n_0,\ n\rho^n \le 1$. Let

$$\delta_N = \max_{2\le n\le n_0} E\left[\frac{wt(T(I_n))}{wt(OPT(I_n))}\right].$$

Now consider $n \ge n_0$.

$$\begin{aligned}
&E\left[\frac{wt(T(I_n))}{wt(OPT(I_n))}\right] \\
&= P[wt(T(I_n)) < \frac{c_{m,N}}{F_N}\cdot wt(OPT(I_n))]\cdot E\left[\frac{wt(T(I_n))}{wt(OPT(I_n))}\middle| wt(T(I_n)) < \frac{c_{m,N}}{F_N}\cdot wt(OPT(I_n))\right] \\
&+P[wt(T(I_n)) \ge \frac{c_{m,N}}{F_N}\cdot wt(OPT(I_n))]\cdot E\left[\frac{wt(T(I_n))}{wt(OPT(I_n))}\middle| wt(T(I_n)) \ge \frac{c_{m,N}}{F_N}\cdot wt(OPT(I_n))\right] \\
&< \frac{c_{m,N}}{F_N} + \rho^n n \\
&\le \frac{c_{m,N}}{F_N} + 1.
\end{aligned}$$

Taking $c'_{m,N} = \max\{\delta_N, \frac{c_{m,N}}{F_N} + 1\}$, we are done. ∎

7. Expected running time of 2-opt. This section gives polynomial upper bounds on the average number of iterations of 2-opt under the L_2 and L_1 norms.

7.1. The L_2 metric. In the first subsection, we prove that the average number of iterations done by the 2-opt local-improvement algorithm on n random points in the Euclidean unit square is $O(n^{10}\log n)$. Prior to this paper, no polynomial upper bound on the expected time was known. However, Kern proved a related result [9].

THEOREM 7.1. *There is a c such that the probability that 2-opt does more than n^{16} iterations is at most c/n.*

Kern's proof allows the possibility that 2-opt does exponentially many iterations with probability $\Omega(1/n)$. Kern writes: "Our approach does not seem to yield interesting results about average running times." We will prove that the expected time is polynomial, and we will rely heavily on Kern's lemmas in doing so.

The basic idea of Kern's proof is to show that, with probability at least $1 - c/n$, every iteration decreases the cost by at least $\epsilon(n) > 0$; the initial tour being of length at most $\sqrt{2}n$, the number of iterations can then not exceed $\frac{\sqrt{2}n}{\epsilon(n)}$.

To prove that the expected number of iterations is polynomial, we need the following definitions and lemma from [9].

DEFINITION 7.2. *Given points $P, Q, R, S \in [0,1]^2$, define $G(P,Q,R,S) = [d(P,Q) + d(R,S)] - [d(P,R) + d(Q,S)]$.*

DEFINITION 7.3. *Given three points P, Q, R in the unit square and $\epsilon > 0$, define $B_\epsilon(P,Q,R)$ to be the set of points S in the unit square such that $|G(P,Q,R,S)| \le \epsilon$.*

LEMMA 7.4 (see [9]). *There is a $K \ge 1$ with the following property. For any three points P, Q, R in the unit square with $P \ne Q$, the area of $B_\epsilon(P,Q,R)$ (which is the conditional probability that $|G(P,Q,R,S)| \le \epsilon$, given P, Q, R) is bounded above by $K\sqrt{\epsilon}/d(P,Q)$.*

Let $X_1, X_2, \ldots, X_n$ be points chosen independently and uniformly at random from the unit square.

DEFINITION 7.5. *If i, j, k, l are distinct elements of $\{1, 2, \ldots, n\}$, define $F(i,j,k,l) = G(X_i, X_j, X_k, X_l)$.*

DEFINITION 7.6. *Define $H = \{(i,j,k,l,i',j',k',l')$ such that $i,j,k,l,i',j',k',l' \in \{1,2,\ldots,n\}$, $|\{i,j,k,l\}| = |\{i',j',k',l'\}| = 4$, and $\{i,j,k,l\} \ne \{i',j',k',l'\}\}$.*

DEFINITION 7.7. *Given n random points $X_1, X_2, \ldots, X_n$ in the unit square, define*

$$\hat{F} = \min\{F(i',j',k',l') : (i,j,k,l,i',j',k',l') \in H$$

$$\text{and } 0 < F(i,j,k,l) \le F(i',j',k',l')\}.$$

DEFINITION 7.8. *Let $N = \min\{n!, 1/\hat{F}\}$.*

We now give a very rough road map of the proof that the expected number of iterations done by 2-opt is $O(n^{10} \log n)$. It is not hard to see that any two consecutive improving 2-changes involve distinct 4-sets of vertices. By the definition of $\hat{F}$, 2-opt must decrease the cost of the tour by at least $\hat{F}$ in any two consecutive iterations. The cost of the initial tour being at most $\sqrt{2}n$, the number of iterations cannot exceed $2\sqrt{2}n/\hat{F}$. Clearly, the number of iterations never exceeds $n!$. Thus the number of iterations done is at most $\min\{(2n\sqrt{2})\cdot n!, 2n\sqrt{2}/\hat{F}\} = (2n\sqrt{2})N$. Our goal is therefore to bound $E[N]$.

Notice that if $\hat{F} \in [\frac{\epsilon}{2}, \epsilon)$, then N, which is at least $\frac{1}{2n\sqrt{2}}$ times the number of iterations, is bounded by $\frac{2}{\epsilon}$. The chance that $\hat{F}$ is in this interval is bounded by $Cn^8\epsilon$, since $P[\hat{F} \le \epsilon] \le Cn^8\epsilon$ (this is Lemma 7.11). Hence the contribution to the expected value of N due to $[\frac{\epsilon}{2}, \epsilon)$ is bounded by $2Cn^8$. Since $N = n!$ if $\hat{F} < \frac{1}{n!}$ and $\hat{F} \le 2$ always, we need consider only $\lg n! + O(1)$ intervals, each of which contributes at most $2Cn^8$ to $E[N]$. Thus $E[N]$ is $O(n^8 \lg n!)$, and the expected number of iterations is $O(n^{10} \log n)$.

Now we continue with the proof.

LEMMA 7.9. *Let $\epsilon > 0$. Choose points $X_1, X_2, \ldots, X_n$ in the unit square independently and uniformly at random. Let $(i,j,k,l,i',j',k',l') \in H$. Suppose that $l \notin \{i',j',k',l'\}$ and that $l' \notin \{i,j,k,l\}$.*

1. *If $\{i,j\} \ne \{i',j'\}$, then $P[|F(i,j,k,l)| \le \epsilon, |F(i',j',k',l')| \le \epsilon] \le 64\pi^2K^2\epsilon$.*
2. *If $\{i,j\} = \{i',j'\}$, then $P[|F(i,j,k,l)| \le \epsilon, |F(i',j',k',l')| \le \epsilon] \le 14\pi K^2\epsilon \cdot \lg \frac{1}{\epsilon}$ if $\epsilon \le \frac{1}{2}$.*

Proof. From Lemma 7.4 we have

$$P[X_l \in B_\epsilon(X_i,X_j,X_k) : X_i, X_j, X_k] = P[|F(i,j,k,l)| \le \epsilon : X_i, X_j, X_k] \le \frac{K\sqrt{\epsilon}}{d(X_i,X_j)}.$$

Then

$$P[|F(i,j,k,l)| \le \epsilon, |F(i',j',k',l')| \le \epsilon : X_i, X_j, X_k, X_{i'}, X_{j'}, X_{k'}]$$

(notice that l and l' must be distinct from each other and from i, j, k, i', j', k', although the latter six need not be distinct)

$$\begin{aligned}
&= P[X_l \in B_\epsilon(X_i, X_j, X_k), X_{l'} \in B_\epsilon(X_{i'}, X_{j'}, X_{k'}) : X_i, X_j, X_k, X_{i'}, X_{j'}, X_{k'}] \\
&\le \frac{K\sqrt{\epsilon}}{d(X_i, X_j)} \frac{K\sqrt{\epsilon}}{d(X_{i'}, X_{j'})} \\
&= \frac{K^2\epsilon}{d(X_i, X_j) d(X_{i'}, X_{j'})}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
&lP[|F(i,j,k,l)| \le \epsilon, |F(i',j',k',l')| \le \epsilon : X_i, X_j, X_k, X_{i'}, X_{j'}, X_{k'}] \\
&\qquad \le \frac{K^2\epsilon}{d(X_i, X_j) d(X_{i'}, X_{j'})}
\end{aligned}$$

and therefore

$$P[|F(i,j,k,l)| \le \epsilon, |F(i',j',k',l')| \le \epsilon : X_i, X_j, X_{i'}, X_{j'}] \le \frac{K^2\epsilon}{d(X_i, X_j) d(X_{i'}, X_{j'})}. \tag{4}$$

If $\{i,j\} \ne \{i',j'\}$, then

$$\begin{aligned}
&P[|F(i,j,k,l)| \le \epsilon, |F(i',j',k',l')| \le \epsilon] \\
&= \sum_{r=0}^{\infty} \sum_{s=0}^{\infty} P[|F(i,j,k,l)| \le \epsilon, |F(i',j',k',l')| \le \epsilon : \\
&\qquad\qquad d(X_i, X_j) \in [2^{-r}, 2^{-r+1}), d(X_{i'}, X_{j'}) \in [2^{-s}, 2^{-s+1})] \\
&\cdot P[d(X_i, X_j) \in [2^{-r}, 2^{-r+1}), d(X_{i'}, X_{j'}) \in [2^{-s}, 2^{-s+1})]
\end{aligned}$$

Now

$$\begin{aligned}
&P[d(X_i, X_j) \in [2^{-r}, 2^{-r+1}), d(X_{i'}, X_{j'}) \in [2^{-s}, 2^{-s+1})] \\
&\quad \le P[d(X_i, X_j) \le 2^{-r+1}, d(X_{i'}, X_{j'}) \le 2^{-s+1}] \\
&\quad \le \pi(2^{-r+1})^2 \pi(2^{-s+1})^2
\end{aligned}$$

since $\{i,j\} \ne \{i',j'\}$. Thus

$$P[|F(i,j,k,l)| \le \epsilon, |F(i',j',k',l')| \le \epsilon]$$

$$\leq \sum_{r=0}^{\infty}\sum_{s=0}^{\infty} P[|F(i,j,k,l)| \leq \epsilon, |F(i',j',k',l')| \leq \epsilon : d(X_i,X_j) \in [2^{-r},2^{-r+1}), d(X_{i'},X_{j'}) \in [2^{-s},2^{-s+1})] \cdot P[d(X_i,X_j) \in [2^{-r},2^{-r+1}), d(X_{i'},X_{j'}) \in [2^{-s},2^{-s+1})]$$

$$\leq \sum_{r=0}^{\infty}\sum_{s=0}^{\infty} \frac{K^2\epsilon}{2^{-r}2^{-s}}\pi(2^{-r+1})^2\pi(2^{-s+1})^2$$
$$= 16\pi^2 \sum_{r=0}^{\infty}\sum_{s=0}^{\infty} K^2\epsilon 2^r 2^s 2^{-2r} 2^{-2s}$$
$$= 16\pi^2 K^2\epsilon \leq \sum_{r=0}^{\infty}\sum_{s=0}^{\infty} 2^{-r}2^{-s}$$
$$= 16\pi^2 K^2\epsilon \left(\sum_{r=0}^{\infty} 2^{-r}\right)\left(\sum_{s=0}^{\infty} 2^{-s}\right)$$
$$= 64\pi^2 K^2\epsilon.$$

If instead $\{i,j\} = \{i',j'\}$, we have

$$P[X_l \in B_\epsilon(X_i,X_j,X_k) : X_i,X_j,X_k] = P[|F(i,j,k,l)| \leq \epsilon : X_i,X_j,X_k] \leq \frac{K\sqrt{\epsilon}}{d(X_i,X_j)}.$$

From (4) we know that

$$P[|F(i,j,k,l)| \leq \epsilon, |F(i',j',k',l')| \leq \epsilon : X_i,X_j] \leq \frac{K^2\epsilon}{d(X_i,X_j)^2}.$$

Thus

$$P[|F(i,j,k,l)| \leq \epsilon, |F(i',j',k',l')| \leq \epsilon]$$

$$= \sum_{s=0}^{\infty} P[|F(i,j,k,l)| \leq \epsilon, |F(i',j',k',l')| \leq \epsilon : d(X_i,X_j) \in [2^{-s},2^{-s+1})] \cdot P[d(X_i,X_j) \in [2^{-s},2^{-s+1})].$$

Since

$$P[|F(i,j,k,l)| \leq \epsilon, |F(i',j',k',l')| \leq \epsilon : d(X_i,X_j) \in [2^{-s},2^{-s+1})] \leq \min\left\{\frac{K^2\epsilon}{(2^{-s})^2}, 1\right\}$$
$$= \min\{K^2\epsilon 2^{2s}, 1\},$$

we have

$$P[|F(i,j,k,l)| \leq \epsilon, |F(i',j',k',l')| \leq \epsilon]$$
$$\leq \sum_{s=0}^{\infty} P[d(X_i,X_j) \in [2^{-s},2^{-s+1})] \cdot \min\{K^2\epsilon 2^{2s}, 1\}$$

$$\leq \sum_{s=0}^{\infty} P[d(X_i, X_j) \leq 2^{-s+1}] \cdot \min\{K^2 \epsilon 2^{2s}, 1\}$$

$$\leq \sum_{s=0}^{\infty} 4\pi 2^{-2s} \cdot \min\{K^2 \epsilon 2^{2s}, 1\}.$$

Now $K^2\epsilon 2^{2s} < 1$ if and only if $s < \frac{1}{2} \lg \frac{1}{K^2\epsilon}$. Therefore, the quantity is at most

$$\sum_{s=0}^{\lfloor \frac{1}{2} \lg \frac{1}{K^2\epsilon} \rfloor} 4\pi 2^{-2s} K^2 \epsilon 2^{2s} + \sum_{s=\lceil \frac{1}{2} \lg \frac{1}{K^2\epsilon} \rceil}^{\infty} 4\pi 2^{-2s} \cdot 1$$

$$\leq 4\pi K^2 \epsilon \left(1 + \frac{1}{2} \lg \frac{1}{K^2\epsilon}\right) + 8\pi K^2 \epsilon.$$

If $\epsilon \leq \frac{1}{2}$, then since $K \geq 1$ we have

$$4\pi K^2 \epsilon \left(1 + \frac{1}{2} \lg \frac{1}{K^2\epsilon}\right) + 8\pi K^2 \epsilon \leq 14\pi K^2 \epsilon \cdot \lg \frac{1}{\epsilon}. \qquad \square$$

LEMMA 7.10. *Let $\epsilon > 0$. Let $(i, j, k, l, i', j', k', l') \in H$.*

1. *If $|\{i, j, k, l\} \cap \{i', j', k', l'\}| \leq 1$, then $P[|F(i, j, k, l)| \leq \epsilon, |F(i', j', k', l')| \leq \epsilon] \leq 64\pi^2 K^2 \epsilon$.*
2. *If $|\{i, j, k, l\} \cap \{i', j', k', l'\}| \geq 2$ and $\epsilon \leq \frac{1}{2}$,then*

$$P[|F(i, j, k, l)| \leq \epsilon, |F(i', j', k', l')| \leq \epsilon] \leq 64\pi^2 K^2 \epsilon \cdot \lg \frac{1}{\epsilon}$$

.

Proof. By the symmetry in F, we have $F(a, b, c, d) = F(c, d, a, b) = F(b, a, d, c)$. This means that it is possible to move any one of the indices into the final position without changing the value of $F(a, b, c, d)$. Formally, if $x \in \{a, b, c, d\}$, then there is a permutation (a', b', c', d') of $\{a, b, c, d\}$ such that $d' = x$ and $F(a', b', c', d') = F(a, b, c, d)$ always.

Since $\{i, j, k, l\} \neq \{i', j', k', l'\}$, we can find an x in $\{i, j, k, l\} - \{i', j', k', l'\}$ and a y in $\{i', j', k', l'\} - \{i, j, k, l\}$. By moving x and y to the last position, without loss of generality we may assume that $l \notin \{i', j', k', l'\}$ and $l' \notin \{i, j, k, l\}$.

Now we invoke Lemma 7.9. If $|\{i, j, k, l\} \cap \{i', j', k', l'\}| \leq 1$, clearly $\{i, j\} \neq \{i', j'\}$. Lemma 7.9 implies that $P[|F(i, j, k, l)| \leq \epsilon, |F(i', j', k', l')| \leq \epsilon] \leq 64\pi^2 K^2 \epsilon$.

If $|\{i, j, k, l\} \cap \{i', j', k', l'\}| \geq 2$, then possibly $\{i, j\} = \{i', j'\}$ and possibly not. In the former case, $P[|F(i, j, k, l)| \leq \epsilon, |F(i', j', k', l')| \leq \epsilon] \leq 14\pi K^2 \epsilon \cdot \lg \frac{1}{\epsilon} \leq 64\pi^2 K^2 \epsilon \cdot \lg \frac{1}{\epsilon}$ (if $\epsilon \leq \frac{1}{2}$). In the latter case, $P[|F(i, j, k, l)| \leq \epsilon, |F(i', j', k', l')| \leq \epsilon] \leq 64\pi^2 K^2 \epsilon$. $\square$

Recall the definition of $\hat{F}$:

$$\hat{F} = \min\{F(i', j', k', l') : (i, j, k, l, i', j', k', l') \in H \text{ and } 0 < F(i, j, k, l) \leq F(i', j', k', l')\}.$$

LEMMA 7.11. *Let $n \geq 8$ and let $C = 9280\pi^2 K^2$. Let $2^{-n^2} \leq \epsilon \leq \frac{1}{2}$. Then $P[\hat{F} \leq \epsilon] \leq Cn^8\epsilon$.*

In $P[\hat{F} \le \epsilon] \le Cn^8\epsilon$, the n^8 comes from the fact that H in the definition of $\hat{F}$ has at most n^8 8-tuples. The ϵ comes from Lemma 7.4, which has a $\sqrt{\epsilon}$. Very roughly, because $\hat{F}$ involves *two* 4-tuples, we will be able to replace the $\sqrt{\epsilon}$ in Lemma 7.4 with its *square*, ϵ.

Proof. In this terminology, $\hat{F} \le \epsilon$ if and only if there is an 8-tuple $(i, j, k, l, i', j', k', l') \in H$ such that $0 < F(i, j, k, l) \le \epsilon$ and $0 < F(i', j', k', l') \le \epsilon$. Then

$$P[\hat{F} \le \epsilon] \le \sum P[|F(i, j, k, l)| \le \epsilon, |F(i', j', k', l')| \le \epsilon],$$

where the summation is over H. There are at most n^8 8-tuples $(i, j, k, l, i', j', k', l') \in H$ such that $|\{i, j, k, l\} \cap \{i', j', k', l'\}| \le 1$. There are at most 12^2n^6 8-tuples such that $|\{i, j, k, l\} \cap \{i', j', k', l'\}| \ge 2$. By Lemma 7.10, if $0 < \epsilon \le \frac{1}{2}$, then

$$P[\hat{F} \le \epsilon] \le n^8(64\pi^2K^2\epsilon) + 144n^6\left(64\pi^2K^2\epsilon \lg \frac{1}{\epsilon}\right).$$

Since $2^{-n^2} \le \epsilon \le \frac{1}{2}$, $\frac{1}{\epsilon} \le 2^{n^2}$ and $\lg \frac{1}{\epsilon} \le n^2$. Thus the preceding expression is no more than $n^8(64\pi^2K^2)\epsilon + 144n^6(64\pi^2K^2n^2)\epsilon = n^8(9280\pi^2K^2)\epsilon$. Let $C = 9280\pi^2K^2$. Therefore, $P[\hat{F} \le \epsilon] \le Cn^8\epsilon$ if $2^{-n^2} \le \epsilon \le \frac{1}{2}$ and $n \ge 8$. ∎

Recall that $N = \min\{n!, 1/\hat{F}\}$.

LEMMA 7.12. *$E[N] \le 4Cn^9 \lg n$ if $n \ge 8$.*

Proof. We have

$$\begin{aligned} E[N] &\le \sum_{r=-1}^{\lceil \lg n! \rceil} P[\hat{F} \in [2^{-r}, 2^{-r+1})] \cdot 2^r + P\left[\hat{F} < \frac{1}{n!}\right] \cdot n! \\ &\le \sum_{r=-1}^{\lceil \lg n! \rceil} P[\hat{F} \le 2^{-r+1}]2^r + P\left[\hat{F} < \frac{1}{n!}\right] n!. \end{aligned}$$

Thus

$$\begin{aligned} E[N] &\le \sum_{r=-1}^{1} P[\hat{F} \le 2^{-r+1}]2^r + \sum_{r=2}^{\lceil \lg n! \rceil} P[\hat{F} \le 2^{-r+1}]2^r + P\left[\hat{F} < \frac{1}{n!}\right] n! \\ &\le \left(\frac{1}{2} + 1 + 2\right) + \sum_{r=2}^{\lceil \lg n! \rceil} (Cn^8 2^{-r+1})2^r + Cn^8 \frac{1}{n!} n! \\ &= 3.5 + 2Cn^8(\lceil \lg n! \rceil - 1) + Cn^8 \\ &\le 3.5 + 2Cn^8(n \lg n) \\ &\le 4Cn^9 \lg n, \end{aligned}$$

since we are assuming $n \ge 8$. ∎

THEOREM 7.13. *The average number of 2-changes made by algorithm 2-opt when run on n i.i.d. uniform random points in the Euclidean unit square is at most $(8C\sqrt{2})n^{10} \lg n$, for any $n \ge 8$.*

Proof. If a 2-change is made, replacing edges (x_i, x_j) and (x_k, x_l) by (x_i, x_k) and (x_j, x_l), then the same four vertices $\{i, j, k, l\}$ cannot be used in an improving 2-change in the next iteration. Therefore, any two consecutive improving 2-changes involve distinct 4-tuples of vertices.

By the definition of $\hat{F}$, 2-opt must decrease the cost of the tour by at least $\hat{F}$ in any two consecutive iterations. The cost of the initial tour being at most $\sqrt{2}n$, the number of iterations cannot exceed $2\sqrt{2}n/\hat{F}$. Clearly, the number of iterations never exceeds $n!$.

The number of iterations done is at most

$$\begin{aligned} & \min\left\{(2n\sqrt{2})\cdot n!, \frac{2n\sqrt{2}}{\hat{F}}\right\} \\ = \; & (2n\sqrt{2})\cdot \min\left\{n!, \frac{1}{\hat{F}}\right\} \\ = \; & (2n\sqrt{2})N. \end{aligned}$$

Thus the average number of iterations is at most $(2n\sqrt{2})E[N] \le (8C\sqrt{2})n^{10}\lg n$ for all $n \ge 8$. ∎

7.2. The L_1 metric. In this subsection we bound the expected number of iterations of 2-opt under the L_1 norm. In contrast with Theorem 7.13 the proof is somewhat nonconstructive, but it is fairly short and the polynomial is of lower order.

THEOREM 7.14. *Let n points be independently sampled from the uniform distribution in the m-dimensional unit hypercube. Let 2-opting be performed with respect to the L_1 norm. Then the expected number of iterations required is $O(n^6 \log n)$.*

Proof. Let I^m denote the unit hypercube. Suppose v^1, v^2, v^3, v^4 are four points sampled independently from the uniform distribution on I^m. Let Z denote the random variable equal to $G(v^1, v^2, v^3, v^4) = ||v^1 - v^2||_1 + ||v^3 - v^4||_1 - ||v^1 - v^3||_1 - ||v^2 - v^4||_1$. Note that distances are computed according to the L_1 norm.

We study the distribution of Z. For clarity we focus on the case $m = 2$ and use remarks to extend the proof to the m-dimensional case.

The four points in I^2 are defined by eight $(4m)$ random variables, denoted $x_i, y_i : i = 1, \ldots, 4$, and drawn independently from the uniform distribution on $[0, 1]$. Let $g(x_1, \ldots, x_4) = |x_1 - x_2| + |x_3 - x_4| - |x_1 - x_3| - |x_2 - x_4|$. Then Z is the sum of the i.i.d. variables X and Y where $X = g(x_1, x_2, x_3, x_4)$ and $Y = g(y_1, y_2, y_3, y_4)$. The key is to understand the distribution of X, because $X + Y$ will have a distribution found as the convolution of two i.i.d. variables with this distribution. (In m dimensions, Z is the m-fold convolution of i.i.d. variables with this distribution.)

LEMMA 7.15. *With probability $1/3$ the variable $X = 0$; with probability $2/3$ the variable X is distributed according to a continuous density function $\bar{h}$ on $[-2, 2]$.*

Proof. Consider the conditional distribution of X, conditioned on the event $\pi_1 = \{x_1 \ge x_2 \ge x_3 \ge x_4\}$. Notice that $X = 2x_3 - 2x_2$ under this condition. Now if we take four samples i.i.d. from the uniform distribution and let W equal the difference between the third and second largest, then obviously W has a continuous density function. Therefore, X has continuous conditional density function conditioned on event π_1. We denote the conditional probability density by h_{π_1}.

By symmetry, for seven other π's we can make the same argument, and for each we get a continuous conditional probability density function h_{π_i}. For eight additional π_i's the conditional distribution is like $-W$, i.e., twice the difference between the second and third largest.

For the last eight π_i, the conditional distribution of X is degenerate with all its mass at zero. This occurs when the projections of the arcs on the x-axis overlap and have opposite orientations.

The unconditional density function of X is

$$h = \sum_{i=1}^{24} \frac{1}{24} h_{\pi_i}.$$

Therefore, X has a hybrid distribution. It has a mass point of weight $8/24 = 1/3$ at 0. The remaining $2/3$ of the mass has continuous density because each of the 16 contributing h_{π_i}'s is continuous. □

When the distribution of X is convolved with itself to get Z, the result is a hybrid distribution: $Z = 0$ with probability $1/9$; Z is distributed according to a continuous density, denoted $\bar{h}^2$, with probability $8/9$. This is because X and Y are independent. (In higher dimensions each additional convolution is of two independent hybrid distributions, each containing one mass point at zero, and continuous everywhere else, and these properties are obviously preserved in the sum of the distributions.) In m dimensions, the probability is $(1/3)^m$ that a given random 4-tuple of points has $Z = 0$; Z is distributed according to a continuous density, denoted $\bar{h}^m$, with probability $1 - (1/3)^m$.

A local improvement algorithm will not make any of the 2-changes corresponding to 4-tuples where $Z = 0$. The algorithm will be fast if there are no very small *positive* Z values.

Now consider the density function $\bar{h}^m$. It is continuous everywhere on $[-2m, 2m]$ and is symmetric around 0. Since a continuous function on a compact set attains its maximum, $\bar{h}^m$ has a maximum M. This implies that

$$P[0 < |Z| \le \epsilon] \le 2M\epsilon \quad \text{for all } \epsilon > 0.$$

Now let S denote a sample of n points on I^m. Let Δ denote the smallest of all nonzero absolute differences in tour length from 2-changes:

$$\Delta = \min_{\substack{v^1, v^2, v^3, v^4 \in S \\ G(v^1, v^2, v^3, v^4) \neq 0}} |G(v^1, v^2, v^3, v^4)|$$

(where the minimum is over distinct points). Since the minimum is taken over fewer than n^4 4-tuples, we get

$$P[\Delta < \epsilon] < n^4(2M\epsilon) = cn^4\epsilon.$$

By the same argument as in Lemma 7.12, with an n^4 instead of an n^8 term, we have $E[\min\{n!, 1/\Delta\}] \le 4cn^5 \log n$. The cost of the initial tour being at most $2n$, the expected number of iterations is at most $(2n)4cn^5 \log n$, which is $O(n^6 \log n)$. □

8. Extending Lueker's construction. In 1976, Lueker [15] constructed, for each $n \geq 4$, a family of n-node weighted cliques G_n on vertex set $\{x_0, x_1, x_2, \ldots, x_{n-1}\}$ with the property that the naive 2-opt algorithm can do at least $2^{\lfloor n/2 \rfloor - 2}$ iterations. Although the only thing we need from his construction is that each of the tours includes the edge $\{x_0, x_{n-1}\}$, we give Lueker's (unpublished) construction here: For each $0 \leq i < j \leq n-1$, define the weight w_{ij} of edge $\{x_i, x_j\}$ as follows.

1. j is even.
   (a) If i=0, then $w_{ij} = 2^{2j}$.
   (b) If i is positive and odd, then $w_{ij} = 2^{2i+3}$.
   (c) If i is positive and even, then $w_{ij} = 2^{2i}$.
2. j is odd.
   (a) If i=0, then $w_{ij} = 2^{2j+3}$.
   (b) If i is positive and odd, then $w_{ij} = 2^{2i+4}$.
   (c) If i is positive and even, then $w_{ij} = 2^{2i+3}$.

For $q \leq \lfloor n/2 \rfloor - 1$, let $a_q = \langle x_1, x_2, x_3, \ldots, x_{2q} \rangle$ and let a'_q be its reverse. Lueker proves the following theorem.

THEOREM 8.1. *Let T be a tour of G_n that contains the block $\langle x_0, a_q, x_{2q+1} \rangle$. Then there is a sequence of at least 2^{q-1} improving 2-changes which lead to the replacement of this block by $\langle x_0, a'_q, x_{2q+1} \rangle$ (and which leaves the rest of the tour unchanged).*

We refer to [15] for a proof, which converts the string $\langle x_0, a_q, x_{2q+1}, x_{2q+2}, x_{2q+3} \rangle$ to $\langle x_0, a'_q, x_{2q+1}, x_{2q+2}, x_{2q+3} \rangle$ in at least 2^{q-1} steps by induction, then to $\langle x_0, a'_q, x_{2q+2}, x_{2q+1}, x_{2q+3} \rangle$ in one step, then to $\langle x_0, x_{2q+1}, x_{2q+2}, a_q, x_{2q+3} \rangle$ in one step, then to $\langle x_0, x_{2q+1}, x_{2q+2}, a'_q, x_{2q+3} \rangle$ in at least 2^{q-1} steps by a clever use of the inductive assertion, and last to $\langle x_0, x_{2q+2}, x_{2q+1}, a'_q, x_{2q+3} \rangle$. Notice that, as claimed, edge $\{x_0, x_{n-1}\}$ is in all tours.

It is now our job to extend the result to all $k \geq 3$. Since $k \geq 4$ is easy, we do that case first.

THEOREM 8.2. *For any $k \geq 4$, for any $N \geq 2k$, there is an N-vertex weighted graph on which there exists a sequence of at least $2^{\lfloor N/2 \rfloor - k}$ improving k-changes.*

Proof. The idea is to take G_n and add $2(k-2)$ new vertices at distance one from each other and from the original vertices. Any 2-change in the original proof can be converted to a k-change in the new graph by flipping two original edges and $k-2$ new ones.

Let $l = k - 2$ and add to G_n $2l$ new vertices:

$$s, t, y_1, z_1, y_2, z_2, \ldots, y_{l-1}, z_{l-1}.$$

The distance between any pair of these vertices, as well as that between these new vertices and the n old ones, is 1. Let

$$A = \{\{y_1, z_1\}, \{y_2, z_2\}, \{y_3, z_3\}, \ldots, \{y_{l-1}, z_{l-1}\}\},$$

$$L = \{\{s, y_1\}, \{z_1, y_2\}, \{z_2, y_3\}, \ldots, \{z_{l-1}, t\}\},$$

and

$$R = \{\{s, z_1\}, \{y_1, z_2\}, \{y_2, z_3\}, \ldots, \{y_{l-1}, t\}\}.$$

The key fact is that $A \cup L$ is the edge set of a Hamiltonian path from s to t among the new vertices, $A \cup R$ is the edge set of a Hamiltonian path from s to t among the

new vertices, and A, L, R are pairwise disjoint. Furthermore, $A \cup L$ and $A \cup R$ differ in exactly l edges.

Now let S, T be the edge sets of any two tours of G_n that both contain edge $\{x_0, x_{n-1}\}$ and share exactly $n-2$ edges. Let $S' = (S - \{x_0, x_{n-1}\}) \cup (\{\{x_0, s\}, \{x_{n-1}, t\}\} \cup A \cup L)$ and $T' = (T - \{x_0, x_{n-1}\}) \cup (\{\{x_0, s\}, \{x_{n-1}, t\}\} \cup A \cup R)$. S' induces a tour of the new graph, T' induces a tour of the new graph as well, and S' and T' differ in exactly $2 + l = k$ edges. Since all the new edges have weight 1, if moving from S to T on the original graph is a cost-decreasing 2-change, then moving from S' to T' in the new graph is a cost-decreasing k-change. (In the next step, we can interchange the roles of the y's and z's, since they are symmetric.)

Lueker's proof gives a sequence of at least $2^{\lfloor n/2 \rfloor - 2}$ cost-decreasing 2-changes on an n-vertex graph (if $n \geq 4$). The new graph we built has $N = n + 2l = n + 2(k-2) = n + 2k - 4$ vertices, and there is a sequence of $2^{\lfloor n/2 \rfloor - 2}$ cost-decreasing k-changes. In terms of the number N of vertices in the new graph, the number of k-changes is at least $2^{\lfloor N/2 \rfloor - k}$. □

Now we tackle $k = 3$.

THEOREM 8.3. *For each even $N \geq 8$, there is an N-vertex weighted graph on which there exists a sequence of at least $2^{\lfloor N/4 \rfloor - 1}$ improving 3-changes.*

Proof. We use w_{ij} to denote the weight of $\{x_i, x_j\}$ in G_n. Let $w_{ii} = 0$ for all i. Take two copies of G_n. One copy has vertex set $V = \{x_0, \ldots, x_{n-1}\}$; the other has vertex set $V' = \{x'_0, \ldots, x'_{n-1}\}$. The weight of edge $\{x'_i, x'_j\}$, $i \neq j$, and that of $\{x_i, x'_j\}$ equals w_{ij}. Call the new $2n$-node graph H_n.

Let S be the edge set of any tour in G_n containing edge $\{x_0, x_{n-1}\}$. There is an obvious associated tour of H_n: Let $A = S - \{\{x_0, x_{n-1}\}\}$. Let $A^* = \{\{x'_i, x'_j\} | \{x_i, x_j\} \in A\}$. Then the associated tour contains edges $S' = \{\{x_0, x'_{n-1}\}, \{x'_0, x_{n-1}\}\} \cup A \cup A^*$. It is not hard to see that S' is the edge set of a tour in H_n.

We will prove the following. Let S be any tour of G_n containing $\{x_0, x_{n-1}\}$, and let S' be the associated tour of H_n. Let T be a tour of G_n obtained from S by one cost-decreasing 2-change, such that the 2-change does not involve the edge $\{x_0, x_{n-1}\}$. Then the tour T' of H_n associated with T can be obtained from S' via *two* cost-decreasing 3-changes. (We are implicitly identifying tours with their edge sets.)

Consider the tour S as directed from x_0 to x_{n-1}. Suppose that the 2-change involves swapping the edges $\{x_i, x_j\}, \{x_k, x_l\}$, where, in S, the vertex x_i is the first among the four visited by S, x_j is second, x_l is third, and x_k is fourth. Since the 2-change results in a tour T, it must replace the two missing edges by $\{x_i, x_l\}, \{x_k, x_j\}$. (If they were replaced by $\{x_i, x_k\}, \{x_l, x_j\}$, then the new "tour" would be disconnected.)

Let S' and T' be the tours of H_n associated with S and T, respectively. S' and T' differ only in that to get from S' to T' one drops the four edges $\{x_i, x_j\}$, $\{x'_i, x'_j\}$, $\{x_k, x_l\}$, $\{x'_k, x'_l\}$ and adds $\{x_i, x_l\}$, $\{x'_i, x'_l\}$ and $\{x_k, x_j\}$, $\{x'_k, x'_j\}$.

We know that switching from S to T decreased the cost; thus $(w_{ij} + w_{kl}) - (w_{il} + w_{kj}) > 0$. In S', we have the four edges $\{x_i, x_j\}$, $\{x_k, x_l\}$, $\{x'_i, x'_j\}$, $\{x'_k, x'_l\}$. We leave the last one unchanged and change the first three to $\{x_i, x_l\}$, $\{x_k, x'_j\}$, $\{x'_i, x_j\}$. It is easy to see that the new "tour" is indeed a tour, so this is a valid 3-change, provided that we have decreased the cost. The decrease in cost is

$$(w_{ij} + w_{kl} + w_{ij}) - (w_{il} + w_{kj} + w_{ij}) = (w_{ij} + w_{kl}) - (w_{il} + w_{kj}),$$

which we know to be positive. The next 3-change leaves edge $\{x_i, x_l\}$ unchanged. It flips $\{x_k, x'_j\}, \{x'_i, x_j\}, \{x'_k, x'_l\}$ to $\{x_k, x_j\}$, $\{x'_i, x'_l\}$, $\{x'_k, x'_j\}$. The decrease in cost is

$$(w_{kj} + w_{kl} + w_{ij}) - (w_{il} + w_{kj} + w_{kj}) = (w_{ij} + w_{kl}) - (w_{il} + w_{kj}),$$

which we know to be positive. The resulting edge set contains $\{x_k, x_j\}$, $\{x'_k, x'_j\}$, $\{x_i, x_l\}$, $\{x'_i, x'_l\}$ and is otherwise the same as S'; thus it is T' and therefore a tour.

Lueker's proof gave $2^{\lfloor n/2 \rfloor - 2}$ 2-changes on an n-vertex graph, $n \geq 4$. We have $N = 2n$, and we make two 3-changes for each original 2-change. In terms of N, we have $2^{\lfloor N/4 \rfloor - 1}$ 3-changes if $N \geq 8$ is even. □

9. Open problems.

- One of the best TSP algorithms in actual experiments [2] is the Lin–Kernighan algorithm [13], a local search algorithm with a more complex neighborhood structure. Since a Lin–Kernighan optimal tour is also 2-optimal, all the upper bounds on the performance ratio of 2-opt also hold for Lin–Kernighan. Can one do better for Lin–Kernighan?
- Our lower bounds on the performance ratio of k-opt are obtained by showing that there is some k-optimal tour of large weight. Suppose that we start with a random tour and then deterministically make improving k-changes. Can we get better performance guarantees?
- Can Lueker's results be extended to the Euclidean plane; i.e., is there a graph in the Euclidean plane for which there exists an exponential number of improving 2-changes?
- Can Theorem 3.4 be generalized to any k-opt algorithm; i.e., for arbitrary metric spaces can it be proved that, as k increases, the performance guarantee of the k-opt algorithm improves?

Acknowledgments. We thank Jim Dai, Bob Foley, Tony Hayter, Mark Krentel, Mike Todd, David Shmoys, and Sundar Vishwanathan for helpful discussions on the material of the paper and George Lueker for kindly permitting us to include his construction in section 8. We also thank the two anonymous referees for their remarks.

REFERENCES

[1] N. Alon and Y. Azar, *On-line Steiner trees in the Euclidean plane*, in Proc. 8th ACM Symposium on Computational Geometry, Berlin, 1992, pp. 337–343.
[2] J. L. Bentley, Invited talk at the 1st SIAM Symposium on Discrete Algorithms, San Francisco, 1990.
[3] J. L. Bentley and J. Saxe, *An analysis of two heuristics for the Euclidean traveling salesman problem*, in Proc. 18th Allerton Conference on Communication, Control and Computing, Urbana-Champaign, IL, 1980, pp. 41–49.
[4] B. Bollobás, *Extremal Graph Theory*, Academic Press, London, 1978.
[5] B. Chandra, G. Das, G. Narasimhan, and J. Soares, *New sparseness results on graph spanners*, in Proc. 8th ACM Symposium on Computational Geometry, Berlin, 1992, pp. 192–201.
[6] L. K. Grover, *Local search and the local structure of NP-complete problems*, Oper. Res. Lett., 12 (1992), pp. 235–243.
[7] D. J. Johnson, C. H. Papadimitriou, and M. Yannakakis, *How easy is local search?*, J. Comput. System Sci., 37 (1988), pp. 79–100.
[8] R. M. Karp and J. M. Steele, *Probabilistic analysis of heuristics*, in The Traveling Salesman Problem, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds., John Wiley and Sons, New York, 1985, pp. 181–205.
[9] W. Kern, *A probabilistic analysis of the switching algorithm for the Euclidean TSP*, Math. Programming, 44 (1989), pp. 213–219.

[10] M. W. Krentel, *Structure in locally optimal solutions*, in Proc. 30th Symposium on Foundations of Computer Science, Research Triangle Park, NC, 1989, pp. 216–221.
[11] M. W. Krentel, personal communication, Rice University, Houston, TX.
[12] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds., *The Traveling Salesman Problem*, John Wiley and Sons, New York, 1985.
[13] S. Lin and B. W. Kernighan, *An effective heuristic for the traveling salesman problem*, Oper. Res., 21 (1973), pp. 489–516.
[14] L. Loomis and S. Sternberg, *Advanced Calculus*, Addison-Wesley, Reading, MA, 1968.
[15] G. Lueker, Unpublished manuscript, Princeton University, Princeton, NJ, 1975.
[16] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.
[17] C. A. Rogers, *Covering a sphere with spheres*, Mathematika, 10 (1963), pp. 157–164.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2030–2058

PARALLEL COMPLEXITY OF NUMERICALLY ACCURATE LINEAR SYSTEM SOLVERS*

MAURO LEONCINI†, GIOVANNI MANZINI‡, AND LUCIANO MARGARA§

Abstract. We prove a number of negative results about practical (i.e., work efficient and numerically accurate) algorithms for computing the main matrix factorizations. In particular, we prove that the popular Householder and Givens methods for computing the QR decomposition are P-complete, and hence presumably inherently sequential, under both real and floating point number models. We also prove that Gaussian elimination (GE) with a weak form of pivoting, which aims only at making the resulting algorithm nondegenerate, is likely to be inherently sequential as well. Finally, we prove that GE with partial pivoting is P-complete over GF(2) or when restricted to symmetric positive definite matrices, for which it is known that even standard GE (no pivoting) does not fail. Altogether, the results of this paper give further formal support to the widespread belief that there is a tradeoff between parallelism and accuracy in numerical algorithms.

Key words. P-complete problems, parallel complexity, NC algorithms, inherently sequential algorithms, matrix factorization, numerical stability

AMS subject classifications. 68Q22, 68Q25, 65F05

PII. S0097539797327118

1. Introduction.

Matrix factorization algorithms form the backbone of many numerical libraries, such as LINPACK and LAPACK [7, 2]. They are also available as primitive routines in state-of-the-art scientific computing environments like MATLAB [17]. Indeed, factoring a matrix is almost always the first step of many scientific computations, and usually the one which places the heaviest demand in terms of computing resources. Some authors have investigated the parallel complexity of the most popular and important matrix factorizations, namely, the $(P)LU$ and $QR(\Pi)$ decompositions (see section 2 for definitions and simple properties). A list of positive known results follows.

1. LU decomposition is in arithmetic NC, whenever it exists, i.e., provided that the leading principal minors of the input matrix are nonsingular (in this case we will say that the matrix is *strongly nonsingular*) [19, 21].

2. QR decomposition is in arithmetic NC for matrices with full column rank, since it easily reduces to LU decomposition of strongly nonsingular matrices [19].

3. PLU decomposition is in arithmetic NC for nonsingular matrices [8]. The algorithm for finding a permutation matrix P such that P^TA is strongly nonsingular

*Received by the editors September 10, 1997; accepted for publication July 5, 1998; published electronically June 23, 1999. This work merges and extends preliminary results that appeared as *Parallel complexity of Householder QR factorization*, in Proc. European Symp. on Algorithms, Lecture Notes in Comput. Sci. 1136, Springer-Verlag, New York, 1996, pp. 290–301, and *On the parallel complexity of matrix factorization algorithms*, in Proc. 9th ACM Symp. on Parallel Algorithms and Architectures, ACM, New York, 1997, pp. 63–71.

http://www.siam.org/journals/sicomp/28-6/32711.html

†Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56125 Pisa, Italy and IMC-CNR, Via S. Maria 46, I-56126 Pisa, Italy (leoncini@di.unipi.it). The work of this author was supported by Murst 40% funds.

‡Dipartimento di Scienze e Tecnologie Avanzate, Università del Piemonte Orientale, Via Cavour 84, I-15100 Alessandria, Italy and IMC-CNR, Via S. Maria 46, I-56126 Pisa, Italy (manzini@mfn.al.unipmn.it). The work of this author was supported by Murst 40% and 60% funds.

§Dipartimento Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, I-40127 Bologna, Italy (margara@cs.unibo.it).

builds upon the computation of the lexicographically first maximal independent subset (LFMIS) of the rows of a matrix, which is in NC^2 [3].[1]

4. $QR\Pi$ factorization of an arbitrary matrix A is in arithmetic NC [8]. A permutation Π such that the leftmost $n \times r$ submatrix of A has full column rank, for $r = \text{rank}(A)$, can be found by computing the LFMIS of sets of (column) vectors.

Unfortunately, none of the above algorithms seems to be of practical worth. Except for [21], they boil down to fast parallel determinant or inverse matrix computations (see, e.g., [5, 9]), and the few experiments and theoretical analyses that have been performed indicate that these are very unstable in general [4, 6, 22]. Moreover, the analysis in [4] of Reif's LU factorization algorithm [21] suggests that the latter can be highly unstable as well. Indeed, finding a numerically stable NC algorithm to compute the LU (or QR) decomposition of a matrix can be regarded as one important open problem in parallel computation theory [11].

That a positive solution to the above problem may not be just around the corner is confirmed by the negative complexity results that can be proved for the *stable* algorithms adopted in practice to compute the LU and QR decompositions, namely, Gaussian elimination (GE) and the Householder and Givens methods (HQR and GQR, respectively).

In 1989 Vavasis proved that GE with partial pivoting (GEP), which is the standard method for computing the PLU decomposition, is P-complete over the reals or the rationals [24]. Note that, strictly speaking, membership in P could not be defined for the real number model. When dealing with real matrices and real number computations, we then assume implicitly that the class P be defined to include the problems solvable in polynomial time on such models as the real RAM (see [20]). The result in [24] was proved by showing that a decision problem defined in terms of GEP's behavior was P-complete. For parallel complexity theory the P-completeness result implies that GEP is likely to be inherently sequential, i.e., admitting no NC implementations unless $P = NC$. Leoncini proved then that GEP is probably even harder to parallelize, in the sense that no $O(n^{\frac{1}{2}-\epsilon})$ time PRAM implementation can exist unless all the problems in P admit polynomial speedup [13].

In this paper we prove new negative results for the classical factorization algorithms. We consider the methods of Householder's reflections and of Givens' rotations to compute the QR decomposition of a matrix. The main use of the QR decomposition is within iterative methods for the computation of the eigenvalues of a matrix and to compute least squares solutions to overdetermined linear systems [10]. Moreover, even though QR decomposition is usually not adopted to solve systems of linear equations on uniprocessor machines, both HQR and GQR are potential competitors of GE to solve linear systems stably in parallel. To date, the fastest stable parallel solver is based on GQR and is characterized by $O(n)$ parallel time on an $O(n^2)$ processor PRAM [23], while both GEP and HQR run in $O(n \log n)$ time with $O(\frac{n^2}{\log n})$ processors. Also, GQR is especially suitable for solving large sparse systems, given its ability to annihilate selected entries of the input matrix at very low cost.

We also consider the application of GEP to special classes of matrices and study a weaker form of pivoting which we will call *minimal* pivoting (GEM). Under minimal pivoting, the pivot chosen to annihilate a given column is the first nonzero on or below the main diagonal. Minimal pivoting is especially suitable for systolic-like implementations of linear system solvers (see, e.g., [12], although it is not referred

[1] Not to be confused with the analogous LFMIS problem of graph theory, which is known to be P-complete [11].

to by this name). Minimal pivoting can be regarded as the minimum modification required for GE to be nondegenerate on arbitrary input matrices. Finally, we consider GEP over GF(2), where it clearly coincides with GEM.

The results we obtain are listed below.

1. We prove that HQR and GQR are P-complete over the real or floating point numbers. We exhibit reductions from the NAND circuit value problem (NANDCVP) with fanout ≤ 2. In particular, what we prove to be P-complete is deciding the sign of a given diagonal element of the upper triangular matrices R computed by either HQR or GQR. Our reductions seem to be more intricate than the simple one in [24]. This is probably a consequence of the apparently more complex effect of reflections and rotations with respect to the linear combinations of GE. We would like to stress that the P-completeness proofs for the case of floating point arithmetic apply directly to, and have been checked with, the algorithms available in MATLAB using the IEEE 754 standard for floating point arithmetic. In other words, the negative results apply to widely "in-use" software.

2. We extend Vavasis' result proving that GEP is P-complete on input strongly nonsingular matrices. This class includes matrices which naturally arise in practical applications, namely, diagonally dominant and symmetric positive definite matrices. Note that plain GE (no pivoting) is guaranteed not to fail on input a strongly nonsingular matrix. However, since it is usually unstable, one still uses GEP.

3. We prove that GEM is P-complete on input nonsingular matrices. By this result we also obtain a different proof that GEP is P-complete. In fact, GEP and GEM compute the same factorization on input the special class of matrices that correspond to fanout 2 nand circuits.

4. We prove that GEP is P-complete over GF(2). The general status (P-complete or NC-computable) of GEP over finite fields is one of the open problems in [11].

5. We show that a known NC algorithm for computing a PLU decomposition of a nonsingular matrix corresponds to GE with a nonstandard pivoting strategy which differs only slightly from minimal pivoting. Also, we prove that GE with such a nonstandard strategy is P-complete on input arbitrary matrices, which somehow accounts for the difficulty of finding an NC algorithm to compute the PLU decomposition of possibly singular matrices.

The completeness results have been proved by using a general framework for reducing circuit computations to matrix factorization algorithms with a common structural kernel. The development of such a framework is by itself a major contribution of this paper.

The results presented here give further evidence of the pervasiveness of a phenomenon that also has been observed by numerical analysts (from a more practical perspective)—namely, that there is a "tradeoff" between the degree of parallelism, on the one hand, and nondegeneracy and accuracy properties of numerical algorithms, on the other [6].

The rest of this paper is organized as follows. In section 2 we introduce a little notation and give some preliminary definitions. In section 3 we describe the general framework for proving the P-completeness of the factorization algorithms. In sections 4 and 5 we address QR decomposition via Householder's reflections and Givens' rotations, respectively. In section 6 we prove our negative results for GE with partial and minimal pivoting. In section 7 we show a correspondence between a known NC PLU decomposition algorithm and GE. We conclude with some further considerations and

open problems. In Appendix A we describe the factorization algorithms considered in this paper. In Appendix B we give some basic definitions about the floating point number representation. More material about the algorithms and the computer arithmetic can be found in many excellent textbooks (in particular, see [10]). Finally, we include one technical proof in Appendix C.

2. Preliminaries. We use standard notation for matrices and matrix-related concepts (see [10]). Matrices are denoted by capital letters. The (i, j) entry of a matrix A is referred to by either a_{ij} or $[A]_{ij}$. Vectors are designated by lowercase letters, usually taken from the end of the alphabet, e.g., x, y, etc. Note that the notation x refers to a *column* vector, i.e., an $n \times 1$ matrix, for some $n \geq 1$.

The ith row (resp., column) of a matrix A is denoted by a_{i*} (resp., a_{*i}). A *minor* of a matrix A is any submatrix of A. A *principal* minor is any square submatrix of A defined by the same set of row and column indices.

The symbols I and O denote the identity matrix (with $[I]_{ij} = 1$ if $i = j$ and 0 otherwise) and the zero matrix (such that $[O]_{ij} = 0$), respectively. The zero vector is denoted using the symbol 0. The transpose of A is the matrix B such that $b_{ij} = a_{ji}$. B is denoted by A^T. A matrix Q is orthogonal when $Q^T Q = I$. A permutation matrix P is a matrix which is zero everywhere except for just one 1 in each row and column. Any permutation matrix is orthogonal. The transpose of a (column) vector x is the *row* vector x^T, i.e., a matrix of size $1 \times n$, for some n.

Let A be a square matrix of order n.

(i) The LU decomposition of A is a pair of matrices $\langle L, U \rangle$, such that L is lower triangular (i.e., $l_{ij} = 0$ for $j > i$) with unit diagonal elements, U is upper triangular, and $A = LU$. For an arbitrary (even nonsingular) matrix A the LU decomposition might not be defined. A sufficient condition for its existence (and uniqueness) is that A be strongly nonsingular.

(ii) The PLU decomposition of A is a triple of matrices $\langle P, L, U \rangle$ such that L and U are as above, P is a permutation matrix, and $P^T A = LU$ (or, equivalently, $A = PLU$). The PLU decomposition is always defined but not unique.

(iii) The QR decomposition of A is a pair of matrices $\langle Q, R \rangle$, such that Q is orthogonal, R is upper triangular, and $A = QR$. The QR decomposition always exists.

In all the above cases, if A is $m \times n$, with $m < n$, and when the factorization exists, we get a matrix that is properly said to be in *row echelon* form (rather than upper triangular). Its leftmost $m \times m$ minor is upper triangular while its rightmost $m \times (n - m)$ minor is, in general, a dense submatrix. However, when no confusion is possible, we will always speak of the triangular factor of a given factorization.

A detailed description of the algorithms considered in this paper can be found in Appendix A. The details, however, are not necessary to understand the common structure of the reductions. Some details will be required in the proofs of Theorems 4.3 and 5.3, which deal with the floating point version of the QR algorithms. Except for these, the following general description is sufficient. It defines a class $\mathcal{F}$ of matrix factorization algorithms that includes, among others, the classical QR algorithms and GE.

Let A be the input matrix. The algorithms in $\mathcal{F}$ take A to upper triangular form by applying a series of transformations that introduce zeros in the strictly lower triangular portion of A, from left to right. The notation $A^{(k)}$ is usually adopted to indicate the matrix obtained after $k-1$ transformations, and its elements are referred to by $a_{ij}^{(k)}$. $a_{ij}^{(k)}$ is zero for $j < \min\{i, k\}$. A transformation is applied during one *stage*

of the algorithm.

Every algorithm $\mathcal{A} \in \mathcal{F}$ satisfies the following properties.

p1. If $a_{ij} = 0$, for $j = 1, \ldots, s$, then $a_{i*}^{(k)} = a_{i*}$, $k = 1, \ldots, s$. In other words, if the first s entries in row i are zero, then the first s stages of $\mathcal{A}$ do not modify row i.

p2. If column k has a complementary nonzero structure with respect to columns $j = 1, \ldots, k-1$ (by which we mean $a_{ij} \neq 0 \Rightarrow a_{ik} = 0$ for any i), then $a_{*k}^{(j)} = a_{*k}$, i.e., column k is not affected by the first $k-1$ transformations.

p3. This is a property that we will call *proper embedding* of a matrix A into a larger matrix E. Let A be of order k and let R be the triangular factor computed by $\mathcal{A}$ on input A. Also, let E be a matrix having a minor $E_A = A$. Suppose that, as a consequence of the repeated applicability of **p1** and **p2**, the first $k-1$ stages of algorithm $\mathcal{A}$ on input E affect only E_A and possibly some columns with higher indices. Then $E^{(k)}$ contains R as the minor corresponding to E_A. In other words, the factorization of A, viewed as a part of E, is the same as the factorization of A alone.

p4. Stage k modifies the entry (i, j) only if $i, j \geq k$. In particular, it introduces zeros in the kth column of $A^{(k-1)}$ without destroying the previously introduced zeros. In view of this, and to avoid some redundant descriptions, in the rest of this paper we will use the notation $A^{(k)}$ to indicate the submatrix of $A^{(k)}$ with elements from $a_{kk}^{(k)}$ rightward and downward.

3. A framework for reductions to $\mathcal{F}$. Our P-completeness results are all based on reductions from the NANDCVP, a restricted version of the CVP (circuit value problem) which we now briefly recall.

> *Input*: the description of a k-input Boolean circuit C composed entirely of fanin 2 nand gates, and Boolean values $x_1, \ldots, x_k$.
>
> *Output*: the value $C(x_1, \ldots, x_k)$ computed by C on input $x_1, \ldots, x_k$.

NANDCVP is P-complete, as reported in [11]. In order to simplify the proofs we will further assume, without loss of generality, that each gate of C has fanout at most 2. We shall prove in this section the following general result, which applies to any factorization algorithm $\mathcal{A} \in \mathcal{F}$:

> There is an encoding scheme of logical values and a logspace bounded transducer M with the following properties: given the description of a fanout 2 nand circuit C and Boolean inputs $x_1, \ldots, x_k$ for C, M builds a matrix A_C of order n_C such that, if $A_C = XR$ is the factorization computed by algorithm $\mathcal{A}$, with R upper triangular, then $[R]_{n_C, n_C}$ is the encoding of $C(x_1, \ldots, x_k)$.

We shall prove (Theorem 3.1) that the transducer does exist provided that there are certain elementary matrices with well-defined properties. We will later show that such matrices actually exist for the algorithms considered in this paper.

Unfortunately, a formal description and the proof of correctness of the transducer will require quite a large amount of details. In spite of this, the idea behind the construction is easy. Hence we first describe the reduction in an informal way and only afterward proceed to a formal derivation. Moreover, we have actually implemented the transducer as a collection of MATLAB m-files. These and the elementary matrices for the floating point versions of HQR and GQR, and of GEM as well (whose real and floating point versions are coincident), are available electronically [16].

3.1. Informal description. Let $\mathcal{A} \in \mathcal{F}$ and let $\mathbf{a}$ and $\mathbf{b}$ denote appropriate numerical encodings of arbitrary truth values a, b. We need three kinds of (square) elementary matrices for $\mathcal{A}$ to simulate basic logical operations.

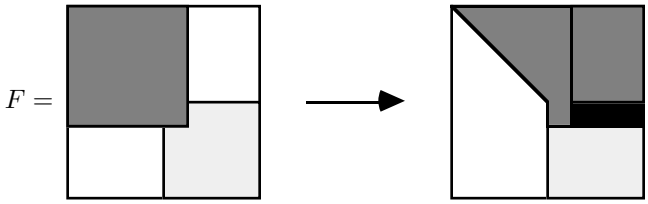


FIG. 1. *N-D matrix composition: effect of the first $\nu - 1$ stages.*

The first such matrix is the *nand* N. It has $[N]_{11} = \mathbf{a}$ and $[N]_{22} = \mathbf{b}$ and, if we apply $\mathcal{A}$ to compute the factorization of N, we get the encoding of $\texttt{NAND}(a, b)$ in the right bottom entry of the upper triangular factor.

The second elementary matrix is the *duplicator* D. It has $[D]_{11} = \mathbf{a}$. Let d be the order of D. Then the application of $d - 2$, rather than $d - 1$, stages of $\mathcal{A}$ to D is sufficient to triangularize D. The trailing 2×2 principal minor of the triangular factor is $\left(\begin{smallmatrix} \mathbf{a} & 0 \\ 0 & \mathbf{a} \end{smallmatrix}\right)$.

The third elementary matrix is the *copier* or *wire* W. It has $[W]_{11} = \mathbf{a}$, and if we compute the factorization of W we get $\mathbf{a}$ in the right bottom entry of the triangular factor.

Using these matrices as the building blocks we can construct a matrix A_C that simulates the circuit C. The structure of A_C is close to block diagonal, with one N block for each nand gate in the circuit C. Duplicator blocks are used to simulate fanout 2 nand gates, and wire blocks to route the computed values according to the circuit's structure. As the factorization of a block diagonal matrix could be performed by independently factoring the single blocks, a certain degree of overlapping between the blocks is necessary to pass around the computed values. The key idea is to repeatedly use the proper embedding property of the factorization algorithms.

To illustrate how the preceding scheme can work in practice, and to see where the difficulties may appear, consider first the construction of a submatrix which simulates a fanout 2 nand gate. The basic idea is to append a duplicator to a nand block as shown in Figure 1 (left). The dark gray area is a minor F_N of F such that $F_N = N$. Analogously, the light gray area is a minor F_D equal to D (in principle). Finally, the white zones contain zeros. The right bottom entry of F_N coincides with the top left entry of F_D. This is an example of proper embedding, for, if N has order ν, the first $\nu - 1$ steps affect only F_N (and the first ν entries of the columns with indices greater than the indices of F_N). This is a consequence of property **p1**. However, after $\nu - 1$ stages of $\mathcal{A}$ the encoding of $\texttt{NAND}(a, b)$ is exactly where required, i.e., in the top left entry of F_D, from where it can be duplicated. Note, however, that the whole first row of F_D has possibly been modified (i.e., the black-colored entries of Figure 1 (right)). Hence, for the simulation to proceed correctly, it is required that the black entries store the rest of the first row of D *by the time the algorithm reaches stage* ν. We cannot rely on their initial contents.

As a second example, Figure 2 describes how a W block, of order w, can be used to pass a value to a possibly far away place. The dark gray area represents a minor of T equal to a W block. Note that the minor is split across nonconsecutive rows and columns, namely, those with indices 1 through $w - 1$ and the one with index t (t = order of T). As before, the white zones contain zeros, while the light gray area is of arbitrary size and stores arbitrary values. This situation again represents

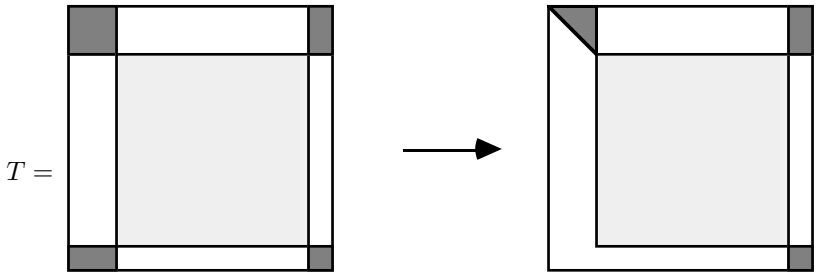


FIG. 2. *Routing of a logical value.*

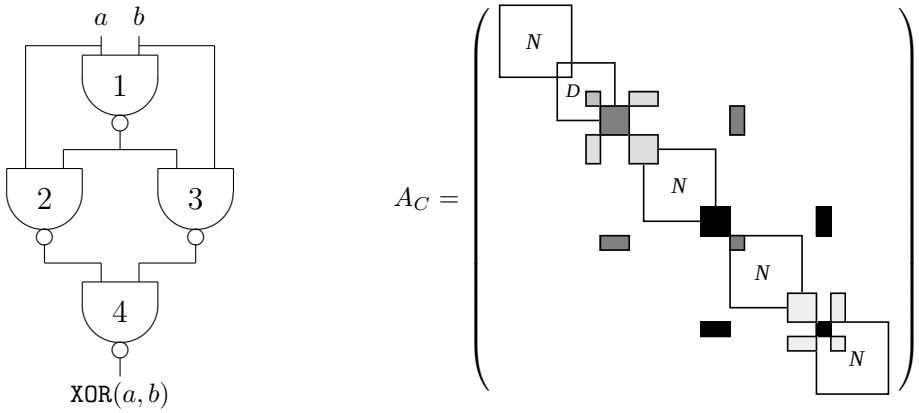


FIG. 3. *Circuit C computing* $\texttt{XOR}(a, b)$ *(left) and the structure of the corresponding matrix A_C (right).*

a proper embedding, so that the first $w-1$ stages of $\mathcal{A}$ on input T will result in the factorization of the W block. This implies that (the encoding of) the logical value initially in the top left entry has been copied to the right bottom entry.

To get an idea of what a complete matrix might looks like, see Figure 3, where the circuit C computing the *exclusive or* of two bits is considered. The corresponding matrix A_C has four N blocks, one D block, and four W blocks denoted by different gray levels. Note, however, that Figure 3 does not yet incorporate the solution to the "black entries" problem mentioned above.

3.2. Elementary matrices. To prove the correctness of the transducer (see Theorem 3.1 below) it is convenient to introduce a block partitioning of the elementary matrices defined in the previous section. Let $E \in \{N, D, W\}$ denote one such matrix. We partition E as

$$E = \begin{pmatrix} E_I & \bar{E}_I & \hat{E}_I \\ \bar{E}_M & E_M & \hat{E}_M \\ \bar{E}_O & \hat{E}_O & E_O \end{pmatrix}, \tag{1}$$

where the diagonal blocks E_I, E_M, and E_O are square matrices and E_I and E_O are also diagonal (i.e., with nonzero entries only on the main diagonal). We will refer to E_I and E_O as the *input* and *output* submatrices and let i and o denote their orders, respectively. Note that an elementary matrix actually defines a set of matrices. In fact,

we regard the i diagonal entries of E_I as the *input places*. A particular elementary matrix is obtained by filling the input place(s) with the encoding of some logical value(s).

We can now formally define the "behavior" of the elementary matrices with respect to the factorization algorithm $\mathcal{A}$.

Nand matrix (N). We let ν denote the order of N, which has $i = 2$ and $o = 1$ in the block partitioning (1), and set $[N]_{11} = \mathbf{a}$, $[N]_{22} = \mathbf{b}$. If $XN = R$ is the factorization computed by $\mathcal{A}$, we have

$$R = \begin{pmatrix} * & \dots & \dots & * \\ & \ddots & & \vdots \\ & & * & * \\ & & & \mathbf{c} \end{pmatrix},$$

where $\mathbf{c}$ is the encoding of $\texttt{NAND}(a, b)$. If, as is often required in practice, R overwrites the input matrix, the value $\mathbf{c}$ will replace N_O, and this is the reason why we defined E_O in (1) as the output submatrix. The same remark applies to the other elementary matrices. We also require that, for any real value x, an *auxiliary vector* $a_N = a_N(x) = (0, 0, a_3, \dots, a_\nu)^T$ can be defined such that $Xa_N = (*, \dots, *, x)^T$. Using the auxiliary vectors we can solve the "black entries" problem outlined in section 3.1. Intuitively, by appending auxiliary vectors to the right of N in A_C (instead of simply zeros, as in Figure 1 (left)), we can obtain the desired values in the black-colored entries of Figure 1 (right). As we will see in the proof of Theorem 3.1, the initial zeros in the auxiliary vector prevent the problem from pumping up in the construction of A_C.

Duplicator matrix (D). Let d denote the order of D, which has $i = 1$ and $o = 2$ in the block partitioning (1), and set $[D]_{11} = \mathbf{a}$. If $XD = R$ is the incomplete factorization computed by $\mathcal{A}$, i.e., X represents the first $d-2$ transformations applied to D by $\mathcal{A}$, we obtain

$$R = \begin{pmatrix} * & \dots & \dots & \dots & * \\ & \ddots & & & \vdots \\ & & * & \dots & * \\ & & & \mathbf{a} & 0 \\ & & & 0 & \mathbf{a} \end{pmatrix}.$$

We also require that, for any pair of real numbers x and z, an auxiliary vector $a_D = a_D(x, z) = (0, a_2, \dots, a_d)^T$ can be defined such that $Xa_D = (*, \dots, *, x, z)^T$.

Wire matrix (W). Let w denote the order of W, which has $i = o = 1$ in the block partitioning (1), and set $[W]_{11} = \mathbf{a}$. If $XW = R$ is the factorization computed by $\mathcal{A}$, we get

$$R = \begin{pmatrix} * & \dots & \dots & * \\ & \ddots & & \vdots \\ & & * & * \\ & & & \mathbf{a} \end{pmatrix}.$$

We also require that, for any real number x, an auxiliary vector $a_W = a_W(x) = (0, a_2, \dots, a_w)^T$ can be defined such that $Xa_W = (*, \dots, *, x)^T$.

As we shall see, elementary matrices (including auxiliary vectors) exist for both Householder's and Givens' methods and for GE as well.

3.3. Construction and correctness. In this section we present our main result (Theorem 3.1) on the existence of a single reduction scheme that works for any algorithm in $\mathcal{F}$. We still require some definitions.

Suppose that C has k input variables and let $k' \geq k$ be the number of places (i.e., inputs to the gates) where the variables are used. k' is the number of inputs counting multiplicities. Let the gates of C be sorted in topological order. Given a specific assignment of logical values to the input variables, we may then refer to the ith *actual input* as the ith value from the input set that is required at some gate, $1 \leq i \leq k'$.

When we say that A_C *has an input row at position* i, we mean that, initially, row i is either

$$\left(\overbrace{0, \ldots, 0}^{i-1}, \mathbf{a}, 0, [\bar{N}_I]_{1*}, [\hat{N}_I]_{1*}, 0^T \right) \tag{2}$$

or

$$\left(\overbrace{0, \ldots, 0}^{i-2}, 0, \mathbf{a}, [\bar{N}_I]_{2*}, [\hat{N}_I]_{2*}, 0^T \right), \tag{3}$$

where $\mathbf{a}$ is the encoding of one of the actual inputs to C.

THEOREM 3.1. *Let elementary matrices N, D, and W be given for $\mathcal{A} \in \mathcal{F}$. For any fanout 2 nand circuit C with input variables $x_1, \ldots, x_k$ and any truth assignment to $x_1, \ldots, x_k$, we can build a square matrix A_C such that the following hold:*

(a) *A_C has order $n_C = O(n)$, where n is the number of gates in C.*

(b) *If $A_C = XR$ is the factorization computed by $\mathcal{A}$, then $[R]_{n_C,n_C}$ is the encoding of $C(x_1, \ldots, x_k)$.*

(c) *A_C has a number of input rows which equals the number $k' \geq k$ of actual inputs to C; the ith such row has either the structure* (2) *or* (3) *depending on whether the ith actual input to C enters the first or the second input of some gate.*

(d) *Any actual input affects A_C only through one input place.*

The construction can be done by using $O(\log n)$ work space.

Proof. We prove the result by induction on n. Let $z_1, \ldots, z_{k'}$, for $k' \geq k$, be the actual inputs to C and let $\mathbf{a}$ and $\mathbf{b}$ be the encodings of z_1 and z_2, respectively. The case $n = 1$ is easy. We only have to set $A_C = N$, with $[A_C]_{11} = \mathbf{a}$ and $[A_C]_{22} = \mathbf{b}$. Clearly $n_C = O(1)$. Property (b) follows from the definition of N. Properties (c) and (d) are easily verified as well. In particular, A_C has exactly two input rows at positions 1 and 2, with structure (2) and (3), respectively, and this clearly matches the number of actual inputs to C. Finally, the actual inputs affect A_C only through the input places $[A_C]_{11}$ and $[A_C]_{22}$.

Now suppose that the number of gates in C is $n > 1$, and let $g_1, \ldots, g_n$ be a topological ordering of the DAG representing C. Clearly, all the inputs to g_1 are actual inputs to C. Let C' be the circuit with g_1 removed and any of its outputs replaced with x_1, the first input variable. Since C' has $n - 1$ gates, we may assume that $A_{C'}$ can be constructed which satisfies the induction hypothesis. To build A_C we simply extend $A_{C'}$ to take g_1 into account. There are two cases, depending on the fanout of g_1. We fully work out only the case of fanout 1. The other case is similar but is more tedious to develop fully (and for the reader to follow).

1. Let g_h be the gate connected to the output of g_1. Suppose, without loss of generality, that g_1 provides the first input to g_h. By the induction hypothesis (in particular, by (c)) $A_{C'}$ has the following structure:

$$A_{C'} = \begin{array}{c} \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccc} & & i_h & & & \\ & & \downarrow & & & \end{array} \\ \begin{pmatrix} X_1 & x_1 & x_2 & X_2 & X_3 & X_4 \\ 0^T & \mathbf{a} & 0 & [\bar{N}_I]_{1*} & [\hat{N}_I]_{1*} & 0^T \\ y_1^T & 0 & \gamma & y_2^T & y_3^T & y_4^T \\ Z_1 & z_1 & z_2 & Z_2 & Z_3 & Z_4 \end{pmatrix} \end{array} \begin{array}{l} \\ \\ \leftarrow i_h. \\ \\ \end{array}$$

$A_{C'}$ has an input row at some position i_h corresponding to the first input to gate g_h and $\mathbf{a}$ is the encoding of x_1. Note that, by property (d), the actual logical value encoded by $\mathbf{a}$ affects only the definition of $A_{C'}$ through the entry (i_h, i_h). Using $A_{C'}$ and N and W elementary matrices we define A_C as follows:

$$A_C = \begin{pmatrix} N_I & \bar{N}_I & \hat{N}_I & \boxed{O} & O & \boxed{0} & 0 & O & O & O \\ \bar{N}_M & N_M & \hat{N}_I & \boxed{A_1} & O & \boxed{a_1'} & 0 & O & O & O \\ \bar{N}_O & \hat{N}_O & N_O & \boxed{a_1^T} & 0^T & \boxed{\alpha} & 0 & 0^T & 0^T & 0^T \\ O & O & \bar{W}_M & W_M & O & \hat{W}_M & 0 & A_2 & A_3 & O \\ O & O & 0 & O & X_1 & x_1 & x_2 & X_2 & X_3 & X_4 \\ 0^T & 0^T & \bar{W}_O & \hat{W}_O & 0^T & W_O & 0 & a_2^T & a_3^T & 0^T \\ 0^T & 0^T & 0^T & 0^T & y_1^T & 0 & \gamma & y_2^T & y_3^T & y_4^T \\ O & O & 0 & O & Z_1 & z_1 & z_2 & Z_2 & Z_3 & Z_4 \end{pmatrix}.$$

We set $[A_C]_{11} = \mathbf{a}$ and $[A_C]_{22} = \mathbf{b}$. The minor enclosed in boxes is a set of $w-1$ (w is the order of W) auxiliary column vectors for N that we choose such that

$$X \begin{pmatrix} O & 0 \\ A_1 & a_1' \\ a_1^T & \alpha \end{pmatrix} = \begin{pmatrix} * & * \\ * & * \\ \bar{W}_I & \hat{W}_I \end{pmatrix},$$

where X is the matrix that factorizes N. Observe that only the i_hth row of $A_{C'}$ has been modified by replacing $\mathbf{a}$, $[\bar{N}_I]_{1*}$, and $[\hat{N}_I]_{1*}$ with W_O, a_2^T, and a_3^T, respectively.

In what follows we regard A_C as a block 8×10 matrix, and when we refer to the ith row (or column) we really mean the ith block of rows (columns). Nonetheless, A_C is square, if $A_{C'}$ is, with order $\nu + w - 2$ plus the size of $A_{C'}$. Using property (a) of the induction hypothesis we then see that $n_C = O(n)$. It is easy to prove that A_C enjoys properties (b) through (d) as well. Assume C has k' actual inputs. Since g_1 has fanout 1, C' has $k'-1$ actual inputs and, by induction, $A_{C'}$ has $k'-1$ input rows. Now, by the above construction, A_C has exactly $(k'-1)-1+2 = k'$ input rows, which proves (c). Property (d) also easily holds. To prove (b) we use the properties of $\mathcal{F}$. By **p1**, the application of $\nu - 1$ stages of $\mathcal{A}$ to A_C affects only the first three (blocks of) rows. Thus N (including its auxiliary vectors) is properly embedded in A_C, and hence after the first $\nu - 1$ stages of $\mathcal{A}$, we get

$$A_C^{(\nu-1)} = \begin{pmatrix} W_I & \bar{W}_I & 0^T & \hat{W}_I & 0 & \boxed{0^T} & \boxed{0^T} & 0^T \\ \bar{W}_M & W_M & O & \hat{W}_M & 0 & \boxed{A_2} & \boxed{A_3} & O \\ 0 & O & X_1 & x_1 & x_2 & X_2 & X_3 & X_4 \\ \bar{W}_O & \hat{W}_O & 0^T & W_O & 0 & \boxed{a_2^T} & \boxed{a_3^T} & 0^T \\ 0^T & 0^T & y_1^T & 0 & \gamma & y_2^T & y_3^T & y_4^T \\ 0 & O & Z_1 & z_1 & z_2 & Z_2 & Z_3 & Z_4 \end{pmatrix},$$

where $W_I = \mathbf{c}$ is the encoding of $\mathtt{NAND}(z_1, z_2)$. The submatrix enclosed in boxes is a

set of $\nu - 2$ auxiliary vectors for W that we choose such that

$$X \begin{pmatrix} 0^T & 0 \\ A_2 & A_3 \\ a_2^T & a_3^T \end{pmatrix} = \begin{pmatrix} * & * \\ * & * \\ [\bar{N}_I]_{1*} & [\hat{N}_I]_{1*} \end{pmatrix},$$

where X is the transformation matrix that triangularizes W. Note that the entries corresponding to the first elements of auxiliary vectors contain zeros, as required. If the first element (in the definition) of auxiliary vectors were not zero, we would be faced with the additional problem of guaranteeing that the first $\nu - 1$ stages would set these entries to the required values.

It is easy to see that W (including its auxiliary vectors) is properly embedded in $A_C^{(\nu-1)}$ so that additional $w - 1$ stages of $\mathcal{A}$ lead to

$$A_C^{(n+w-2)} = \begin{pmatrix} X_1 & x_1 & x_2 & X_2 & X_3 & X_4 \\ 0^T & N_I & 0 & [\bar{N}_I]_{1*} & [\hat{N}_I]_{1*} & 0^T \\ y_1^T & 0 & \gamma & y_2^T & y_3^T & y_4^T \\ Z_1 & z_1 & z_2 & Z_2 & Z_3 & Z_4 \end{pmatrix},$$

with $N_I = W_I = \mathbf{c}$. The correctness now follows from the induction hypothesis and property **p4**.

2. The full description of the fanout 2 case is definitely more tedious but does not introduce additional difficulties. A_C extends $A_{C'}$ by means of an initial N block, followed by a D block, followed by two W blocks. Taking the partial overlapping into account, it immediately follows that the order of A_C is $\nu + d - 1 + 2(w - 2)$ plus the order of $A_{C'}$.

The construction of matrix A_C can be done in space proportional to $\log n$ by simply reversing the steps of the above inductive process. That is, instead of constructing $A_{C'}$ and using it to build A_C, which would require more than logarithmic work space, we compute and immediately output the first $\nu + w - 2$ (or $\nu + d - 1 + 2(w - 2)$ in case of a first gate with fanout 2) rows and columns. We also compute and output row i_h (or the two rows where the output of a fanout 2 gate has to be sent). All of this essentially can be done in space $O(\log n)$ by copying the elementary matrices N, D, and W to the output medium. The only possible problem might be the computation of i_h, but this is not the case. In fact, for any $1 \leq i \leq n$, let $f_2(i)$ be the number of fanout 2 nand gates preceding gate i in the linear ordering of C. This information can be obtained, when required, by repeatedly reading the input and using only $O(\log n)$ work space for counting. It easily follows from the above results that the index of the top left entry $\pi(h)$ of the hth N block is

$$\begin{aligned} \pi(h) &= f_2(h)(2(w-2) + \nu + d - 1) + (h - 1 - f_2(h))(\nu + w - 2) + 1 \\ &= (h-1)(\nu + w - 2) + f_2(h)(d + w - 3) + 1. \end{aligned}$$

Hence i_h will be either $\pi(h)$ or $\pi(h) + 1$, depending on whether the value under consideration is the first or second input to g_h.

The MATLAB program that implements the transducer [16] is indeed logspace bounded. It uses only the definition of the blocks and simple variables (whose contents never exceed the size of A_C) in magnitude. No data structure depending on n is required. Clearly, as it is implemented using double precision IEEE 754 arithmetic, it can properly handle only the circuits with up to approximately 2^{53} gates. ∎

$$N = \begin{pmatrix}
\mathbf{a} & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{5}{4} & 0 & 0 \\
0 & \mathbf{b} & -1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 & -\frac{579}{145} & -\frac{211}{145} & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 & -\frac{23}{116} & -\frac{70}{29} & 0 & 0 & 0 \\
0 & 0 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{4}{3} & 1 & 0 & 0 & 0 & 0 & \frac{2575}{3552} & 0 \\
0 & 0 & \frac{4}{3} & \frac{33}{8} & 0 & 0 & 0 & 0 & -\frac{38525}{10656} & 0 \\
\frac{4}{3} & 0 & 0 & 0 & 0 & 0 & \frac{5}{4} & 0 & \frac{125}{72} & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{4}{3} & 0 & 1 & \frac{25}{24} & \frac{25}{12}x
\end{pmatrix}.$$

FIG. 4. *The N block for HQR.*

4. Householder's *QR* decomposition algorithm. In this section we prove that HQR is presumably inherently sequential under both exact and floating point arithmetic. This is done by proving that a certain set H, defined in terms of HQR's behavior, is logspace complete for P.

$H = \{A\colon A \text{ is } n\times n,\ A = QR \text{ is the factorization computed by HQR, and } [R]_{nn} > 0\}$.

Note that by HQR we mean the classical Householder's algorithm presented in many numerical analysis textbooks. In particular we refer to the one in [10]. This is also the algorithm available as a primitive routine in scientific libraries (such as LINPACK's `ZQRDC` [7]) and environments (like MATLAB's `qr` [17]).

We begin with the simple result about the membership in P.

THEOREM 4.1. *H is in P under both exact and floating point arithmetic.*

Proof. The proof follows from standard implementations, which perform $O(n^3)$ arithmetic operations and $O(n)$ square root computations (see, e.g., [10]). □

According to the result of section 3, to prove that H is also logspace hard for P it is sufficient to exhibit an encoding scheme and the elementary matrices required in the proof of Theorem 3.1. As we will see, however, the floating point case requires additional care to rule out the possibility of fatal roundoff error propagations.

THEOREM 4.2. *H is logspace hard for P under the real number model of arithmetic.*

Proof. We simply list the three elementary matrices required by Theorem 3.1. For each elementary matrix E, the corresponding auxiliary vector a_E is shown as an additional column of E. That the matrices enjoy the properties defined in section 3.2 can be automatically checked using any symbolic package, such as Mathematica.

N is the 9×10 matrix of Figure 4, where $\mathbf{a}, \mathbf{b} \in \{-1, 1\}$ are the encoding of logical values (1 for `True` and -1 for `False`) and x is an arbitrary real number. Performing eight steps of HQR on input N gives $N = QR$ with $[R]_{9,9} = \mathbf{c}$ and $[R]_{9,10} = x$, where $\mathbf{c} = \frac{1-\mathbf{a}-\mathbf{b}-\mathbf{ab}}{2}$ is the arithmetization of $\mathtt{NAND}(a, b)$ under the selected encoding.

D is the 6×7 matrix shown in Figure 5 (left). Performing four steps of HQR on input D gives $D = QR$ with $[R]_{5,5} = [R]_{6,6} = \mathbf{a}$, $[R]_{5,6} = [R]_{6,5} = 0$, $[R]_{5,7} = z$, and $R_{6,7} = x$, where z and x are arbitrary real numbers.

W is the 2×3 matrix of Figure 5 (right). Performing one step of HQR on input W gives $W = QR$ with $[R]_{2,2} = \mathbf{a}$ and $[R]_{2,3} = x$, for x an arbitrary real number. □

Applying a floating point implementation of HQR to any single block defined in Theorem 4.2[2] results in approximate results. For instance, we performed the QR

[2]More precisely, to the best possible approximations of the blocks under the particular machine arithmetic.

$$D = \begin{pmatrix} \mathbf{a} & -1 & -2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & -\frac{579}{145} & -\frac{211}{145} & 0 \\ 2 & 0 & 0 & 0 & -\frac{23}{116} & -\frac{70}{29} & 0 \\ 0 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{4}{3} & 1 & 0 & 0 & 0 & \frac{27\,x+254\,z}{222} \\ 0 & \frac{4}{3} & \frac{33}{8} & 0 & 0 & 0 & \frac{4767\,x+256\,z}{1776} \end{pmatrix} \qquad W = \begin{pmatrix} \mathbf{a} & -\frac{5}{4} & 0 \\ \frac{4}{3} & 0 & \frac{5\,x}{3} \end{pmatrix}$$

FIG. 5. *The D and W blocks for HQR.*

decomposition of the four N matrices using the built-in function `qr` available in MATLAB. We found that the relative error affecting the computed encoding of $\texttt{NAND}(a, b)$ ranged from a minimum of 0.5ϵ to a maximum of 3ϵ. Here ϵ is the roundoff unit and equals $2.2204 \cdot 10^{-16}$ under IEEE 754 standard arithmetic. These might appear to be insignificant errors. However, for a matrix containing an arbitrary number of blocks, the roundoff error may accumulate to a point where it is impossible to recover the exact (i.e., 1 or -1) result. Clearly, direct error analysis is not feasible here, since it should apply to an infinite number of reduction matrices. Our solution is to control the error growth by "correcting" the intermediate results as soon as they are "computed" by nand blocks. Note that, by referring to the values *computed* by a certain elementary matrix E, we mean precisely the nonzero values one finds in the last row of the triangular factor computed by HQR on input E (including the auxiliary vectors). Analogously, the input values to E are the ones computed by the elementary matrix preceding E in A_C.

THEOREM 4.3. *H is logspace hard for P under finite precision floating point arithmetic.*

Proof. We take duplicator and wire blocks as in Theorem 4.2 and provide a new definition for nand blocks so that they always compute exact results. To do this, we have to consider again the structure of A_C, as resulting from Theorem 3.1.

Let g_i be the ith gate in the topological ordering of C, and let g_j and g_k be the gates providing the inputs to g_i. Let $N^{(l)}$ denote the N block of A_C corresponding to g_l, according to the construction of Theorem 3.1. To prove the result we maintain the invariant that the values computed by $N^{(1)}, N^{(2)}, \ldots, N^{(i-1)}$ are exact. This is clearly true for $i = 1$. Using the invariant we first verify that the errors affecting the values computed by $N^{(i)}$ can be bounded by a small multiple of the roundoff unit. We then use the bound to show how to redefine N so that it computes exact results, thus extending the invariant to $N^{(i)}$.

From the proof of Theorem 3.1 we know that the output of $N^{(j)}$ (and similarly of $N^{(k)}$) is placed in one of the input rows of $N^{(i)}$ as a consequence of the factorization of possibly a D followed by a W block. It follows that the error affecting the output of $N^{(i)}$ is due only to the above factorizations *and* to the factorization of $N^{(i)}$ itself. Since there is a limited number of structural cases (depending on the fanout of gates j and k) and considering all the possible combinations of logical values involved, the largest error ever affecting the output of $N^{(i)}$ can be determined by direct (but tedious) error analysis or, more simply, by test runs. For the purpose of the following discussion we may safely assume that the relative errors affecting the computed quantities are bounded by $c\epsilon$, for some constant c of order unit (c is actually smaller than 10). In other words, we may assume that the actual outputs of $N^{(i)}$ are $\mathbf{a}(1+\delta)$ and $x(1+\eta)$, with $|\delta|, |\eta| \leq c\epsilon$. Recall that x is the last entry of the generic auxiliary vector $a_N(x)$

of N after the factorization (see the definition of N in section 3.2). Here, however, we require that x be a machine number (i.e., a rational number representable without error under the arithmetic under consideration).

Having a bound on the error, we are now ready to show how to "correct" the erroneous outputs. The new nand block, denoted by N_{corr}, extends N with two additional rows and columns, as shown below:

$$N_{\text{corr}} = \begin{pmatrix} N & a_N(-1) & 0 \\ b^T & 0 & -(2^m+1) \\ 0^T & 2^m & 0 \end{pmatrix},$$

where

$$b^T = (\overbrace{0,\ldots,0}^{8}, 2^m)$$

and m is some positive integer (to be specified below). Note that $a_N(-1)$ is precisely that auxiliary vector for the old N that produces -1 as output, i.e.,

$$a_N(-1) = (\overbrace{0,\ldots,0}^{8}, -\frac{25}{12})^T.$$

The auxiliary vector for N_{corr} is

$$(\overbrace{0,\ldots,0}^{10}, (2^m+1)x).$$

Thus, a first requirement on m is that the quantity $(2^m+1)x$ be a computer number. As the length of the significand of 2^m+1 is $m+1$, we see that a sufficient condition is that the length of the significand of x does not exceed $t-m-1$ (for the definition of t see Appendix B). Now, let us apply HQR to N_{corr} extended by its auxiliary vector. As N is properly embedded in N_{corr}, after eight stages of HQR we get (using the above result on the error)

$$N_{\text{corr}}^{(9)} = \begin{pmatrix} \mathbf{a}(1+\delta) & -1+\eta & 0 & 0 \\ 2^m & 0 & -(2^m+1) & 0 \\ 0 & 2^m & 0 & (2^m+1)x \end{pmatrix}.$$

A second condition that we place on m is that $2^m + \mathbf{a}(1+\delta) = 2^m + \mathbf{a}$, to get rid of the error δ. An easy argument shows that this implies $m > \lceil \log c \rceil$. Thus, recalling the bound on $|\delta|$ and $|\eta|$, we see that $m \geq 5$ is sufficient. As a consequence, the length of x cannot exceed $t-6$. The actual reflection matrix applied to $N_{\text{corr}}^{(9)}$ is then

$$I - \frac{1}{2^m(2^m+1)} \begin{pmatrix} \mathbf{a}(2^m+1) \\ 2^m \end{pmatrix} \begin{pmatrix} \mathbf{a}(2^m+1) & 2^m \end{pmatrix},$$

which, by easy floating point computation, gives

$$N_{\text{corr}}^{(10)} = \begin{pmatrix} \mathbf{a}(1-\delta) & -1 & 0 \\ 2^m & 0 & (2^m+1)x \end{pmatrix}.$$

Applying one more stage now leads to the correct results $\mathbf{a}$ and x. The above requirement on x is by no means a problem. In fact, the only auxiliary values ever required are the nonzero elements in the input rows of the blocks that possibly follow nand elementary matrices, i.e., D and W blocks. These are simply -1, -2, and $-5/4$, all of which can be represented exactly with a three-bit significand. ∎

The elementary matrices of Theorem 4.3 are available for the general transducer implemented in MATLAB. In particular, N_{corr} is defined with $m = 30$.

5. *QR* decomposition through Givens' rotations. In this section we prove that the set

$$G = \{A\colon A \text{ is } n \times n,\ A = QR \text{ is the factorization computed by GQR, and } [R]_{nn} > 0\}$$

is logspace complete for P. The way we present the results of this section closely follows the methodology of section 4. Here, however, we need to further discuss the particular algorithm considered. In fact, the computation of the QR decomposition can be done in various ways using plane (or Givens') rotations. Different from Householder's reflections, a single plane rotation annihilates only one element of the matrix to which it is applied, and different sequences of annihilation result in different algorithms. By the way, this degree of freedom has been exploited to obtain the currently faster (among the known accurate ones) parallel linear system solvers [23, 18]. We also note that there is no GQR algorithm available in MATLAB (nor in libraries such as LINPACK or LAPACK), which instead provides the primitive `planerot` to compute a plane rotation. The hardness results of this section apply to the particular algorithm that annihilates the subdiagonal elements of the input matrix by proceeding downward and rightward. This choice places GQR in the class $\mathcal{F}$ defined in section 2, with the position that one stage of the algorithm is the sequence of plane rotations that introduce zeros in one column.

THEOREM 5.1. *G is in P under both exact and floating point arithmetic.*

Proof. For the proof, see, e.g., [10]. We point out only that the membership in P holds independently of the annihilation order. ∎

THEOREM 5.2. *G is logspace hard for P under real number arithmetic.*

Proof. As in Theorem 4.2, we simply list the three elementary matrices extended with the generic auxiliary vector. The matrices are shown in Figures 6 through 8, where $\mathbf{a}, \mathbf{b} \in \{-1, 1\}$ are encodings of logical values (1 for `True` and -1 for `False`) and x and z are arbitrary real numbers. Again, that the matrices enjoy the properties defined in section 3.2 can be verified with the help of a symbolic package. ∎

We now switch to the more delicate case of finite precision arithmetic.

THEOREM 5.3. *G is logspace hard for P under finite precision floating point arithmetic.*

Proof. We apply the ideas of Theorem 4.3. That is, we extend the definition of N so that it always computes the exact results. Here, however, there is a subtle problem whose solution requires a different definition of the duplicator block. Let us look at the details. If we apply a floating point implementation of GQR to the block of Figure 7, we clearly get approximate results. In particular, instead of $\left(\begin{smallmatrix} \mathbf{a} & 0 \\ 0 & \mathbf{a} \end{smallmatrix}\right)$, in the bottom right corner of R we get $\left(\begin{smallmatrix} \mathbf{a}(1+\delta') & \epsilon' \\ \epsilon'' & \mathbf{a}(1+\delta'') \end{smallmatrix}\right)$. Even if δ', δ'', ϵ', and ϵ'' are of the order of the roundoff unit ϵ, the fact that ϵ'' is not zero causes the whole construction to fail. Note that the same kind of approximate results are obtained under HQR, but with no damage there. For suppose that ϵ'' is in column k of A_C and consider stage k of both algorithms. In HQR one single transformation annihilates the whole column so that the contribution of a tiny ϵ'' to the kth transformation matrix is negligible. On the other hand, in GQR the elements are annihilated selectively and, if ϵ'' is not a true zero, one additional plane rotation is triggered between rows k and $k+1$ which places zero in the entry $(k+1, k)$. Unfortunately, this rotation also has the effect of placing a positive value in position (k, k), thus (possibly) altering the sign that encoded the logical value being passed around and causing the whole simulation to fail in general.

We thus need to replace the duplicator with one that returns a true zero in the entry $(d, d-1)$ of the incomplete factor R. To do this we must exploit the proper-

$$N = \begin{pmatrix}
\mathbf{a} & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & \mathbf{b} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 3 & 0 & 0 & 0 & \frac{3-3\sqrt{65}}{16} & \frac{-3+\sqrt{65}}{4} & -\frac{183+25\sqrt{65}}{64} & -\frac{723+125\sqrt{65}}{384} & 0 \\
0 & 1 & 0 & 2 & 0 & \frac{39\sqrt{\frac{13}{7}}}{2} + 4\sqrt{30} & \frac{-19\zeta}{14} & 0 & 0 & \alpha & 0 \\
0 & 1 & 0 & 3 & 0 & 6\sqrt{\frac{13}{7}} + \sqrt{30} & \frac{-4\zeta}{7} & 0 & 0 & \beta & 0 \\
0 & 1 & 0 & 4 & 0 & \frac{29\sqrt{\frac{13}{7}}}{2} + 3\sqrt{30} & \frac{-17\zeta}{14} & 0 & 0 & \gamma & 0 \\
0 & 1 & 0 & 5 & 0 & \frac{7\sqrt{30}+5\sqrt{91}}{2} & \frac{\sqrt{10}-3\zeta}{2} & 0 & 0 & \delta & 0 \\
2 & 0 & 4 & 0 & 0 & 0 & \frac{5-3\sqrt{65}}{8} & \frac{-1+\sqrt{65}}{2} & -\frac{97+25\sqrt{65}}{32} & -\frac{357+125\sqrt{65}}{192} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \frac{-5x}{3} \\
0 & 0 & 0 & 0 & 0 & \frac{4}{3} & 0 & -1 & 0 & -\frac{125}{96} & 0
\end{pmatrix}.$$

$$\alpha = \frac{-16625\sqrt{\frac{5}{2}}}{504} + \frac{875\sqrt{\frac{15}{2}}}{112} - \frac{4875\sqrt{91}}{1792} + \frac{2375\sqrt{371}}{1008}, \quad \beta = \frac{125}{64}\left(3\sqrt{\frac{53}{7}} - 3\sqrt{\frac{13}{7}} - 6\sqrt{\frac{5}{2}} + \sqrt{\frac{15}{2}}\right),$$

$$\gamma = \frac{-14875\sqrt{\frac{5}{2}}}{504} + \frac{2625\sqrt{\frac{15}{2}}}{448} - \frac{3625\sqrt{91}}{1792} + \frac{2125\sqrt{371}}{1008}, \quad \delta = 125\left(\frac{\sqrt{371}}{48} - \frac{5\sqrt{\frac{5}{2}}}{18} + \frac{7\sqrt{\frac{15}{2}}}{128} - \frac{5\sqrt{91}}{256}\right)$$

$\zeta = 7\sqrt{10} + \sqrt{371}$.

FIG. 6. *The N block for GQR.*

$$D = \begin{pmatrix}
\mathbf{a} & 1 & 0 & 0 & 0 & 0 \\
1 & 2 & 0 & \frac{39\sqrt{\frac{13}{7}}}{2} + 4\sqrt{30} & \frac{-19\zeta}{14} & \left(\frac{-39\sqrt{\frac{13}{7}}}{2} + 4\sqrt{30}\right)x + \left(-19\sqrt{\frac{5}{2}} + \frac{19\sqrt{\frac{53}{7}}}{2}\right)z \\
1 & 3 & 0 & 6\sqrt{\frac{13}{7}} + \sqrt{30} & \frac{-4\zeta}{7} & \left(-6\sqrt{\frac{13}{7}} + \sqrt{30}\right)x + \left(4\sqrt{\frac{53}{7}} - 4\sqrt{10}\right)z \\
1 & 4 & 0 & \frac{29\sqrt{\frac{13}{7}}}{2} + 3\sqrt{30} & \frac{-17\zeta}{14} & \left(\frac{-29\sqrt{\frac{13}{7}}}{2} + 3\sqrt{30}\right)x + \left(-17\sqrt{\frac{5}{2}} + \frac{17\sqrt{\frac{53}{7}}}{2}\right)z \\
1 & 5 & 0 & \frac{7\sqrt{30}+5\sqrt{91}}{2} & \frac{\sqrt{10}-3\zeta}{2} & \left(7\sqrt{\frac{15}{2}} - \frac{5\sqrt{91}}{2}\right)x + \left(-10\sqrt{10} + \frac{3\sqrt{371}}{2}\right)z
\end{pmatrix}$$

$\zeta = 7\sqrt{10} + \sqrt{371}$.

FIG. 7. *The D block for GQR.*

ties of floating point arithmetic. Let m and M denote the length of the significand and the largest exponent e such that 2^e can be represented in the arithmetic under consideration, respectively. For the standard IEEE 754, $m = 53$ and $M = 1023$. The nonzero entries of the floating point D block, shown in Figure 9, are powers of 2. In this way any operation will be either exact or simply a no operation.

As the new auxiliary vector, we define

$$a_D(x, y) = \left(0, y2^{M-2}, 1, 1, x2^m, 2^{-M}, 2^m, 2^m, 2^{m-\lceil \frac{m}{2} \rceil}, 2^{m-\lceil \frac{m}{2} \rceil}\right)^T.$$

Note that the only assignments to x and y we need are 0 and 1 or 1 and 0.

The rest of the proof is now similar to that of Theorem 4.3. We show how to correct the slightly erroneous values computed by an N block, assuming that the previous N blocks return exact results. Let N stand for the nand block adopted for

$$W = \begin{pmatrix} \mathbf{a} & 1 & 1 & 0 \\ 2 & 3 & \frac{-3+\sqrt{65}}{4} & -\frac{15\,x}{4} - \frac{\sqrt{65}\,x}{4} \\ 2 & 4 & \frac{-1+\sqrt{65}}{2} & -\frac{9\,x}{2} - \frac{\sqrt{65}\,x}{2} \end{pmatrix}.$$

FIG. 8. *The W block for GQR.*

$$D = \begin{pmatrix} \mathbf{a} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2^{-m} & 2^{-M+2m+1} & -2^{M-1} & 0 & 0 & 0 & 0 & 0 \\ 2^{-m} & 2^{-3m} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2^{-M+3m+2} & 1 & 2^{M-1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2^{-M+1+m} & 0 & 2^{-M+2m+1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2^{-m} & 2^{M-1-m} & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2^{-\lfloor \frac{m}{2} \rfloor} & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2^{-\lfloor \frac{m}{2} \rfloor} & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2^{-\lceil \frac{m}{2} \rceil} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2^{-\lceil \frac{m}{2} \rceil} & 0 & 1 \end{pmatrix}.$$

FIG. 9. *The D block for floating point GQR.*

the exact arithmetic version of GQR (Figure 6). The new nand block is then

$$N_{\text{corr}} = \begin{pmatrix} N & 0 & 0 \\ c^T & 1 & 0 \\ 0^T & 2^{-\lceil \frac{m}{2} \rceil} & 1 \end{pmatrix},$$

where

$$c^T = (\overbrace{0, \ldots, 0}^{9}, 2^{-\lfloor \frac{m}{2} \rfloor}).$$

As the new auxiliary vector, we take

$$a_{N_{\text{corr}}}(x) = \left(\overbrace{0, \ldots, 0}^{8}, -\frac{5}{3} x 2^m, 0, 2^m, 2^{m-\lceil \frac{m}{2} \rceil} \right)^T,$$

i.e., the first 10 entries of $a_{N_{\text{corr}}}(x)$ coincide with $a_N(x2^m)$. Now, let us apply GQR to N_{corr} extended by its auxiliary vector. As N is properly embedded in N_{corr}, after nine stages of GQR we get

$$N_{\text{corr}}^{(10)} = \begin{pmatrix} \mathbf{a}(1+\delta) & 0 & 0 & x2^m(1+\eta) \\ 2^{-\lfloor m/2 \rfloor} & 1 & 0 & 2^m \\ 0 & 2^{-\lceil m/2 \rceil} & 1 & 2^{m-\lceil m/2 \rceil} \end{pmatrix},$$

where $|\delta|, |\eta| \leq c\epsilon$, for some small constant c of order unit. The plane rotation to annihilate the entry $(2,1)$ of $N_{\text{corr}}^{(10)}$ is represented by

$$G_1 = \frac{1}{\sqrt{\mathbf{a}^2(1+\delta)^2 \oplus 2^{-2\lfloor m/2 \rfloor}}} \otimes \begin{pmatrix} \mathbf{a}(1+\delta) & 2^{-\lfloor m/2 \rfloor} \\ -2^{-\lfloor m/2 \rfloor} & \mathbf{a}(1+\delta) \end{pmatrix} = \begin{pmatrix} \mathbf{a} & \frac{2^{-\lfloor m/2 \rfloor}}{1+\delta} \\ -\frac{2^{-\lfloor m/2 \rfloor}}{1+\delta} & \mathbf{a} \end{pmatrix},$$

and its application in floating point gives

$$N_{\text{corr}}^{(10)} = \begin{pmatrix} \mathbf{a} & 0 & \mathbf{a}2^m \ominus x2^{\lceil m/2 \rceil}(1+\zeta) \\ 2^{-\lceil m/2 \rceil} & 1 & 2^{m-\lceil m/2 \rceil} \end{pmatrix},$$

where $1+\zeta = \frac{1+\eta}{1+\delta}$ and $|\zeta| \leq 2\epsilon + O(\epsilon^2)$. The crucial point is that, if x can be represented with no more than $2^{\lfloor m/2 \rfloor}$ significant bits, the alignment of the fractional part performed during the execution of $\mathbf{a}2^m \ominus x2^{\lceil m/2 \rceil}(1+\zeta)$ will simply cause the contribution $x2^{\lceil m/2 \rceil}\zeta$ to be lost. Hence, the computed element in the entry $(1,3)$ will be $\mathbf{a}2^m - x2^{\lceil m/2 \rceil}$. However, one more rotation produces the exact values $\mathbf{a}$ and x in the last row. Note that the only value required in place of x is 1. □

6. GE with pivoting. In this section we consider the algorithm of GE with partial pivoting, or simply GEP, a technique that avoids nondegeneracies and ensures (almost always) numerical accuracy. We also consider the lesser-known minimal pivoting technique, GEM, one that guarantees only that a PLU factorization is found. Minimal pivoting has been adopted for systolic-like implementations of GE [12]. A brief description of these algorithms is reported in Appendix A.

We prove that GEM is P-complete, unless applied to strongly nonsingular matrices, while GEP is P-complete even when restricted to strongly nonsingular matrices. Finally, we prove that GEP is P-complete over GF(2).

6.1. Partial pivoting. The proof we give here builds on the original proof in [24] and hence does not share the common structure of the other reductions in this paper. Essentially we show that, with little additional effort with respect to Vavasis' proof, we can exhibit a reduction in which the matrix obtained is strongly nonsingular. As already pointed out, strongly nonsingular matrices are of remarkable importance in practical applications. This class contains symmetric positive definite (SPD) and diagonally dominant matrices, which often arise from the discretization of differential problems. Observe that, with any such matrix in input, plain GE (no pivoting) is nondegenerate, but it is not stable in general and hence is not the algorithm of choice.

As in [24], what we actually prove to be P-complete is the following set:

$$L = \{(i,j,A) : \ A \text{ is strongly nonsingular and, on input } A, \text{ GEP uses row } i \text{ to eliminate column } j \ \}.$$

THEOREM 6.1. *The set L is logspace complete for P over the real or rational numbers.*

We postpone the technical proof of Theorem 6.1 to Appendix C but give an example that shows the way the matrix given in [24] is modified. Figure 10 depicts the reduction matrix $\mathbf{M}_C$ corresponding to the circuit of Figure 3, obtained according to the rules in [24]. The matrix is nonsingular; however, it can be seen that the leading principal minor of order 2 is singular. The matrix we obtain, according to Theorem 6.1, is shown in Figure 11. It can be easily seen that our matrix is strongly diagonally dominant by rows and hence strongly nonsingular.

6.2. Minimal pivoting. The technique of minimal pivoting, i.e., selecting as the pivot row at stage k the first one with a nonzero entry (below or on the main diagonal) in column k, is probably the simplest modification that allows GE to cope with degenerate cases. However, such a simple technique is sufficient to make GE P-complete. Note that, even if no formal error analysis is available for GEM, it is not difficult to exhibit matrices (that can plausibly appear in real applications) such that the error incurred by GEM is very large. Actually, GEM is likely to be as unstable as standard GE (no pivoting).

Consider the following set:

$$L' = \{A : \ A \text{ is } n \times n,\ PA = LU \text{ is the factorization computed by GEM, and } [U]_{nn} > 0\}.$$

$$\begin{pmatrix}
-3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\mathbf{a} & \cdot & 1 & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \mathbf{b} & 1 & \cdot & 1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & 1 & 4 & 1 & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & 1 & 4 & \cdot & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & 1 & \cdot & 4 & 1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1 & 1 & 4 & \cdot & \cdot & \cdot & \cdot & \cdot & 1
\end{pmatrix}$$

FIG. 10. *Matrix* $\mathbf{M}_C$ *corresponding to the* exclusive or *circuit. The symbol* $\cdot$ *stands for a zero entry.*

$$\begin{pmatrix}
-3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\mathbf{a} & 10 & \cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \mathbf{b} & 10 & 1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & 1 & \cdot & 4 & 10 & 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & \cdot & \cdot & 1 & \cdot & 4 & 10 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & 1 & \cdot & \cdot & \cdot & 4 & 10 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -3.9 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & \cdot & 4 & 10 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 20 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 30 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 20 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 30 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 20 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 30 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 20 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 30 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 20 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 30 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 20 & \cdot & \cdot & \cdot & \cdot & \cdot & 30
\end{pmatrix}$$

FIG. 11. *The matrix* $\mathbf{A}_C$ *for the computation of* $\texttt{XOR}(a, b)$.

THEOREM 6.2. *The set* L' *is logspace complete for* P *under both real and finite precision floating point arithmetic.*

Proof. The set is clearly in P, as GEM runs in time $O(n^3)$ under both models of arithmetic. We first show that L' is also logspace hard for P when the input matrices are singular and then show how to restrict the input set. As GEM belongs to the class $\mathcal{F}$, to prove the hardness of L' we simply list the three elementary matrices required by Theorem 3.1. Note that the matrices are the same for both models of arithmetic, as the operations performed by GEM in floating point are exact. The encoding of logical values here is 0 for `False` and 1 for `True`. The matrices are depicted in Figures 12 and 13.

A_C is clearly singular. Now consider the following matrix B_C of order $2n_C$, where n_C is the order of A_C:

$$B_C = \begin{pmatrix} A_C & E \\ E & O \end{pmatrix}, \tag{4}$$

where E is the matrix with 1 on the antidiagonal and 0 elsewhere. The determinant of B_C can be easily proved to be ± 1. Moreover, if $\bar{P}B_C = \bar{L}\bar{U}$ is the factorization computed by GEM, then $[\bar{U}]_{n_C,n_C} = [U]_{n_C,n_C}$, where U is the upper triangular factor computed by GEM on input A_C. Note, then, that what we prove to be P-complete is not exactly L' but a set with a slightly more complicated definition.

As usual, in the following the notation $A_C^{(k)}$ will be used to denote the matrix

$$N = \begin{pmatrix} \mathbf{a} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{b} & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 & 0 & x \end{pmatrix} \qquad W = \begin{pmatrix} \mathbf{a} & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & x \end{pmatrix}$$

FIG. 12. *The nand (left) and wire (right) blocks for GEM.*

$$D = \begin{pmatrix} \mathbf{a} & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & z \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & z \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

FIG. 13. *The D block for GEM.*

obtained after $k-1$ stages of GEM on input A_C and considering only the entries (i,j) with $i,j \geq k$. However, by writing $B_C^{(k)}$ we mean the submatrix obtained after $k-1$ stages of GEM (on input B_C) and considering only the entries (i,j) such that $k \leq i,j \leq n_C$. With this position, to prove that the output of C can be read off entry (n_C, n_C) of the U factor of B_C, we show that the executions of GEM on input A_C and on input B_C result in identical submatrices $A_C^{(k)}$ and $B_C^{(k)}$, for $0 \leq k \leq n_C$. The proof is by induction. Initially, for $k = 1$, the equality follows from the definition of B_C. Consider stage $k \geq 1$. If column k of $A_C^{(k)}$ contains a nonzero element below or on the main diagonal, say, at row index i, then the selected pivot row is the ith under both executions. The result follows then from the induction hypothesis and the fact that exactly the same operations are performed on the elements of the submatrices. If no nonzero element is found in column k of $A_C^{(k)}$, then stage k of the first execution has no effect, and hence $A_C^{(k+1)} = A_C^{(k)}$. Under the second execution (the one on input B_C) by construction the pivot is taken from row $2n_C - k + 1$. However, the pivot is the only nonzero element in row $2n_C - k + 1$ and thus the effect of this step is simply the exchange of rows k and $2n_C - k + 1$. However, once more $B_C^{(k+1)} = B_C^{(k)}$. □

Two concluding remarks are in order. First, we note that the application of GEP to the elementary matrices of Theorem 6.2 would return the same factors determined by GEM. This is because the only nonzero elements appearing during the factorizations are -1 and 1, and hence the first nonzero in a column is also the largest magnitude element in that column. Within our general framework, this observation represents another proof that GEP is presumably inherently sequential. The first such proof was given in [24].

As the second observation, we note that the set L' of Theorem 6.2 would be NC computable if the input set were restricted to the class of strongly nonsingular matrices. In fact, on input these matrices GEM and standard GE behave exactly the same.

6.3. Completeness over GF(2). The status of GEP over finite fields (i.e., whether P-complete or NC-computable) is mentioned as an open problem in [11]. Over finite fields there is clearly no question of stability, so the only problem is to find a nonzero pivot at each stage. Under these circumstances GEP is intended to select, as the pivot for stage l, the first nonzero element in column l below or on the main diagonal and thus coincides with GEM. Consider the set

$$F = \{(A, l, k) : \ A \text{ is } n \times n \text{ with entries over a finite field, and the pivot for the } l\text{th stage of GEP is taken from row } k\}$$

mentioned in [11].

THEOREM 6.3. *The set F is logspace complete for P over GF(2).*

Proof. We first show that deciding whether the element $[U]_{nn}$ (where U is the upper triangular factor computed by GEM on input A) is 0 or 1 is P-complete. This is easy using our general framework. The elementary matrices are exactly the ones given in Theorem 6.2, where -1 has to be interpreted as the additive inverse of 1 in GF(2) (and thus replaced by 1). Next we prove that this decision problem can be easily reduced to membership in F. To see this, it is sufficient to observe the behavior of GEP on input a nand elementary matrix. It is easy to check that the following row permutations take place:

(i) $1 \leftrightarrow 3$ and $2 \leftrightarrow 4$, if the logical inputs are both 0;
(ii) $1 \leftrightarrow 3$ if the inputs are 0 and 1 (in this order);
(iii) $2 \leftrightarrow 3$ if the inputs are 1 and 0;
(iv) no permutation if the inputs are both 1.

It follows that the first nonzero is taken from row 3 in either the first or the second stage provided that some input is 0, and hence if the output is 1. By our construction, no row corresponding to the last nand block in the reduction matrix A_C is used to eliminate previous columns. Hence, to decide whether the element $[U]_{nn}$ is 0 or 1 is equivalent to asking whether $(A_C, n_C - 4, n_C - 2) \in F$ or $(A_C, n_C - 3, n_C - 2) \in F$, where n_C is the size of A_C. ∎

7. On *NC* algorithms for the *PLU* decomposition. In this section we show that a known NC algorithm for computing a PLU decomposition of a nonsingular matrix (see [8]) corresponds to GE with a nonstandard pivoting strategy which is only a minor variation of minimal pivoting. This result seems to be just a "curiosity"; however, we can prove that the same strategy is inherently sequential on input arbitrary matrices, which can be seen as further evidence of the difficulties of finding an NC algorithm to compute the PLU decomposition of possibly singular matrices.

The new strategy will be referred to as minimal pivoting with circular shift and the corresponding elimination algorithm referred to simply as GEMS. The reason for its name is that GEMS, like GEM, searches the current column (say, column k) for the first nonzero element. Once one is found, say in row i, a circular shift of rows k through i is performed to bring row i in place of row k (and the latter in place of row $k + 1$).

THEOREM 7.1. *Computing the PLU factorization returned by GEMS on input a nonsingular matrix is in arithmetic NC^2.*

Proof. We consider the algorithm of Eberly [8]. Given A, nonsingular of order n, let A_i denote the $n \times i$ matrix formed from the first i columns of A, $i = 1, \ldots, n$. If S_i denotes the set of indices of the lexicographically first maximal independent subset of the rows of A_i, then $|S_i| = i$, since A_i has full column rank. Moreover, $S_i \subseteq S_{i+1}$, $i = 1, \ldots, n - 1$. Note that the computation of all the S_i is in NC^2 (see [3]). Now,

let $S_1 = \{j_1\}$, and, for $i = 2, \ldots, n$, $S_{i+1} - S_i = \{j_{i+1}\}$. Then a permutation P such that $P^T A$ has LU factorization is simply

$$P = (e_{j_1} | e_{j_2} | \ldots | e_{j_n}),$$

where e_i is the ith unit (column) vector. Clearly, once P has been determined, computing the LU factorization of $P^T A$ can be done in polylogarithmic parallel time using known algorithms. We now show by induction on the column index k that P is the same permutation determined by GEMS. The basis is trivial, since j_1 is the index of the first nonzero element in column 1 of A. Now, for $k > 1$, let

$$(e_{j_1} | \ldots | e_{j_k} | e_{l_{k+1}} | \ldots | e_{l_n})^T A = L_k U_k = L_k \begin{pmatrix} R_k & B_k \\ O & \mathcal{A}_k \end{pmatrix} \tag{5}$$

be the (partial) factorization computed by GEMS, where R_k is upper triangular with nonzero diagonal elements (since A is nonsingular) and the unit vectors $e_{l_{k+1}}, \ldots, e_{l_n}$ extend $e_{j_1}, \ldots, e_{j_k}$ to form a permutation matrix. Clearly, minimal pivoting ensures that $l_{k+1} < \cdots < l_n$. Now, the next pivot row selected by GEMS is the one corresponding to the first nonzero element in the first column of $\mathcal{A}_k$. Let $k + 1 \le i \le n$ denote the index of the pivot row. Since GE does nothing but linear combinations between rows, it follows that the initial matrix A_{k+1} satisfies

$$\det\left((e_{j_1} | \ldots | e_{j_k} | e_{l_m})^T A_{k+1}\right) = 0,$$

for any $m \in \{k + 1, \ldots, i - 1\}$, and

$$\det\left((e_{j_1} | \ldots | e_{j_k} | e_{l_i})^T A_{k+1}\right) \neq 0.$$

This in turn implies that $S_{i+1} = \{j_1, \ldots, j_k, l_i\}$, i.e., that $l_i = j_{k+1}$. □

We now show that GEMS is inherently sequential by proving that the set

$$L'' = \{A\colon\ A \text{ is } n \times n,\ PA = LU \text{ is the factorization computed by GEMS, and } [U]_{nn} > 0\}$$

is P-complete. Clearly, that L'' is in P is obvious, so what remains to prove is the following.

THEOREM 7.2. *L'' is logspace hard for P.*

Proof. Once more, GEMS is in the class $\mathcal{F}$. So we simply give the elementary matrices. This is very easy. Everything is the same as in the first part of the proof of Theorem 6.2, *except for the auxiliary vector of D.* The new definition for $a_D(x, z)$ is

$$a_D(x, z) = (x - z, z, -x, x - z, 0, 0, 0, 0, 0, 0, 0)^T. \qquad \square$$

It is an easy but interesting exercise to understand why the second part of Theorem 6.2, which extends the P-completeness result to nonsingular matrices, does not work here. (We know that it cannot work, in view of Theorem 7.1.)

8. Conclusions and open problems. The matrices corresponding, for both Householder's and Givens' algorithms, to a circuit C are singular, in general. More precisely, the duplicator elementary matrix is singular, so that all the matrices that do not correspond to simple formulas (fanout 1 circuits) are bound to be singular. All the attempts we made to extend the proofs to nonsingular matrices failed. The reason for this state of affairs could be an interesting subject in itself. To see that

TABLE 1
Parallel complexity of GE with different pivoting strategies and for different classes of input matrices. The results proved in this paper are in boldface.

	General matrices	Nonsingular matrices	Strongly nonsingular matrices
GEP	Inherently seq.	Inherently seq.	**Inherently seq.**
GEM	**Inherently seq.**	**Inherently seq.**	*NC*
GEMS	**Inherently seq.**	***NC***	*NC*

the reason for these failures might be deeper than simply our technical inability, we mention a result of Allender et al. [1] about the "power" of singular matrices. They prove that the set of singular integer matrices is complete for the complexity class $C_=L$.[3] The result extends to the problem of verifying the rank of integer matrices. Of course, our work is at a different level: we are essentially dealing with presumably inherently sequential algorithms for problems that parallelize very well (using different approaches). However, the coincidence suggests that nonsingular matrices might not have enough power to map a general circuit. This is the major open problem for the QR algorithms.

Also, for general matrices, it would be interesting to know the status of Householder's algorithm with column pivoting, which is particularly suitable for the accurate rank determination under floating point arithmetic.

For what concerns Givens' rotations, an obvious open problem is to determine the status of other annihilation orderings, especially the ones that proved to be very effective in limited parallelism environments [23, 18]. We suspect that these lead to inherently sequential algorithms as well.

Finally, Table 1 provides a summary of the known results for the three pivoting strategies investigated in this paper for GE.

As already mentioned, the results of this paper support the belief that there is a tradeoff between parallelism, on the one hand, and nondegeneracy and accuracy, on the other, in numerical algorithms [6]. We suspect that far deeper work is needed to either prove such a tradeoff on a solid theoretical ground or to exhibit stable algorithms substantially more efficient than the ones adopted by numerical analysts for decades.

Appendix A. Algorithms for matrix factorization. In this appendix we describe the factorization algorithms considered in this paper.

Gaussian elimination (GE). GE computes the LU decomposition of A (whenever it exists) by determining a sequence of $n-1$ elementary transformations $M^{(k)}$ with the following properties:

$$\begin{cases} A^{(1)} &= A, \\ A^{(k+1)} &= M^{(k)}A^{(k)}, & k = 1, \ldots, n-1, \\ a_{ij}^{(k)} &= 0, & i > j \text{ and } j < k, \\ U &= A^{(n)}, \\ L &= \prod \left(M^{(k)}\right)^{-1}. \end{cases}$$

In other words, the transformation $A^{(k+1)} = M^{(k)}A^{(k)}$ sends to zero the elements in column k of $A^{(k)}$ below the main diagonal, leaving the already introduced zeros

[3] A set A is in $C_=L$ provided that there is a nondeterministic logspace bounded Turing machine M such that $x \in A$ iff M has the same number of accepting and rejecting computations on input x.

unchanged. The kth transformation $M^{(k)}$ is defined as $I - \tau e_k^T$, where

$$\tau^T = (0, \ldots, 0, \tau_{k+1}, \ldots, \tau_n)$$

and $\tau_i = a_{ik}^{(k)}/a_{kk}^{(k)},\ i = k+1, \ldots, n$. If, for some k, $a_{kk}^{(k)} = 0$, the algorithm fails. However, it can be proved that, if A is strongly nonsingular, $a_{kk}^{(k)} \neq 0,\ k = 1, \ldots, n$.

GE with partial pivoting (GEP). GEP computes a PLU decomposition of A. GEP never fails. As in GE, the matrices L and U are built using a sequence of elementary transformations. However, before applying $M^{(k)}$ to $A^{(k)}$, GEP determines the minimum index h such that

$$|a_{hk}^{(k)}| = \max_{k \leq i \leq n} |a_{ik}^{(k)}|$$

and swaps the rows k and h of $A^{(k)}$. If the maximum above is 0, the algorithm sets $A^{(k+1)} = A^{(k)}$. The rule used for choosing the index h is an example of *pivoting strategy*, and the row h itself is called the *pivot row*.

GE with minimal pivoting (GEM). This is similar to GEP, with the only difference being that h is the minimum among the indices i, $k \leq i \leq n$, such that $a_{ik}^{(k)} \neq 0$. If no such index exists, the algorithm sets $A^{(k+1)} = A^{(k)}$.

QR factorization via Householder's reflections (HQR). HQR applies a sequence of $n-1$ elementary orthogonal transformations $Q^{(k)}$ to A, i.e.,

$$\begin{cases} A^{(0)} & = & A, & \\ A^{(k+1)} & = & Q^{(k)} A^{(k)}, & k = 1, \ldots, n-1, \\ a_{ij}^{(k)} & = & 0, & i > j \text{ and } j < k, \\ R & = & A^{(n-1)}. & \end{cases}$$

Since the $Q^{(k)}$ are orthogonal, we can write

$$A = (Q^{(1)})^T (Q^{(2)})^T \cdots (Q^{(n-1)})^T R = QR,$$

which represents the factorization computed by the algorithm. The matrix $Q^{(k)}$ is defined as follows. Let

$$a = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{kk}^{(k-1)} \\ \vdots \\ a_{nk}^{(k-1)} \end{pmatrix}, \quad \text{and define} \quad v = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{kk}^{(k-1)} \\ \vdots \\ a_{nk}^{(k-1)} \end{pmatrix} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \theta\sqrt{a^T a} \\ \vdots \\ 0 \end{pmatrix}, \tag{6}$$

where $\theta = a_{kk}^{(k-1)}/|a_{kk}^{(k-1)}|$ is the sign of $a_{kk}^{(k-1)}$. Then

$$Q^{(k)} = \begin{cases} I - \frac{2}{v^T v} v v^T & \text{if } v \neq 0, \\ I & \text{otherwise.} \end{cases} \tag{7}$$

The important point to observe is that there are other possible strategies for choosing θ in (6) that would produce the same effect of annihilating the kth column. However, the choice adopted here is be preferred for stability reasons.

QR factorization via Givens' rotations (GQR). GQR applies to general real matrices. It computes a sequence of $\frac{n(n-1)}{2}$ transformations (called *rotations*), such that each transformation annihilates one element below the main diagonal, leaving all the already introduced zeros unchanged. GQR annihilates the subdiagonal part of the matrix in the natural order (left to right and top to bottom).

The rotation used to annihilate a selected entry a_{ji} of a matrix A is the orthogonal matrix G_{ij} defined as follows:

$$G_{ij} = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \begin{matrix} \\ \\ \longleftarrow i \\ \\ \longleftarrow j \\ \\ \\ \end{matrix},$$

where $c = \frac{a_{ii}}{\sqrt{a_{ii}^2+a_{ji}^2}}$ and $s = \frac{-a_{ji}}{\sqrt{a_{ii}^2+a_{ji}^2}}$. One can easily verify that G_{ij} is indeed orthogonal and that the entry j, i of $G_{ij} \cdot A$ is zero.

Appendix B. Floating point number representation. A floating point system $\mathcal{S}$ is characterized, on a particular computer, by four integers: the *base* b of the representation (usually $b = 2$), the *precision* t, and the *range* of the exponent $[\ell, L]$. A number $f \in \mathcal{S}$ has the form

$$f = \pm .b_1 b_2 \cdots b_t \times b^e,$$

where $0 \le b_i < b$ and $\ell \le e \le L$. It is also required that $b_1 \neq 0$, which is a normalization condition. The sequence $.b_1 b_2 \cdots b_t$ is the *significand*, while e is the *exponent*.

If $\mathrm{fl}(x)$ is the (rounded or chopped) floating point representation of a real x, then

$$\mathrm{fl}(x) = x(1+\delta)$$

(if no exceptional condition occurs), where $|\delta| \le \epsilon$ and

$$\epsilon = \begin{cases} \frac{1}{2} b^{1-t} & \text{for rounded arithmetic,} \\ \\ b^{1-t} & \text{for chopped arithmetic.} \end{cases}$$

ϵ is the so-called *machine precision* or *roundoff unit* and is used for roundoff error analysis. In particular, let $\odot$ denote the floating point implementation of the arithmetic operation $\cdot \in \{+, -, \times, /\}$. If $x, y \in \mathcal{S}$, then

$$x \odot y = \mathrm{fl}(x \ \cdot \ y) = (x \ \cdot \ y)(1+\eta), \tag{8}$$

where $|\eta| \le \epsilon$. Equation (8) is known as the *standard model* of arithmetic. A property that plays a crucial role in our reduction is the following. Let a and b be floating point numbers such that $|b| < \epsilon |a|$. Then $a \oplus b = a$.

Appendix C. Proof of Theorem 6.1. The set is clearly in P. Now, given a nand circuit C with n inputs and gates together, we define a strongly nonsingular

matrix $\mathbf{A}_C$ such that, for $k = 1, \ldots, n$, if the output of node k of C (either an input or a nand gate) is `True`, the pivot for step $2k-1$ will be taken from row $2k-1$, or else the pivot will be taken from row $2k$.

For the benefit of the reader, we first briefly recall the P-completeness proof for nonsingular matrices given in [24]. Let $\mathbf{M}_C$ denote the matrix corresponding to the circuit C according to [24] (see Figure 10). If the circuit has n inputs and gates, the matrix $\mathbf{M}_C$ has order $2n$. However, the circuit simulation takes place while performing the elimination of the first n columns, with columns $n+1, \ldots, 2n$ having the only purpose of ensuring nonsingularity. For example, in the matrix of Figure 10 the first two columns correspond to the inputs a, b, while, for $i = 3, \ldots, 6$, column i corresponds to gate $i-2$. For the circuit inputs a logical value `False` is represented by the value 4, while a logical value `True` is represented by the value 3.75. When one executes GEP on $\mathbf{M}_C$, a logical value `False` for a certain input or gate yields a pivot 4 in the corresponding column, whereas a logical value `True` yields a pivot -3.9. Therefore, the sequence of rows selected by GEP provides us with the output of each gate in the circuit C.

The proof in [24] is based on the observation that a nand gate outputs `False` unless one of its inputs is `False`. This fact is mirrored in the matrix $\mathbf{M}_C$, where the pivot of each column is 4 unless that value is modified in a previous elimination step. The structure of the matrix is such that the processing of a pivot 4, corresponding to a `False` output, changes the status of the gates receiving that output from `False` to `True`. This change of status is achieved by subtracting 0.25 from an entry initially set to 4. This ensures that at the due time the pivot in the corresponding column will be -3.9. For example, in the matrix of Figure 10 the selection of the pivot 4 in column 4 (corresponding to a `False` output for gate 2) reduces the 4 entry in column 6 by 0.25, ensuring that the pivot for that column will be -3.9 (corresponding to a `True` output for gate 4). Note that the 1's in the first n columns of $\mathbf{M}_C$ correspond to the wires of the circuit, one pair of 1's for each wire. Below and to the right of each 4 entry there is a number of 1's equal to the fanout of the corresponding gate, which in the following we assume be at most two.

Our matrix $\mathbf{A}_C$ has order $3n$. The $2n \times 2n$ leading principal submatrix of $\mathbf{A}_C$, denoted by $\mathbf{A'}_C$, is the *main submatrix*. The odd-numbered columns of $\mathbf{A'}_C$ are precisely the columns of the first half of the matrix $\mathbf{M}_C$. The even-numbered columns of $\mathbf{A'}_C$ are called the *auxiliary columns*; for $k = 1, \ldots, n$, column $2k$ of $\mathbf{A'}_C$ contains the entry 10 in position $2k$, and zero elsewhere. Outside the main submatrix, the entries a_{ij} of $\mathbf{A}_C$ are all zero except for $a_{2n+k,2k} = 20$ and $a_{2n+k,2n+k} = 30$, $k = 1, \ldots, n$. For example, for the circuit of Figure 10, the corresponding matrix $\mathbf{A}_C$ is shown in Figure 11. Note that $\mathbf{A}_C$ is strongly diagonally dominant and hence strongly nonsingular.

Define the *circuit area* of the matrix $\mathbf{A}_C$ to be the set of odd-numbered columns of the main submatrix. Also, for $i = 1, \ldots, n$, let

$$w_i = \sum_{j=2i+1}^{2n} |a_{2n+i,j}|.$$

The value w_i can be seen as the *weight* of certain entries in row $2n+i$ (see Figure 14).

The proof of the theorem is now a consequence of property **p1** (defined in section 2) and of the following lemma.

LEMMA C.1. *For $k = 1, \ldots, n$ the following facts hold for GEP on input A_C:*

(a) *The pivot for step $2k-1$ is either -3.9 or 4;*

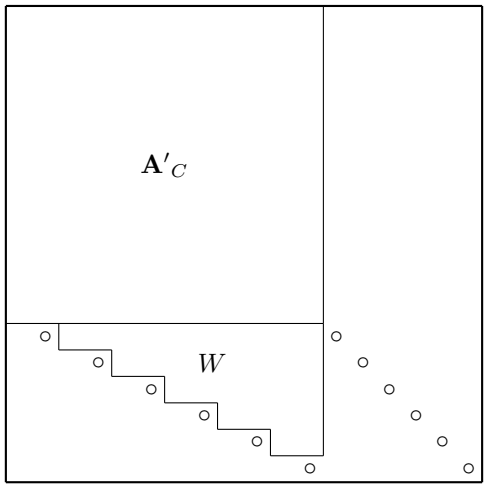


FIG. 14. *The structure of the matrix* $\mathbf{A}_C$. *The area labeled* W *contains the entries which contribute to the weights* w_i*'s. The symbol* $\circ$ *denotes the position of the initial nonzero entries outside the main submatrix* $\mathbf{A'}_C$.

(b) *step* $2k-1$ *modifies the circuit area as the* k*th step in the elimination of matrix* $\mathbf{M}_C$*;*

(c) *step* $2k-1$ *does not affect the auxiliary (even-numbered) columns with index greater than* $2k$*;*

(d) *the pivot for step* $2k$ *is* 20.0;

(e) *step* $2k$ *affects neither the circuit area nor the auxiliary columns with index greater than* $2k$*;*

(f) *at the end of step* $2k$, $w_i \leq 2.5$, *for* $i = 1, 2, \ldots, n$.

Proof. The proof is by complete induction on k. The basis is trivial. For the induction hypothesis, let $k > 1$ and suppose that (a) through (f) hold for any $i < k$.

Consider step $2k-1$ of GEP. By the induction hypothesis, the entries in column $2k-1$ with row index less than $2n$ contain the same values generated during the elimination process on the matrix $\mathbf{M}_C$. This means that $a_{2k-1,2k-1} = -3.9$ and

$$a_{2k,2k-1} = \begin{cases} 4 & \text{if the output of gate } k \text{ is } \texttt{False}, \\ 3.75 \text{ or } 3.50 & \text{otherwise.} \end{cases}$$

Moreover, for $2k < i \leq 2n$, we have $|a_{i,2k-1}| \leq 1.5$ (see part (d) of Lemma 3.2 in [24]). For the entries with row index larger than $2n$, we have, for $2n < i < 2n + k$,

$$|a_{i,2k-1}| \leq w_{i-2n} \leq 2.5,$$

because of (f), and, for $i \geq 2n + k$, we have $a_{i,2k-1} = 0$ by property **p1**. It follows that the pivot is 4 if the output of the gate k is `False`, and -3.9 otherwise, hence proving part (a) and, consequently, part (b).

To prove (c) we first observe that the pivot at step $2k-1$ is either $a_{2k-1,2k-1}$ or $a_{2k,2k-1}$. By the induction hypothesis we know that the first $2k-2$ elimination steps did not affect the auxiliary columns $2k, 2k+2, \ldots, 2n$. Hence, for j even, $2k < j \leq 2n$, at the beginning of step $2k-1$ both $a_{2k-1,j}$ and $a_{2k,j}$ are zero. It follows that, regardless of the pivot, step $2k-1$ does not affect column j.

In order to prove (d), consider step $2k$ of GEP. We need to show that the element with the largest modulus in column $2k$ at the beginning of step $2k$ is $a_{2n+k,2k}$ which,

$$\begin{pmatrix} -3.9 & 0 & \cdots & 0 & \cdots & 0 \\ 4 & 10 & \cdots & 1 & \cdots & 1 \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & 0 & \cdots & y_r & \cdots & y_s \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & 0 & \cdots & z_r & \cdots & z_s \\ \vdots & \vdots & & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \vdots \\ \alpha_1 & \alpha_2 & \cdots & \alpha_r & \cdots & \alpha_s \end{pmatrix} \longrightarrow \begin{pmatrix} 4 & 10 & \cdots & 1 & \cdots & 1 \\ 0 & \frac{39}{4} & \cdots & \frac{39}{40} & \cdots & \frac{39}{40} \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & -\frac{5}{2} & \cdots & y_r - \frac{1}{4} & \cdots & y_s - \frac{1}{4} \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & -\frac{5}{2} & \cdots & z_r - \frac{1}{4} & \cdots & z_s - \frac{1}{4} \\ \vdots & \vdots & & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \vdots \\ 0 & \alpha_2 - \frac{5}{2}\alpha_1 & \cdots & \alpha_r - \frac{1}{4}\alpha_1 & \cdots & \alpha_s - \frac{1}{4}\alpha_1 \end{pmatrix}.$$

FIG. 15. *The effect of the elimination of column $2k-1$ over columns $2k, r, s$ when the pivot is 4. Since the fanout of each gate is at most 2 no other column in the main submatrix is affected. We also show how step $2k-1$ modifies a row outside the main submatrix.*

by property **p1**, is equal to 20.0. Also by property **p1**, we know that at step $2k$ we have $a_{i,2k} = 0$ for $2n + k < i \leq 3n$. Hence, we need only to prove that $|a_{i,2k}| < 20.0$ for $2k \leq i < 2n + k$. We consider two cases. If at step $2k - 1$ the pivot is -3.9, the annihilation of column $2k - 1$ does not affect column $2k$. Hence, at the beginning of step $2k$ we have $a_{2k,2k} = 10$, $a_{i,2k} = 0$, for $2k < i \leq 2n$, and, for $2n < i < 2n + k$, $|a_{i,2k}| \leq w_{i-2n} \leq 2.5$. Vice versa, if at step $2k - 1$ the pivot is 4, the elimination of column $2k-1$ does affect column $2k$. As a result, at the beginning of step $2k$ we have $a_{2k,2k} = 39/4$, and two entries in column $2k$ are equal to $-5/2$. For the entries $a_{i,2k}$, with $2n < i < 2n + k$, we have (see Figure 15) $a_{i,2k} = \alpha_2 - \frac{5}{2}\alpha_1$, where $|\alpha_i| \leq 2.5$ (by part (f) of the induction). Hence, $a_{2n+k,2k} = 20.0$ is the largest entry in column $2k$ and is the pivot chosen by GEP.

To prove (e) we simply note that all the entries in the pivot row with column index less than or equal to $2n$ are zero. Therefore, the annihilation of column $2k$ does not affect the main submatrix outside column $2k$.

To prove (f), we fix an index i and analyze how steps $2k - 1$ and $2k$ affect the weight $w_i = \sum_{j=2i+1}^{2n} |a_{2n+i,j}|$. Again we consider two cases, depending on the pivot chosen at step $2k - 1$. If this is -3.9, GEP sends $a_{2n+i,2k-1}$ to zero without affecting the other entries in row $2n + i$. Similarly, step $2k$ sends $a_{2n+i,2k}$ to zero without side effects. Thus, the weight w_i remains bounded by 2.5. A special case is when $i = k$. In fact, at the beginning of step $2k$ GEP swaps rows $2k$ and $2n + k$. In this case the new value w_k depends on the entries $a_{2k,j}$, with $2k < j \leq 2n$. By parts (b), (c), and (e) of the induction, one can see that at the beginning of step $2k$ there are at most four entries $a_{2k,j} \neq 0$ and that $\sum_{2k<j\leq 2n} |a_{2k,j}| \leq 2.5$. Hence, at the end of step $2k$ we have $w_k \leq 2.5$, as claimed. Suppose now that the pivot at step $2k - 1$ is 4. In this case the effects of step $2k - 1$ on row $2n + i$, $i \neq k$, are shown in Figure 15. Since step $2k$ simply sends to zero the entry in column $2k$ (i.e., $\alpha_2 - \frac{5}{2}\alpha_1$), the weight w_i decreases by an amount of at least $|\alpha_1|/2 + |\alpha_2|$. Finally, for $i = k$, reasoning as for the previous case, we get $w_k \leq 2.5$. □

REFERENCES

[1] E. ALLENDER, R. BEALS, AND M. OGIHARA, *The complexity of matrix rank and feasible systems of linear equations*, in Proc. 28th Annual IEEE Symp. on Theory of Computing (STOC), IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 161–167.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorenson, *Lapack Users' Guide*, SIAM, Philadelphia, PA, 1992.

[3] A. Borodin, J. von zur Gathen, and J. Hopcroft, *Fast parallel matrix and GCD computations*, Inform. and Control, 52 (1982), pp. 241–256.

[4] B. Codenotti, M. Leoncini, and F. P. Preparata, *On the role of arithmetic fast parallel matrix inversion*, Algorithmica, submitted.

[5] L. Csanky, *Fast parallel matrix inversion algorithms*, SIAM J. Comput., 5 (1976), pp. 618–623.

[6] J. W. Demmel, *Trading Off Parallelism and Numerical Stability*, Lapack Working Note 53, Tech. report ut-cs-92-179, Univ. of Tennessee, Knoxville, TN, June 1992. Available online at http://www.cs.utk.edu/~library/1992.html

[7] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK Users' Guide*, SIAM, Philadelphia, PA, 1979.

[8] W. Eberly, *Efficient parallel independent subsets and matrix factorizations*, in Proc. 3rd IEEE Symposium on Parallel and Distributed Processing, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 204–211.

[9] J. Von zur Gathen, *Parallel linear algebra*, in Synthesis of Parallel Algorithms, J. Reif, ed., Morgan–Kaufmann, San Mateo, CA, 1993, pp. 573–617.

[10] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.

[11] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo, *Limits to Parallel Computation*, Oxford University Press, New York, NY, 1995.

[12] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays Trees Hypercubes*, Morgan–Kaufmann, San Mateo, CA, 1992.

[13] M. Leoncini, *On the parallel complexity of Gaussian elimination with pivoting*, J. Comput. System Sci., 53 (1996), pp. 380–394.

[14] M. Leoncini, G. Manzini, and L. Margara, *Parallel complexity of Householder QR factorization*, in Proc. European Symp. on Algorithms, Lecture Notes in Comput. Sci. 1136, Springer-Verlag, New York, 1996, pp. 290–301.

[15] M. Leoncini, G. Manzini, and L. Margara, *On the parallel complexity of matrix factorization algorithms*, in Proc. 9th ACM Symp. on Parallel Algorithms and Architectures, ACM, New York, 1997, pp. 63–71.

[16] M. Leoncini, G. Manzini, and L. Margara, *Companion MATLAB© software to "Parallel Complexity of Numerically Accurate Linear System Solvers,"* available online at http://www.imc.pi.cnr.it/ manzini/transducer/

[17] *Using Matlab* 5.1, The Mathworks Inc., Natick, MA, 1997.

[18] J. J. Modi and M. R. B. Clarke, *An alternative Givens ordering*, Numer. Math., 43 (1984), pp. 83–90.

[19] V. Pan, *Complexity of parallel matrix computations*, Theoret. Comput. Sci., 54 (1987), pp. 65–85.

[20] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York, NY, 1985.

[21] J. H. Reif, *$O(\log^2 n)$ time efficient parallel factorization of dense, sparse separable, and banded matrices*, in Proc. 6th ACM Symp. on Parallel Algorithms and Architectures, ACM, New York, 1994, pp. 278–289.

[22] A. H. Sameh and R. P. Brent, *Solving triangular systems on a parallel computer*, SIAM J. Numer. Anal., 14 (1977), pp. 1101–1113.

[23] A. H. Sameh and D. J. Kuck, *On stable parallel linear system solvers*, J. ACM, 25 (1978), pp. 81–91.

[24] S. A. Vavasis, *Gaussian elimination with pivoting is P-complete*, SIAM J. Discrete Math., 2 (1989), pp. 413–423.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2059–2089

APPROXIMATE COMPLEX POLYNOMIAL EVALUATION IN NEAR CONSTANT WORK PER POINT*

JOHN H. REIF†

Abstract. Given the n complex coefficients of a degree $n-1$ complex polynomial, we wish to evaluate the polynomial at a large number $m \geq n$ of points on the complex plane. This problem is required by many algebraic computations and so is considered in most basic algorithm texts (e.g., [A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974]). We assume an arithmetic model of computation, where on each step we can execute an arithmetic operation, which is computed exactly. All previous exact algorithms [C. M. Fiduccia, *Proceedings 4th Annual ACM Symposium on Theory of Computing*, 1972, pp. 88–93; H. T. Kung, *Fast Evaluation and Interpolation*, Carnegie-Mellon, 1973; A. B. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numerical Problems*, American Elsevier, 1975; V. Pan, A. Sadikou, E. Landowne, and O. Tiga, *Comput. Math. Appl.*, 25 (1993), pp. 25–30] cost at least work $\Omega(\log^2 n)$ per point, and previously, *there were no known approximation algorithms for complex polynomial evaluation within the unit circle with work bounds better than the fastest known exact algorithms.* There are known approximation algorithms [V. Rokhlin, *J. Complexity*, 4 (1988), pp. 12–32; V. Y. Pan, J. H. Reif, and S. R. Tate, in *Proceedings 32nd Annual IEEE Symposium on Foundations of Computer Science*, 1992, pp. 703–713] for polynomial evaluation at real points, but these do not extend to evaluation at general points on the complex plane.

We provide approximation algorithms for complex polynomial evaluation that cost, in many cases, near constant amortized work per point. Let $k = \log(|P|/\epsilon)$, where $|P|$ is the sum of the moduli of the coefficients of the input polynomial $P(z)$. Let $\tilde{P}(z_j)$ *be an* ϵ*-approx of* $P(z)$ if ϵ upper bounds the modulus of the error of the approximation $\tilde{P}(z_j)$ at each evaluation point z_j, that is, $|P(z_j) - \tilde{P}(z_j)| \leq \epsilon$; note that ϵ is an absolute error bound rather than a relative error bound. In many applications (particularly in signal processing) the evaluation points z_j are fixed and require only polylogarithmic $k = \log(|P|/\epsilon) = O(\log^{O(1)} n)$; for these cases we get a surprising reduction in work by use of approximation algorithms, as compared to the fastest known exact algorithms.

We ϵ-approx complex degree $n-1$ polynomial evaluation at $m \geq n \log n / \log^2 k$ fixed points on or within the unit disk in the complex plane in amortized work $O(\log^2 k)$ per point, which is $O(\log^2 \log n)$ for polylogarithmic k. If the m points are not fixed, then we have increased amortized work $O(\log^2 k + \log m)$ per point, which is $O(\log m)$ for polylogarithmic k and $m \geq n \log n / \log k$, and is still substantially below the previous bound of $\Omega(\log^2 m)$ for known exact algorithms. We further reduce our amortized bounds for special sets of evaluation points widely used in signal processing applications. The *chirp transform* is equivalent to evaluating a complex degree $n-1$ polynomial at the *chirp points*, which are $\zeta^j, j = 0, \ldots, m-1$, for some fixed complex number ζ. We ϵ-approx complex degree $n-1$ polynomial evaluation at these m chirp points, where $m \geq n \log n / \log^2 k$ and $|\zeta| \leq 1$ in amortized work $O(\log k)$ per point, whereas the previous best bounds for exact evaluation (via the chirp transform) were $\Omega(\log m)$ per point [A. V. Aho, K. Steiglitz, and J. D. Ullman, *SIAM J. Comput.*, 4 (1975), pp. 533–539]. All these results use an interesting reduction to fast multipole algorithms for solving Trummer's problem.

Using instead a reduction to approximate real polynomial evaluation (by interpolation at the Chebyshev points), in total work $O(n \log k)$, we

- ϵ-approx the evaluation of a degree n polynomial at the first n powers of the n'th root of unity, where $n' \geq \Omega(n^2/k)$, and
- ϵ-approx the n-point DFT for certain inputs with descending coefficient magnitude.

All of our results require polylogarithmic (that is, $\log^{O(1)} n$) depth with the same work bounds. We also provide a lower bound for a wide class of schemes for approximate evaluation of a degree $n-1$ polynomial on the unit circle; namely, we prove that if a scheme uses an approximation polynomial of degree $k-1$, then it can be convergent only over a small fraction $O(k/n)$ of the unit circle. We believe this is the first lower bound of this sort proved, and the proof uses an interesting reduction

*Received by the editors July 14, 1997; accepted for publication (in revised form) May 26, 1998; published electronically June 23, 1999. This work was supported by NSF grant NSF-IRI-91-00681 and Army Research Office contract DAAH-04-96-1-0448. A preliminary version of this paper appeared in 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997.

http://www.siam.org/journals/sicomp/28-6/32429.html

†Department of Computer Science, Duke University, Durham, NC 27708-0129 (reif@cs.duke.edu, http://www.cs.duke.edu/∼reif/HomePage.html).

to the approximation of a matrix product by a matrix of reduced rank.

Key words. algebraic computation, discrete Fourier transform (DFT), fast Fourier transform (FFT), multipoint polynomial evaluation, complex plane, approximate algorithm

AMS subject classifications. 12Y05, 12-04, 65D15, 65D05, 41A21, 41A10, 68Q25

PII. S0097539797324291

1. Introduction.

The following subsections give an extended description of motivation, statement of the problem, and comparison with prior work.

1.1. Machine model.

For most of this paper, we assume an arithmetic circuit model of sequential computation, where each basic arithmetic or logical operation such as addition, subtraction, multiplication, division, and comparison over the domain of rational numbers can be exactly computed in one step. The floor and ceiling operations are not allowed in this model. (One of our results assumes an *extended arithmetic model*, where given a real θ, then $e^{i\theta} = \cos(\theta) + i\sin(\theta)$ can be computed in $O(1)$ steps. The assumptions made for the extended arithmetic model will be stated during presentation of our algorithms.) We also assume an arithmetic (EREW) PRAM model of parallel computation, where the processors can execute these arithmetic operations in parallel, with exclusive reading and exclusive writing into locations of the shared memory. The computation is *sequential* if there is only one processor. The resource metrics of this model are: *parallel time* (the time to compute the outputs) and *work*, which is the total number of such steps summed over all processors.

1.2. Multipoint polynomial evaluation and interpolation.

The *multipoint polynomial evaluation problem* over ring $\mathcal{D}$ is defined as follows:
Input: n coefficients $p_0, \ldots, p_{n-1} \in \mathcal{D}$ defining a polynomial

$$P(z) = \sum_{j=0}^{n-1} p_j z^j$$

of degree $n-1$ and m evaluation points

$$z_0, \ldots, z_{m-1} \in \mathcal{D}.$$

Output: The values $P(z_0), \ldots, P(z_{m-1})$.

The *multipoint polynomial interpolation problem* over ring $\mathcal{D}$ is defined as follows:
Input: n distinct interpolation points $z_0, \ldots, z_{n-1} \in \mathcal{D}$ and n values $y_0, \ldots, y_{n-1} \in \mathcal{D}$.
Output: coefficients $p_0, \ldots, p_{n-1} \in \mathcal{D}$ defining the unique polynomial

$$P(z) = \sum_{j=0}^{n-1} p_j z^j$$

of degree $n-1$ such that $y_j = P(z_j)$ for $j = 0, \ldots, n-1$.

The multipoint *complex (real)* polynomial evaluation and interpolation problems restrict $\mathcal{D}$ to the complex numbers $\mathcal{C}$ (real numbers $\mathcal{R}$, respectively). Exact algorithms for multipoint polynomial evaluation and interpolation use modular techniques developed in the initial work of Moenck and Borodin [27], Borodin and Munro [4], Horowitz [25], Fiduccia [16], and Kung [26] and were improved by Borodin and Munro [5] to the best known work bounds: $O((n+m)\log^2(n+m))$ for evaluation and $O(n\log^2 n)$ for interpolation. An error analysis for these algorithms is given by Newbery [28]. Pan et al. [31] also gave polynomial evaluation and interpolation algorithms with at least this same work, which in certain cases provide improved numerical stability.

1.3. Multipoint polynomial evaluation at a small number of points. A special case of the multipoint polynomial evaluation problem is where the polynomial $P(z)$ has degree $n-1$, where n is much larger than the number m of evaluation points $z_0, \ldots, z_{m-1}$. This special case has been well known since the early days of algebraic computation and is frequently used in recursive algorithms for multipoint polynomial evaluation.

PROPOSITION 1.1. *The work to evaluate a polynomial $P(z)$ of degree $n-1$ at $m < n$ evaluation points $z_0, \ldots, z_{m-1}$ is $O(n)$ plus $\lceil n/m \rceil$ times the work to evaluate a polynomial of degree $m-1$ at m evaluation points over the same domain.*

Proof. Given $P(z)$ of degree $n-1$, we define $\lceil n/m \rceil$ polynomials

$$P_0(z), \ldots, P_{\lceil n/m \rceil - 1}(z)$$

of degree at most $m-1$, where

$$P(z) = \sum_{j=0}^{\lceil n/m \rceil - 1} P_j(z) z^{jm}.$$

To initialize, we precompute the (trivial) multipoint evaluation z_k^m for each $k = 0, \ldots, m-1$. Then for each $j = 0, \ldots, \lceil n/m \rceil - 1$ we do the multipoint evaluation $P_j(z_k)$ for $k = 0, \ldots, m-1$ and also compute each z_k^{jm} by multiplication of z_k^m times $z_k^{(j-1)m}$. ∎

By application of Proposition 1.1, combined with previous exact algorithms [27, 16, 26, 5] for m point polynomial evaluation for $\lceil n/m \rceil$ polynomials of degree $m-1$, which require work $O(m \log^2 m)$ each, we have the m point polynomial evaluation problem, for $m \leq n$, which can be computed within work

$$\lceil n/m \rceil (m \log^2 m) \leq O(n \log^2 m).$$

Also, the multipoint polynomial evaluation problem, for $m \geq n$, can be solved within work

$$m/n O(n \log^2 n) \leq O(m \log^2 n).$$

Hence we have the following proposition.

PROPOSITION 1.2. *The multipoint polynomial evaluation problem costs work* $\leq O((m+n) \log^2 \min(n, m))$.

1.4. The DFT and generalizations. An nth root of unity ω satisfies $\omega^n = 1$ and $\omega^j \neq 1$ for $1 \leq j < n$. The *nth root of unity over the complex numbers* $\mathcal{C}$ is $\omega = e^{i2\pi/n}$, where $i = \sqrt{-1}$. The *n roots of unity over* $\mathcal{C}$ are $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

The *discrete Fourier transform (DFT) problem* over the complex numbers $\mathcal{C}$ is defined as follows:

Input: coefficients $p_0, \ldots, p_{n-1} \in \mathcal{D}$ defining a polynomial

$$P(z) = \sum_{j=0}^{n-1} p_j z^j$$

of degree $n-1$.

Output: The values

$$P(\omega^0), P(\omega^1), \ldots, P(\omega^{n-1}).$$

The celebrated *fast Fourier transform (FFT)* algorithm of Cooley and Tukey [9] (which, according to Cooley, Lewis, and Welch [8], originates with early work by Runge and König [36] and had early use in the works of Danielson and Lanczos [11] and Good [20]) provides the DFT in work $O(n \log n)$ (also see Gentleman and Sande [17] and Rabiner and Rader [32] for efficient implementations of the DFT).

The DFT^{-1}, which is the inverse problem to the DFT (that is, interpolation from the n roots of unity), reduces to multiplying $\frac{1}{n}$ by the evaluation of a given polynomial at the inverses of each of the n roots of unity. Since for each of the n roots of unity ω^j, the inverse $\omega^{-j} = \omega^{n-j}$ is also one of the n roots of unity (now reverse ordered), the DFT^{-1} can also be solved via the FFT in work $O(n \log n)$. The chirp transform generalizes the FFT to the evaluation of a polynomial of degree $n-1$ over the points ζ^j, for a complex constant ζ. The chirp transform can be computed in work $O(n \log n)$ by a generalization of the FFT algorithm. By a reduction to convolution and thus DFT, the following has also been shown.

PROPOSITION 1.3 (see Aho, Steiglitz, and Ullman [2]). *For fixed complex constants s_0, s_1, ζ, the generalized chirp transform problem of evaluation of a polynomial of degree $n-1$ over the points $s_0 + s_1\zeta^j$ for $j = 0, \ldots, n-1$ (and also the inverse of this problem, that of interpolation from these chirp points) can be solved within work $O(n \log n)$.*

By application of Proposition 1.1, we have the following proposition.

PROPOSITION 1.4. *For fixed complex constants s_0, s_1, ζ, the generalized chirp transform problem of evaluation of a polynomial of degree $n-1$ for over the points $s_0 + s_1\zeta^j$ for $j = 0, \ldots, m-1$ can be solved within work $O((m+n)\log \min(n, m))$.*

Recently, Dutt and Rokhlin, [14] gave an ϵ-approx algorithm for evaluation of a degree $n-1$ polynomial at $n-1$ points on a unit circle with work $O(n \log n + n \log(1/\epsilon))$ and Dutt, Gu, and Rokhlin [12] gave an ϵ-approx algorithm for interpolation of a degree $n-1$ polynomial from $n-1$ Chebyshev points on a unit circle with work $O(n \log(1/\epsilon))$.

1.5. Organization of this paper. The results of this paper are summarized in the abstract. Section 1 provides the standard definitions of the assumed arithmetic model of computation, multipoint polynomial evaluation and interpolation, the DFT, and generalizations to the chirp transform (see also Aho, Steiglitz, and Ullman [2]).

Section 2 gives our *main result*, Theorem 2.7, an approximate complex polynomial evaluation algorithm using a reduction to Trummer's problem which we approximately solve by multipole algorithms. We first define Trummer's problem in subsection 2.1 and describe Multipole methods for approximately solving Trummer's problem in section 2.2. We then give an algorithm for multipoint complex polynomial re-evaluation by reduction to Trummer's problem in section 2.3, and finally in section 2.4 we use this reduction to do approximate complex polynomial evaluation via DFT and multipole algorithms.

Note. Our computational model assumes each arithmetic operation yields exact results; thus, it is not yet known if our algorithm yields the performance given in theory if each arithmetic operation is computed in approximate floating point. However, implementations (see [22, 15]) of fast multipole algorithms using approximate floating point operation do give excellent performance in practice.

Section 3 describes known results for approximate real polynomial evaluation; section 3.1 defines the Chebyshev point evaluation problem and known algorithms; section 3.2 bounds the errors of interpolation at the Chebyshev points; and section 3.3

describes known methods for approximate real polynomial evaluation via interpolation at the Chebyshev points.

Section 4 has a *secondary (less general) result:* it gives approximate polynomial evaluation on a circle in a number of interesting cases, using classical methods of real approximation by interpolation at the Chebyshev points. Subsection 4.1 describes approximate polynomial evaluation on a circle via interpolation at the Chebyshev angles and section 4.2 gives an alternative method which is proved in Appendix A.

Section 5 proves a lower bound for a wide class of schemes for polynomial evaluation on the unit circle; namely, there is no general approximation method, using a low degree polynomial, that is convergent over a large fraction of the unit circle.

Appendix B gives an algorithm for exact Chebyshev point evaluation in $O(n \log n)$ work, which is useful for the approximate real polynomial evaluation results of section 3 (this can be used as an alternative for a somewhat more complex algorithm for Chebyshev point evaluation of Gerasoulis [18]).

2. Approximate complex polynomial evaluation via the multipole methods.

2.1. Trummer's problem. We define the *(generalized) Trummer's problem* over the complex plane $\mathcal{C}$:

Input: n points $a_0, \ldots, a_{n-1} \in \mathcal{C}$, n weights $c_0, \ldots, c_{n-1} \in \mathcal{C}$, defining a rational function $\psi(z) = \sum_{j=0}^{n-1} \frac{c_j}{z-a_j}$, and also $m \geq n$ evaluation points $z_0, \ldots, z_{m-1} \in \mathcal{C}$.

Output: The values $\psi(z_0), \ldots, \psi(z_{m-1})$.

Trummer's problem has widespread application to calculation of electrostatic and gravitational forces. Gerasoulis [18] (also see Gerasoulis, Grigoriadis, and Sun [19]) gave an exact algorithm for Trummer's problem, requiring $O(m \log^2 m)$ work. For the case where $m = n$ and $a_j = z_j$, for $j = 1, \ldots, n$, Gerasoulis defined a polynomial $\sigma(z) = \Pi_{j=0}^{n-1}(z - a_j)$ and its derivative $\sigma^{(1)}(z) = \frac{d\sigma(z)}{dz}$. By *l'Hôpital's* rule (the limit is preserved by taking derivatives), we have the following proposition.

PROPOSITION 2.1.

$$\psi(a_j) = \frac{c_j}{\sigma^{(1)}(a_j)}$$

for $j = 0, \ldots, n-1$

Thus Gerasoulis gave a reduction of the probem of evaluation of $\psi(z)$ to evaluation of $\sigma^{(1)}(z)$ at the n points $a_0, \ldots, a_{n-1}$, which requires $O(n \log^2 n)$ work for the exact solution of Trummer's problem. The result of Gerasoulis also easily extends (e.g., by use of additional weights c_j with value 0) to allow for exact solution of the generalized Trummer's problem, as defined above, in $O(m \log^2 m)$ work.

2.2. Multipole methods. Let $k = \log(|C|/\epsilon)$, where $|C| = \sum_{j=0}^{n-1} |c_j|$ is the sum of the moduli (the modulus of complex number $re^{i\theta}$ is r) of the weights $c_0, \ldots, c_{n-1}$ and ϵ is a given absolute error bound on each output. Greengard and Rokhlin [21] (also see Carrier, Greengard, and Rokhlin [7]) gave an ϵ-approx algorithm, known as the multipole algorithm, for Trummer's problem using an $O(k)$ term rational series expansion of $O(k)$ terms and requiring $O(mk^2)$ work over the complex plane. They computed ϵ-approx values $\widetilde{\psi}(a_0), \ldots, \widetilde{\psi}(a_m)$ such that $|\psi(a_j) - \widetilde{\psi}(a_j)| \leq \epsilon$ for $j = 0, \ldots, m-1$. Given the n points $a_0, \ldots, a_{n-1} \in \mathcal{C}$ for Trummer's problem, rational function $\psi(z)$, and also $m \geq n$ evaluation points $z_0, \ldots, z_{m-1} \in \mathcal{C}$, the multipole algorithm requires using these $m+n$ points to construct a certain tree-like data structure known as the *well-separated decomposition*; see Callahan and Kosaraju [6] for details.

Given these $m+n$ points, the well-separated decomposition algorithm of Callahan and Kosaraju [6] costs work $O(m \log m)$. Also, Pan, Reif, and Tate [30] (see Reif and Tate [33] for the full paper) gave an $O(m \log\log m)$ algorithm for the well-separated decomposition in the case where the input points have logarithmic bit-precision. Assuming a well-separated decomposition, the Multipole algorithm over the complex plane was improved by Greengard and Rokhlin [22] to $O(mk \log k)$ work by use of the FFT, to do each of the $O(m)$ operations on $O(k)$ term power series required by this multipole algorithm, each within work $O(k \log k)$. Later work by Pan, Reif, and Tate [30] (see Reif and Tate [34] for the full paper) gave a further substantial improvement, which remains the most efficient known algorithm for approximate solution of the Trummer's problem.

LEMMA 2.2. *Given a well-separated decomposition of the set of the input points, a Trummer's problem for m evaluation points can be ϵ-approx on every output within work $O(m \log^2 k)$, where $k = \log(|C|/\epsilon)$.*

The algorithm of [30, 34] uses a reduction from Trummer's problem for m evaluation points to ϵ-approx solution of a (slightly generalized) Trummer's problem of size $m' \leq m/k^c$ for a constant c with cost $O(m'k \log k) \leq O(m)$, and also solves $O(m')$ instances of Trummer's problem; each has $m/m' \leq k^c$ evaluation points costing $O((m/m') \log^2 k)$ per such instance, resulting in the total cost

$$O(m'(m/m') \log^2 k) \leq O(m \log^2 k).$$

Note. In the special case that the m evaluation points are chirp points, the algorithm of [30, 34] can be sped up as follows: again we compute an ϵ-approx solution of a (slightly generalized) Trummer's problem of size $m' \leq m/k^c$ with cost $O(m'k \log k)$ $\leq O(m)$ and also solve $O(m')$ instances of Trummer's problem with $m/m' \leq k^c$ chirp evaluation points, with the reduced cost $O((m/m') \log k)$ for each such instance, resulting in a somewhat reduced total cost $O(m'(m/m') \log k) \leq O(m \log k)$.

2.3. Reduction to Trummer's problem. The *multipoint polynomial re-evaluation problem* over $\mathcal{C}$ is defined as follows:

Input: n distinct points $a_0, \ldots, a_{n-1} \in \mathcal{C}$ and n values $y_0, \ldots, y_{n-1} \in \mathcal{C}$ defining a unique polynomial $P(z)$ of degree $n-1$ such that $y_j = P(a_j)$ for $j = 0, \ldots, n-1$, and m re-evaluation points $z_0, \ldots, z_{m-1} \in \mathcal{C}$, which are distinct from these points $a_0, \ldots, a_{n-1}$.

Output: The values $P(z_0), \ldots, P(z_{m-1})$.

Note that in the polynomial re-evaluation problem, the coefficients of the input polynomial are not explicitly given.

Given as input n distinct points $a_0, \ldots, a_{n-1} \in \mathcal{C}$, and values $y_0, \ldots, y_{n-1}$, let $\sigma(z) = \Pi_{j=0}^{n-1}(z - a_j)$. Our basic approach is as follows. We construct the rational function $\psi(z) = P(z)/\sigma(z)$, which can be expanded as a Trummer's function $\psi(z)$ $= \sum_{j=0}^{n-1} \frac{c_j}{z-a_j}$. The weights $c_0, \ldots, c_{n-1} \in \mathcal{C}$ are determined by reversing the formula of Proposition 2.1, as follows in this proposition.

PROPOSITION 2.3. *Suppose we set $c_j = y_j/\sigma^{(1)}(a_j)$ for $j = 0, \ldots, n-1$, where $\sigma^{(1)}(z) = \frac{d\sigma(z)}{dz}$. Then $P(z) = \psi(z)\sigma(z)$ for all z, where $\psi(z) = \sum_{j=0}^{n-1} \frac{c_j}{z-a_j}$, and $P(z)$ is the unique degree $n-1$ polynomial such that $P(a_0) = y_0, \ldots, P(a_{n-1}) = y_{n-1}$.*

Proof. If $P(z) = \psi(z)\sigma(z)$, then for $j = 0, \ldots, n-1$,

$$c_j = \lim_{z \to a_j} (z - a_j)\psi(z) = \lim_{z \to a_j} (z - a_j)(P(z)/\sigma(z))$$

$$= P(a_j)/\sigma^{(1)}(a_j) = y_j/\sigma^{(1)}(a_j). \qquad \square$$

We will exactly compute the values $\sigma^{(1)}(a_j)$, $j = 0, \ldots, n-1$. We will compute or approximate (as in certain specialized cases described below) the values $\sigma(z_j)$, $j = 0, \ldots, m-1$. Next we apply the efficient multipole method of [30, 34] (Lemma 2.2) to construct an ϵ-approx solution $\widetilde{\psi}(z_0), \ldots, \widetilde{\psi}(z_{m-1})$ of this Trummer's problem, such that $|\psi(z_j) - \widetilde{\psi}(z_j)| \leq \epsilon$ for $j = 0, \ldots, m-1$. If we exactly compute $\sigma(z_j)$, then we approximate each $P(z_j)$ by $\widetilde{P}(z_j) = \widetilde{\psi}(z_j)\sigma(z_j)$. Otherwise, we approximate each $P(z_j)$ by $\widetilde{P}(z_j) = \widetilde{\psi}(z_j)\widetilde{\sigma}(z_j)$, where $\widetilde{\sigma}(z_j)$ is an ϵ^*-approx to $\sigma(z_j)$ (as determined below).

PROPOSITION 2.4. *For $j = 0, \ldots, m-1$, the $\widetilde{P}(z_j)$ are an $\widehat{\epsilon}$-approx to the $P(z_j)$, where if the $\sigma(z_j)$ are exactly computed, then*

$$\widehat{\epsilon} \leq \epsilon \max_j |\sigma(z_j)|,$$

and if each $\sigma(z_j)$ is ϵ^*-approx by $\widetilde{\sigma}(z_j)$, then

$$\widehat{\epsilon} \leq (\epsilon^* + \max_j |\sigma(z_j)|)\epsilon + (2 + 3\epsilon^*/\min_j |\sigma(z_j)|)|P|,$$

where $|P| = \sum_{j=0}^{n-1} |p_j|$.

Proof. First suppose the $\sigma(z_j)$ are exactly computed, so $\epsilon^* = 0$. By definition, $P(z_j) = \psi(z_j)\sigma(z_j)$ and $\widetilde{P}(z_j) = \widetilde{\psi}(z_j)\sigma(z_j)$, so we have

$$\begin{aligned} |P(z_j) - \widetilde{P}(z_j)| &= |\psi(z_j)\sigma(z_j) - \widetilde{\psi}(z_j)\sigma(z_j)| \\ &= |\psi(z_j) - \widetilde{\psi}(z_j)||\sigma(z_j)| \leq \epsilon|\sigma(z_j)|. \end{aligned}$$

Otherwise, suppose each $\sigma(z_j)$ is ϵ^*-approx by $\widetilde{\sigma}(z_j)$. We have $|\psi(z_j)| \leq |P|/|\sigma(z_j)|$, and $|\widetilde{\psi}(z_j)| \leq \epsilon + |\psi(z_j)| \leq \epsilon + |P|/|\sigma(z_j)|$, so

$$|\widetilde{\psi}(z_j)||\sigma(z_j)| \leq |\sigma(z_j)|\epsilon + |P|,$$

and also

$$|\psi(z_j)| + |\widetilde{\psi}(z_j)| \leq \epsilon + 2|\psi(z_j)| \leq \epsilon + 2|P|/|\sigma(z_j)|.$$

Furthermore, $|\sigma(z_j) - \widetilde{\sigma}(z_j)| \leq \epsilon^*$, so $|\widetilde{\sigma}(z_j)| \leq \epsilon^* + |\sigma(z_j)|$, and

$$|\psi(z_j)\widetilde{\sigma}(z_j)| \leq \epsilon^*|\psi(z_j)| + |P| = \epsilon^*(|P|/|\sigma(z_j)|) + |P|.$$

Also,

$$(|\psi(z_j)| + |\widetilde{\psi}(z_j)|)|\sigma(z_j) - \widetilde{\sigma}(z_j)| \leq \epsilon^*(\epsilon + 2|P|/|\sigma(z_j)|).$$

Note that

$$\begin{aligned} P(z_j) - \widetilde{P}(z_j) &= \psi(z_j)\sigma(z_j) - \widetilde{\psi}(z_j)\widetilde{\sigma}(z_j) \\ &= (\psi(z_j) + \widetilde{\psi}(z_j))(\sigma(z_j) - \widetilde{\sigma}(z_j)) + \psi(z_j)\widetilde{\sigma}(z_j) - \widetilde{\psi}(z_j)\sigma(z_j). \end{aligned}$$

Hence

$$\begin{aligned} |P(z_j) - \widetilde{P}(z_j)| &= |\psi(z_j)\sigma(z_j) - \widetilde{\psi}(z_j)\widetilde{\sigma}(z_j)| \\ &\leq (|\psi(z_j)| + |\widetilde{\psi}(z_j)|)|\sigma(z_j) - \widetilde{\sigma}(z_j)| + |\psi(z_j)\widetilde{\sigma}(z_j)| + |\widetilde{\psi}(z_j)\sigma(z_j)| \\ &\leq \epsilon^*(\epsilon + 2|P|/|\sigma(z_j)|) + (\epsilon^*(|P|/|\sigma(z_j)|) + |P|) + (|\sigma(z_j)|\epsilon + |P|) \\ &= (\epsilon^* + |\sigma(z_j)|)\epsilon + (2 + 3\epsilon^*/|\sigma(z_j)|)|P|. \qquad \square \end{aligned}$$

To approximately compute each of the $\sigma(z_j)$, given z_j (and with fixed a_j) for $j = 0, \ldots, m-1$, we define a *unitary* Trummer's problem of size m where we specialize the y_j to 1 but keep the a_j fixed as before. Hence, since the a_j are fixed, the weights are fixed as $c_j = 1/\sigma^{(1)}(a_j)$, and so can be assumed to be precomputed. This corresponds to defining a constant polynomial $P^*(z) = 1 = \psi^*(z)\sigma(z)$; with this specialization, $\psi^*(z) = 1/\sigma(z)$. Next we can apply again Lemma 2.2 to construct an ϵ-approx solution: $\widetilde{\psi}^*(z_0), \ldots, \widetilde{\psi}^*(z_{m-1})$ of the unitary Trummer's problem. Then we define $\widetilde{\sigma}(z) = 1/\widetilde{\psi}^*(z)$, and use $\widetilde{\sigma}(z_j)$ as an approximation to $\sigma(z_j)$.

PROPOSITION 2.5. *For $j = 0, \ldots, n-1$, each*

$$\widetilde{\sigma}(z_j) = 1/\widetilde{\psi}^*(z_j)$$

is an ϵ^*-approx to $\sigma(z_j)$, where

$$\epsilon^* = \frac{\epsilon|\sigma(z_j)|^2}{1 - |\sigma(z_j)|\epsilon}.$$

Proof. The error is bounded as

$$|\widetilde{\sigma}(z_j) - \sigma(z_j)| = |\frac{1}{\widetilde{\psi}^*(z_j)} - \frac{1}{\psi^*(z_j)}| = |\frac{\widetilde{\psi}^*(z_j) - \psi^*(z_j)}{\psi^*(z_j)\widetilde{\psi}^*(z_j)}| = \frac{|\widetilde{\psi}^*(z_j) - \psi^*(z_j)|}{|\psi^*(z_j)\widetilde{\psi}^*(z_j)|}$$
$$\leq \frac{\epsilon}{|\psi^*(z_j)||\widetilde{\psi}^*(z_j)|} \leq \frac{\epsilon}{|\psi^*(z_j)|(|\psi^*(z_j)| - \epsilon)} = \epsilon^*$$

for

$$\epsilon^* = \frac{\epsilon|\sigma(z_j)|^2}{1 - |\sigma(z_j)|\epsilon},$$

since $\sigma(z) = 1/\psi^*(z)$, and

$$|\widetilde{\psi}^*(z_j)| \geq |\psi^*(z_j)| - \epsilon = \frac{1 - |\sigma(z_j)|\epsilon}{|\sigma(z_j)|}. \qquad \square$$

2.4. Approximate complex polynomial evaluation via DFT and multipole algorithms. Let $\omega = e^{i2\pi/n}$ be the nth root of unity $\in \mathcal{C}$. We assume the n roots of unity ω^j, for $j = 1, \ldots, n-1$ are precomputed. Let us specialize the points $a_0, \ldots, a_{n-1}$ to be the n roots of unity, so $a_j = \omega^j$ for $j = 0, \ldots, n-1$. Then it is known (e.g., see Aho, Hopcroft, Ullman [1]) that $\sigma(z) = \Pi_{j=0}^{n-1}(z - \omega^j) = z^n - 1$. In this case, each $\sigma(z_j) = z_j^n - 1$ can be exactly computed by repeated squaring with work only $O(\log n)$. Since $\sigma^{(1)}(z) = nz^{n-1}$ in this case, and $\omega^n = 1$, each

$$\sigma^{(1)}(a_j) = \sigma^{(1)}(\omega^j) = n\omega^{j(n-1)} = n/\omega^j = n\omega^{n-j}.$$

Thus each $\sigma^{(1)}(a_j)$ costs one multiplication to compute from the known (precomputed) root of unity ω^{n-j}. Also,

$$\max_j |\sigma(z_j)| \leq 1 + \max_j |z_j|^n.$$

Note that if all the re-evaluation points z_j, for $j = 0, \ldots, m-1$, are on or within the unit disk,

$$\max_j |{z_j}^n| \leq \max_j |z_j|^n \leq 1.$$

This implies that

$$\max_j |\sigma(z_j)| \leq 1 + \max_j |{z_j}^n| \leq 2,$$

so by Proposition 2.4 (where we fix $\epsilon^* = 0$ since the $\sigma(z_j)$ are exactly computed) we have the following lemma.

LEMMA 2.6. *Let the points $a_0, \ldots, a_{n-1}$ be the n roots of unity, and all the re-evaluation points z_j, $j = 0, \ldots, m-1$ be on or within the unit disk, so $|z_j \leq 1$, and let the z_j be distinct from the n roots of unity. We construct an ϵ-approx solution $\widetilde{\psi}(z_0), \ldots, \widetilde{\psi}(z_{m-1})$ of the resulting Trummer's problem. Then for $j = 0, \ldots, m-1$, each $P(z_j)$ is 2ϵ-approx by*

$$\widetilde{P}(z_j) = \widetilde{\psi}(z_j)\sigma(z_j).$$

Now we bound the work. Given as input the coefficients of a degree $n-1$ polynomial $P(z)$, we can exactly evaluate $P(z)$ at the n roots of unity by applying the DFT algorithm to the coefficients of $P(z)$ in work $O(n \log n)$. Now given m evaluation points $z_0, \ldots, z_{m-1}$ on or within the unit disk, and distinct from the n roots of unity, we apply the above reduction to Trummer's problem. Let $|P| = \sum_{j=0}^{n-1} |p_j|$. Since $\sigma^{(1)}(z) = \frac{d\sigma(z)}{dz} = nz^{n-1}$, it follows that $|\sigma^{(1)}(\omega^j)| = n$. Hence the coefficients $c_j = P(\omega^j)/\sigma^{(1)}(\omega^j)$ of Trummer's problem have magnitude

$$|c_j| = |P(\omega^j)|/n = |P|/n$$

and have summed magnitude

$$|C| = \sum_{j=0}^{n-1} |c_j| = |P|.$$

We compute all the $\sigma(a_j)$ by $O(n)$ multiplications (of n times each of the precomputed roots of unity). If the input evaluation points $z_0, \ldots, z_{m-1}$ are fixed, then we can assume a precomputed well-separated decomposition of the set containing all the n roots of unity and the set of m evaluation points. Also, when the input evaluation points are fixed, we can assume precomputed $\sigma(z_j)$, $j = 0, \ldots, m-1$.

By the efficient multipole algorithm of [30, 34] (Lemma 2.2) the resulting Trummer's problem of size m can be ϵ-approx on every output within work $O(m \log^2 k)$, where $k = O(\log(|C|/\epsilon)) = O(\log(|P|/\epsilon))$. Thus if $m \geq (n \log n)/\log^2 k$, then the total work for m fixed evaluation points is $O(m \log^2 k)$. By Proposition 2.4, for $j = 0, \ldots, m-1$, the $\widetilde{P}(z_j)$ are an $\widehat{\epsilon}$-approx to the $P(z_j)$, where

$$\widehat{\epsilon} = \epsilon \max_j |\sigma(z_j)| \leq 2\epsilon,$$

since $\max_j |\sigma(z_j)| \leq 2$. Thus the approximation errors are upper bounded by 2ϵ, and rescaling (to simplify notation) 2ϵ to ϵ, we have the following theorem.

THEOREM 2.7. *Suppose we are given a complex degree $n-1$ polynomial $P(z)$, m fixed evaluation points on or within the unit disk (with a precomputed well-separated decomposition), and a given $\epsilon > 0$. Let $k = O(\log(|P|/\epsilon))$. Then we can compute an ϵ-approx of this complex polynomial evaluation problem within work $O(m \log^2 k + n \log n)$. If $m \geq (n \log n)/\log^2 k$, the amortized work per evaluation point is bounded by $O(\log^2 k)$.*

Note. For practical implementation of approximate polynomial evaluation via Theorem 2.7, one may substitute a theoretically less efficient multipole algorithm in place of the efficient multipole algorithm of [30, 34] (Lemma 2.2), which costs $O(m \log^2 k)$ work. For example, the FFT-accelerated fast multipole algorithm of Greengard and Rokhlin [22], which requires $O(mk \log k)$ work, in theory gives an improvement over the bounds of $O(n \log^2 n)$ for exact algorithms if $k \leq o((\log^2 n)/\log \log n)$ and moreover is known to be very efficient in practice. This FFT-accelerated fast multipole algorithm was implemented by Elliott and Board [15] on a variety of high performance machines, gives the currently fastest running multipole algorithm implementation known in practice (as opposed to theory) on these machines, and is used in many molecular simulation applications. Another approach would be to do a careful implementation of the multipole algorithm of Reif and Tate [30, 34], which may provide improved performance in practice, over the FFT-accelerated fast multipole algorithm.

Further note. We can reduce the work bounds of Theorem 2.7 for the special case of m chirp evaluation points, a Trummer's problem for m chirp evaluation points can be ϵ-approx on every output within work $O(m \log k)$. Thus the work bounds given in Theorem 2.8 can in this case be reduced by replacing $\log^2 k$ by $\log k$.

The case where the m evaluation points are not fixed. For small $k = o(n)$, the most costly parts of the above approximate complex polynomial evaluation algorithm is the computation of the $\sigma(z_j)$ and the well-separated decomposition, both costing work $O(m \log m)$ in the worst case. We now describe how to reduce this cost by approximate computation of the $\sigma(z_j)$ and, in certain cases, more efficient computation of the well-separated decomposition.

Again, we will assume that the input evaluation points $z_0, \ldots, z_{m-1}$ are on or within the unit disk; however, we redefine $a_j = r\omega^j$ where ω is the nth root of unity. Let $r = (1 + \frac{1}{n})$ and note $r^n \approx e$. Recall (e.g., see Aho, Hopcroft, Ullman [1]) that $\Pi_{j=0}^{n-1}(z - \omega^j) = z^n - 1$. In this case we redefine

$$\sigma(z) = \prod_{j=0}^{n-1}(z - r\omega^j) = r^n \prod_{j=0}^{n-1}\left(\frac{z}{r} - \omega^j\right) = r^n\left(\left(\frac{z}{r}\right)^n - 1\right) = z^n - r^n.$$

Since $|z_j| \leq 1$, we have

$$|\sigma(z_j)| = |z_j^n - r^n| \leq |z_j|^n + r^n \leq 1 + r^n \approx 1 + e.$$

Also,

$$|\sigma(z_j)| = |z^n - r^n| \geq |r^n| - |z^n| \leq r^n - 1 \approx e - 1.$$

Since we again have

$$\sigma^{(1)}(z) = \frac{d\sigma(z)}{dz} = nz^{n-1}$$

in this specialized case, and since $|a_j| = 1$, each

$$\sigma^{(1)}(a_j) = \sigma^{(1)}(r\omega^j) = n(r\omega^j)^{n-1},$$

so

$$|\sigma^{(1)}(a_j)| = nr^{n-1} \approx ne/r.$$

1. *Approximate computation of the $\sigma(z_j)$ via the unitary Trummer's problem.* If the input evaluation points are not fixed, we might exactly compute all the $\sigma(z_j) = z_j^n - r^n$ by powering in $O(m \log n)$ work. Instead, we will approximately compute the $\sigma(z_j)$, $j = 0, \ldots, m-1$. To do this, as described above, define a unitary Trummer's problem of size m where we specialize the weights to be 1, so $\psi^*(z) = 1/\sigma(z)$. Since for each j, the $a_j = r\omega^j$ are fixed, we can assume the $\sigma^{(1)}(a_j) = n(r\omega^j)^{n-1}$ are precomputed, thus providing the weights $c_j = 1/\sigma^{(1)}(a_j)$. Applying Lemma 2.2 again, the resulting unitary Trummer's problem of size n is ϵ-approx on every output $\widetilde{\sigma}(z_j)$, so $|\sigma(z_j) - \widetilde{\sigma}(z_j)| \leq \epsilon$, with the same work $O(m \log^2 k)$. We assume $\epsilon \leq 1/4$. By Proposition 2.5, each $\widetilde{\sigma}(z) = 1/\widetilde{\psi}^*(z_j)$ is an ϵ^*-approx to $\sigma(z_j)$, where

$$\epsilon^* = \max_j \frac{\epsilon|\sigma(z_j)|^2}{1 - |\sigma(z_j)|\epsilon} \leq \frac{\epsilon(1+e)^2}{1-(1+e)\epsilon} \leq O(\epsilon),$$

since we have shown $\max_j |\sigma(z_j)| \leq 1 + e$ and we have assumed $\epsilon \leq 1/4$. By Proposition 2.4, for $j = 0, \ldots, m-1$, each $\widetilde{P}(z_j)$ are an $\widehat{\epsilon}$-approx to $P(z_j)$, where

$$\widehat{\epsilon} \leq (\epsilon^* + \max_j |\sigma(z_j)|)\epsilon + (2 + (3\epsilon^*)/\min_j |\sigma(z_j)|)|P| \leq O(\epsilon|P|).$$

2. *Computation of the well-separated decomposition.* We can compute a well-separated decomposition of the set union of the $m \geq n$ evaluation points and the $a_j = r\omega^j$, $j = 0, \ldots, n-1$ within work $O((n+m)\log(n+m)) \leq O(m \log m)$ by the algorithm of Callahan and Kosaraju [6]. Alternatively, if the points have logarithmic bit-precision, then the algorithm of [30, 34] computes, within work $O((n+m)\log\log(n+m)) \leq O(m \log\log m)$, a well-separated decomposition the set of m evaluation points. Furthermore, since the $a_j = r\omega^j$, $j = 0, \ldots, n-1$ are regularly spaced on the radius r circle, a simple modification of the well-separated decomposition algorithm of [30, 33] can be used to compute a well-separated decomposition of the set union of n roots of unity and the set of m evaluation points, within work $O(m \log\log m)$.

In either case, we proceed as follows. We apply the above construction, for re-evaluation via Trummer's problem, to $P(z)$. Since $|\sigma^{(1)}(r\omega^j)| \approx ne/r$, the coefficients $c_j = P(\omega^j)/\sigma^{(1)}(\omega^j)$ of the Trummer's problem have summed magnitude $|C| \approx |P|r/e \leq O(|P|)$, so we may let

$$k = O(\log(|C|/\epsilon)) = O(\log(|P|/\epsilon)).$$

Again the resulting Trummer's problem of size m can be ϵ'-approx on every output using the efficient multipole algorithm of [30, 34] (Lemma 2.2) within work $O(m \log^2 k)$. For $j = 0, \ldots, m-1$, let

$$\widetilde{P}(z_j) = \widetilde{\psi}(z_j)\widetilde{\sigma}(z_j)$$

be the resulting approximation of $P(z_j)$. In either case, the error of approximation of $P(z_j)$ by $\widetilde{P}(z_j)$ is bounded as $|\widetilde{P}(z_j) - P(z_j)| \leq \widehat{\epsilon}$. Thus, by rescaling (to simplify notation) the total error $O(\widehat{\epsilon})$ to ϵ, we have the following theorem.

THEOREM 2.8. *Suppose we are given a complex degree $n-1$ polynomial $P(z)$, m evaluation points (which are not fixed) on or within the unit disk, and a given $\epsilon, 0 < \epsilon < 1/4$. Let $k = O(\log(|P|/\epsilon))$. Then we can compute an ϵ-approx of this complex*

polynomial evaluation problem within work $O(m(\log m + \log^2 k) + n \log n)$. *Thus, if* $m \geq (n \log n)/(\log m + \log^2 k)$, *then we require amortized work* $O(\log m + \log^2 k)$ *per evaluation point.*

Note. If the m evaluation points have logarithmic bit-precision, then the complexity bounds of Theorem 2.8 can be improved by application of the well-separated decomposition algorithm of [30, 33]; in this case each appearance of $\log m$ in Theorem 2.8 can be replaced with $\log \log m$.

Further note. In the special case that the m evaluation points are chirp points, a Trummer's problem for m chirp evaluation points can be ϵ-approx on every output within work $O(m \log k)$. Thus the work bounds given in Theorem 2.8 can in this case be reduced by replacing $\log^2 k$ by $\log k$.

3. Chebyshev points and approximate real polynomial evaluation.

3.1. The Chebyshev point evaluation problem. Let $\omega = e^{i2\pi/n'}$ be the n'th root of unity over the complex numbers $\mathcal{C}$. Note that

$$\omega^j = \cos(j2\pi/n') + i\sin(j2\pi/n').$$

Let $m \leq n'$. The (n', m)-*Chebyshev points* are the set of m real parts $x_j = \cos(j2\pi/n')$ of the powers ω^j, for $j = 0, 1, \ldots, m-1$. (This is slightly nonstandard notation, but convenient for our purposes.) Given real constants s_0, s_1 the *shifted* (n', m)-*Chebyshev points* are real values

$$x'_j = s_0 + s_1 \cos(j2\pi/n').$$

The *(shifted)* (n', m)-*Chebyshev point evaluation problem* is the multipoint real evaluation problem of evaluating a polynomial of degree $n-1$ at the set of m (shifted, respectively) (n', m)-Chebyshev points. The *(shifted)* (n', n)-*Chebyshev point interpolation problem* is the multipoint real interpolation problem of interpolating a polynomial of degree $n-1$ from the set of n (shifted, respectively) (n', n)-Chebyshev points.

Pan [29] has given an $O(n \log^2 n)$ algorithm for the $(4n, n)$-Chebyshev point evaluation problem and the shifted version of this problem. The results of Gerasoulis [18] imply an $O(n \log n)$ algorithm for (n', n)-Chebyshev point evaluation, for $n' = O(n)$, which uses a reduction to Trummer's problem for Chebyshev points. In the appendix we give an alternative algorithm (using a recursive algorithm and a reduction to the DFT and by application of Proposition 1.1) for (n', n)-Chebyshev point evaluation (and also interpolation) with these same work bounds but yielding a simpler algorithm.

LEMMA 3.1. *For* $n' \leq O(n)$, *the* (n', m)-*Chebyshev point evaluation problem can be solved in work* $O((n+m) \log \min(n, m))$, *and the* (n', n)-*Chebyshev point interpolation problem can be solved in work* $O(n \log n)$. *(The circuit depth is* $O(\log n)$.*) Also, the shifted version of these problems can be solved within the same work bounds.*

3.2. Approximate real evaluation via interpolation at the Chebyshev points. Interpolation at the Chebyshev points is a well-known classical method for approximation of real functions (see, for example, Dahlquist and Björck [10] and Henry [24]). Fix an interval $I = [L, U]$ of the real line of length $|I| = U - L$. Let $f(x)$ be any real function over I. We say $\widetilde{f}(x)$ is an ϵ-approx of $f(x)$ over I if $|f(x) - \widetilde{f}(x)| \leq \epsilon$ for any $x \in I$. Fix $\widetilde{f}(x)$ to be the degree k polynomial derived by interpolating $f(x)$ at the shifted $(2k, k)$-Chebyshev points $x'_j = \frac{U+L}{2} + \frac{U-L}{2}\cos(\frac{j\pi}{k})$, for $j = 0, \ldots, k-1$,

which are on the interval $I = [L, U]$. Thus $\widetilde{f}(x)$ is the unique degree k polynomial such that $\widetilde{f}(x'_j) = f(x'_j)$ for $j = 0, \ldots, k-1$. Let

$$f^{(k)}(x) = \frac{d^k f(x)}{d^k x}$$

be the kth derivative of $f(x)$ with respect to x.

PROPOSITION 3.2 (see Dahlquist and Björck [10]). *For any $x \in I$, $\widetilde{f}(x)$ is an ϵ-approx of $f(x)$, for*

$$\epsilon = \frac{2(|I|/4)^k}{k!} \max_{y \in I} |f^{(k)}(y)|.$$

3.3. Approximate real polynomial evaluation via interpolation at the Chebyshev points. Rokhlin [35] applied Proposition 3.2 for a fast algorithm for the discrete Laplace transformation. Fix $P(x)$ to be a real polynomial of degree $n-1$. As observed in Pan, Reif, and Tate [30], direct application of the well-known Proposition 3.2 immediately gives an ϵ-approx to the real multipoint evaluation problem for $P(x)$ at any given m real points $x_0, \ldots, x_{m-1}$. By Lemma 3.1, the shifted $(2k, k)$-Chebyshev point evaluation problem for $P(x)$ can be solved within work $O(n \log k)$. Known exact algorithms [16, 26, 5] for k-point polynomial interpolation require work $O(k \log^2 k)$. By Proposition 1.2, the multipoint polynomial evaluation problem for $\widetilde{P}(x)$ at the real points $x_0, \ldots, x_{m-1} \in I$ can be solved within work $O(m \log^2 k)$. Thus an ϵ-approx of the multipoint polynomial evaluation problem for $P(x)$ at any m real points $\in I$ can be solved within work $O(m \log^2 k)$, where $\epsilon = \frac{2(|I|/4)^k}{k!} \max_{y \in I} |P^{(k)}(y)|$. Let $\beta = \log(|P|/\epsilon)$, where $|P|$ is the sum of the moduli of the coefficients of P. Assuming without loss of generality (w.l.o.g.) $k = o(n)$, note that if we fix $I = [-1, 1]$ we have that

$$\max_{y \in I} |P^{(k)}(y)| \leq \max_{y \in I} \max_{k \leq j < n} \frac{j!}{(j-k)!} y^{-(j-k)} |P| \leq \max_{k \leq j < n} \frac{j!}{(j-k)!} 2^{-(j-k)} |P|,$$

which is maximized when $j = k$, so

$$\max_{y \in I} |P^{(k)}(y)| \leq \frac{j!}{(j-k)!} 2^{-(j-k)} |P| \leq k! |P|.$$

Setting the degree of $\widetilde{P}$ to be $k-1$, where $k = \log(|P|/\epsilon)$, we have that

$$\epsilon \leq \frac{2(|I|/4)^k}{k!} \max_{y \in I} |P^{(k)}(y)| \leq |P|/2^{k-1},$$

implying, by Proposition 3.2, the following known result.

LEMMA 3.3 (see Pan, Reif, and Tate [30] and also Bini and Pan [3]). *An ϵ-approx of the multipoint polynomial evaluation problem for a degree $n-1$ polynomial $P(x)$ at any $m \geq n$ real points $\in [-1, 1]$ can be solved within work $O(m \log^2 \min(n, k))$, where $k = \log(|P|/\epsilon)$.*

So far we have assumed that $P(x)$ has only real coefficients. Note that if $P(x)$ has complex coefficients, then we can let $P(x) = P_0(x) + iP_1(x)$ where $P_0(x), P_1(x)$ have only real coefficients. Then application of Lemma 3.3 for each of the $P_0(x), P_1(x)$

implies that for $k = \log(|P|/\epsilon)$, the resulting error of approximation for evaluation of $P(x)$ is a $\sqrt{2}$ factor more, that is, $\sqrt{2}\epsilon$. Hence rescaling (to simplify notation) the error $\sqrt{2}\epsilon$ to ϵ, we have the following lemma.

LEMMA 3.4. *An ϵ-approx of the multipoint polynomial evaluation problem for a degree $n-1$ polynomial $P(x)$ with complex coefficients at any $m \geq n$ real points $\in [-1,1]$ is solved in work*

$$\leq O(m \log^2 \min(n, \log(|P|/\epsilon))).$$

By Lemma 3.1, the (n', m)-Chebyshev point evaluation problem for the degree $k-1$ polynomial $\widetilde{P}(x)$ at shifted (n', m)-Chebyshev points can be solved within work $O(m \log k)$. All Chebyshev points are within the interval $I = [-1, 1]$. Setting $k = \log(|P|/\epsilon)$, we have again that $\epsilon \leq |P|/2^{k-1}$ implying the improved result.

LEMMA 3.5. *An ϵ-approx of the (n', m)-Chebyshev point evaluation problem for a degree $n-1$ polynomial $P(x)$ is solved in work*

$$O((m+n) \log \min(n, \log(|P|/\epsilon))).$$

(The parallel time is $O(\log \min(n, \log(|P|/\epsilon)))$.)

4. Approximate evaluation on a circle.

4.1. Approximate polynomial evaluation on a circle via interpolation at the Chebyshev angles. We further reduce our amortized work bounds for special sets of evaluation points. By reduction to approximate evaluation of Trummer's problem via the Multipole algorithm, we have already shown how to ϵ-approx complex degree $n-1$ polynomial evaluation at $m \geq n \log n$ chirp points $\zeta^j, j = 0, \ldots, m-1$, for some fixed complex number ζ, $|\zeta| \leq 1$ in amortized work $O(\log k)$ per point. Using quite distinct techniques, we give in this section a reduction from ϵ-approx complex degree $n-1$ polynomial evaluation at $m \geq n$ points on a circle of radius r to approximate real polynomial evaluation, again in amortized work $O(\log^2 k)$ per point. Let $i = \sqrt{-1}$. Fix a complex polynomial $P(z) = \sum_{j=0}^{n-1} p_j z^j$ of degree $n-1$ and let

$$|P| = \sum_{j=0}^{n-1} |p_j|.$$

Let $\epsilon > 0$ be a given error bound. Let $z = re^{i\theta}$ range on a circle of radius r, and define the angle of z to be $\theta(z) = \theta$. We now consider the evaluation of $P(z)$ at a set of points $z_j = re^{i\theta_j}$ for $j = 1, \ldots, m$ over an interval of a circle of radius r with angular bounds $\theta_j \in [0, \Delta]$, for a positive real $\Delta \leq 2\pi$. The set of points $re^{i\theta_j}$, for $j = 0, \ldots, m-1$, are *regularly spaced* on the circle of radius r if the θ_j are a linear function of j (for example, the n roots of unity are regularly spaced on the unit circle). The polynomial $P(z)$ is (k, t)-*descending* if the magnitude of the coefficients drops as

$$|p_j| \leq n^{O(1)} |P| \left(\frac{t}{j}\right)^k,$$

for $j = k, \ldots, n-1$. Here we prove the following.

THEOREM 4.1. *Suppose we are given a complex polynomial $P(z) = \sum_{j=0}^{n-1} p_j z^j$ of degree $n-1$, and a fixed set of evaluation points $z_j = re^{i\theta_j}$ for $j = 0, \ldots, m-1$ on a circle of radius r with angular range $\theta_j \in [0, \Delta]$. Let $c = 1$ if the evaluation points are*

regularly spaced and otherwise $c = 2$. *Then an* ϵ*-approx of this multipoint polynomial evaluation problem can be computed within work* $O(m \log^c k)$ *if either* 1. $r = 1$ *and* $\Delta \leq \frac{k}{e(n-1)}$, *or* 2. $r = 1$ *and* $P(z)$ *is* $(k, k/(e\pi))$*-descending, or* 3. $r \leq 1/(e\pi)$. *Here* $k = \frac{1}{2}\log(|P|/\epsilon) + O(1)$ *in cases* 1 *and* 3 *and* $k = O(\log(n|P|/\epsilon))$ *in case* 2.

Proof. Restrict z to the upper half circle of radius r, so $z = re^{i\theta}$, where $i = \sqrt{-1}$, and $0 \leq \theta \leq \pi$. Let $Q(\theta) = P(re^{i\theta}) = \sum_{j=0}^{n-1} q_j e^{ij\theta}$, where $q_j = r^j p_j$, and $P(z) = \sum_{j=0}^{n-1} p_j z^j$. Fix a number k which divides n, to be determined below. The exact evaluation of $P(z)$ at the evaluation points z_j, for $j = 0, \ldots, m-1$, can be done by an exact evaluation of $Q(\theta)$ at the real points $\theta_0, \ldots, \theta_{m-1}$, where $z_j = re^{i\theta_j}$. Instead, we will do an ϵ-approximate evaluation of $P(z)$ at the evaluation points $z_0, \ldots, z_{m-1}$, as follows. We can assume w.l.o.g. that each $0 \leq \theta_j \leq \pi$. Since $P(-z) = \sum_{j=0}^{n-1} p_j(-z)^j = \sum_{j=0}^{n-1} p'_j z^j$, where $p'_j = -p_j$ if j is odd, and else $p'_j = p_j$, the case where the evaluation points $z_j = re^{i\theta_j}$ are on the lower half circle can be reduced to this case by multiplication of the z_j by $e^{i\pi} = -1$, and switching the sign of the coefficients p_j of $P(z)$ where j is odd. We construct a degree $k-1$ polynomial $\widetilde{Q}(\theta)$ which gives an ϵ-approx of polynomial $Q(\theta)$. To construct $\widetilde{Q}(\theta)$, we do an exact evaluation of $Q(\theta)$ at the shifted $(2k, k)$-Chebyshev points $\theta'_j = \pi\cos(j\pi/k)$ for $j = 0, \ldots, k-1$ over the real interval $I = [0, \Delta]$, for a given $\Delta, 0 < \Delta \leq 2\pi$. To do this first step, we can exactly evaluate $P(z)$ at $re^{i\theta'_j}$ for $j = 0, \ldots, k-1$, which in general costs work $O(n \log^2 k)$ by Proposition 1.1; or if the evaluation points are regularly spaced, then by Proposition 1.4 this costs $O(n \log k)$. We could now exactly interpolate a degree $k-1$ polynomial $\widetilde{Q}(\theta)$ from $Q(\theta'_j) = P(re^{i\theta'_j})$, for $j = 0, \ldots, k-1$, which are the values of $Q(\theta)$ at these shifted $(2k, k)$-Chebyshev points $\theta'_0, \theta'_1, \ldots, \theta'_{k-1}$. By the known algorithms of [16, 26, 5], the exact interpolation of polynomial $\widetilde{Q}(\theta)$ at these k points cost work $O(k \log^2 k)$. Also, by known algorithms (see Proposition 1.2), we can exactly evaluate $\widetilde{Q}(\theta_j)$ for $j = 0, \ldots, m-1$, in work $O(m \log^2 k)$, and if these evaluation points are regularly spaced, then this costs work $O(m \log k)$ by the chirp transform (Proposition 1.4).

LEMMA 4.2. *For* $k = o(n)$, *and* $0 \leq \theta \leq \Delta$, *then*

$$|Q^{(k)}(\theta)| = \left|\frac{d^k Q(\theta)}{d^k \theta}\right|$$

is bounded by $O(k!|P|\Delta^{-k})$ *if either* 1. $r = 1$ *and* $\Delta \leq \frac{k}{e(n-1)} \leq \pi$, *or* $\Delta = \pi$ *and either* 2. $r = 1$ *and* $P(z)$ *is* $(k, k/(e\pi))$*-descending, or* 3. $r \leq 1/(e\pi)$.

Proof.

$$\frac{d^k e^{ij\theta}}{d^k \theta} = \frac{j!}{(j-k)!}(ij)^k e^{ij\theta},$$

so

$$Q^{(k)}(\theta) = \frac{d^k Q(\theta)}{d^k \theta} = \sum_{j=k}^{n-1} q_j \frac{j!}{(j-k)!}(ij)^k e^{ij\theta}.$$

Since $|i| = |e^i| = 1$, and $|q_j| = |p_j| r^j$, we have

$$|Q^{(k)}(\theta)| = \sum_{j=k}^{n-1} |q_j| j^k = \sum_{j=k}^{n-1} |p_j| r^j j^k.$$

1. Now suppose $r = 1$ and $\Delta \leq \frac{k}{e(n-1)}$. Then by the Stirling approximation to factorial, $k! \geq (k/e)^k$ and so

$$j^k \leq (k/e)^k \Delta^{-k} \leq k! \Delta^{-k}.$$

Hence we have

$$|Q^{(k)}(\theta)| \leq \sum_{j=k}^{n-1} |p_j| j^k \leq O(|P| k! \Delta^{-k}).$$

2. Next, suppose $r = 1$, and $\Delta = \pi$, and $P(z)$ is $(k, k/(e\pi))$-descending; so by definition, the magnitude of the coefficients of $P(z)$ drops as

$$|p_j| \leq n^{O(1)} |P| (\frac{k}{je\pi})^k,$$

for $j = k, \ldots, n-1$. Then

$$|p_j| r^j j^k \leq n^{O(1)} |P| (k/(e\pi))^k \leq O(n^{O(1)} |P| k! \Delta^{-k}).$$

Thus,

$$|Q^{(k)}(\theta)| \leq \sum_{j=k}^{n-1} |p_j| r^j j^k \leq \sum_{j=k}^{n-1} O(n^{O(1)} |P| k! \Delta^{-k}) \leq O(n^{O(1)} |P| k! \Delta^{-k}).$$

3. Finally, suppose $r \leq 1/(e\pi)$ and $\Delta = \pi$. Since $|Q^{(k)}(\theta)|$ increases with r, to upper bound $|Q^{(k)}(\theta)|$ we can assume w.l.o.g. that r is at its maximum value $r = 1/(e\pi)$. By taking derivatives, it is easy to verify that $1/(e\pi) \leq (se\pi)^{-1/s}$ for any $s \geq 1$. So

$$1/(e\pi) = \min_{s \geq 1} (se\pi)^{-1/s} = \min_{j \leq k} \left(\frac{k}{je\pi} \right)^{k/j}$$

(by the substitution $j = sk$). Hence for each $r \leq 1/(e\pi)$, we can bound $r^j \leq (\frac{k}{je\pi})^k$ and so for all $j \geq k$ we have

$$r^j j^k \leq (k/(e\pi))^k \leq k! \Delta^{-k},$$

since $\Delta = \pi$ and $k! \geq (k/e)^k$. Thus,

$$|Q^{(k)}(\theta)| \leq \sum_{j=k}^{n-1} |p_j| r^j j^k \leq \sum_{j=k}^{n-1} |p_j| k! \Delta^{-k} \leq O(|P| k! \Delta^{-k}). \qquad \square$$

To complete the proof of Theorem 4.1, we need to bound the approximation errors for evaluation of $\widetilde{Q}(\theta)$ at all the given evaluation points; this is implied by the following proof of ϵ-approximate evaluation of $P(z)$. Since $|I| = \Delta$, then by Lemma 4.2 and Proposition 3.2 the error in the approximation of $P(z) = Q(\theta)$ by $\widetilde{Q}(\theta)$, for $\theta \in I = [0, \Delta]$ is upper bounded by

$$\frac{2}{k!} (|I|/4)^k \max_{\theta \in I} |Q^{(k)}(\theta)| \leq O(|P| (\Delta/4)^k \Delta^{-k}) \leq O(|P| 4^{-k})$$

in cases 1 and 3, and in case 2, the error is upper bounded by $O(n^{O(1)}|P|4^{-k})$. Hence the error is upper bounded by $O(\epsilon)$ if in cases 1 and 3 we set $k = \frac{1}{2}\log(|P|/\epsilon) + O(1)$ and in case 2 we set $k = O(\log(n|P|/\epsilon))$. Finally, note that the (empty) restriction $0 \le \theta \le \pi = \Delta$ in cases 2 and 3 allows z to range over any point on the upper half circle of fixed radius r, so we have the following lemma.

LEMMA 4.3. *$\widetilde{Q}(\theta)$ is an ϵ-approx of $P(z)$ at any point $z = re^{i\theta}$ on the upper half circle of fixed radius r, where $0 \le \theta \le \Delta$, if either* 1. *$r = 1$ and $\Delta \le \frac{k}{e(n-1)} \le \pi$, or* 2. *$r = 1$ and $P(z)$ is $(k, k/(e\pi))$-descending, or* 3. *$r \le 1/(e\pi)$.*

Thus, we have proven Theorem 4.1. □

Cases 1 and 2 of Theorem 4.1 imply the following corollary.

COROLLARY 4.4. *In total work $O(n \log k)$ we can*

1. *ϵ-approx the evaluation of a degree n polynomial at the first n powers of the n'th root of unity, where $n' \ge \Omega(n^2/k)$, and*
2. *ϵ-approx the n-point DFT in total work $O(n \log k)$ for inputs with $(k, k/(e\pi))$-descending coefficient magnitude.*

Note. If the input set of evaluation points are not fixed, the result as given in Theorem 4.1 holds with the same work bounds, assuming an *extended arithmetic model* where, given a real θ, then $e^{i\theta} = \cos(\theta) + i\sin(\theta)$ can be computed in $O(1)$ steps. This assumption is not required if the points are fixed, as assumed in Theorem 4.1, or regularly spaced (as in Corollary 4.4). Also this assumption is not required in the alternative construction given in subsection 4.2, even if the input set of evaluation points are not fixed.

4.2. An alternative approach to approximate evaluation on a circle via approximate real polynomial evaluation. We now give an alternative construction for a similar, but slightly weaker, result as Theorem 4.1, by use of a surprisingly simple algorithm (the proof is somewhat more involved, however), which uses complex polynomials rather than real polynomials.

Input: The coefficients of degree $n-1$ complex polynomial $P(z)$, and $m \ge n$ evaluation points $z_j = re^{i\theta_j}$, for $j = 0, \ldots, m-1$ on an interval of a circle of radius $r = 1$ with $|\cos(\theta_j)| \le \Delta$. We assume either 1. $\Delta \le \frac{k}{e(n-k)}$, or 2. $P(z)$ is $(k, \sqrt{2}/8)$-descending. In case 1. let k be the smallest power of 2 which is $\ge \log(\sqrt{2}|P|/\epsilon)$, and in case 2. let k be the smallest power of 2 which is $O(\log(n|P|/\epsilon))$. In any case we assume $k = o(n)$.

[0] Partition the set of evaluation points $\{z_j | j = 1, \ldots, m-1\}$ into 4 sets

$$\{z_{j,h} | j = 0, \ldots, m_h - 1\}, \qquad h = 0, \ldots, 3,$$

such that $|i^h\theta_{j,h} - \pi/2| \le \pi/4$ for each $j = 0, \ldots, m_h$.

Comment: Multiplication by $i = e^{i\pi/2}$ shifts the angle by $\pi/2$.

For $h = 0, \ldots, 3$ **do**

1. Define polynomial $P_h(z) = P(i^h z)$. Evaluate $P_h(z)$ at

$$z'_j = r(x'_j + i\sqrt{1 - (x'_j)^2}),$$

where

$$x'_j = \cos(j\pi/k)/\sqrt{2},$$

for $j = 0, \ldots, k-1$, in work $O(n \log^2 k)$ (by Proposition 1.1).

2. Interpolate the degree $k-1$ complex polynomial $\widetilde{P}_h(z)$ from the values $P_h(z'_j)$, for $j = 0, \ldots, k-1$, in work $O(k \log^2 k)$.

3. Evaluate polynomial $\widetilde{P}_h(z)$ at the given evaluation points $z_{j,h}$ for $j = 0, \ldots, m_h - 1$. If these evaluation points are regularly spaced, then this costs work $O(m_h \log k)$ by the chirp transform (Proposition 1.4) and otherwise costs work $O(m_h \log^2 k)$ by Proposition 1.1. This gives $\widetilde{P}_h(z_j)$ which ϵ-approx $P_h(z_j)$, for $j = 0, \ldots, m_h - 1$.

Output: $\widetilde{P}(z_j)$ which ϵ-approx $P(z_j)$, for $j = 0, \ldots, m-1$.

We prove the following in Appendix A.

THEOREM 4.5. *$\widetilde{P}(z)$ is an ϵ-approx of $P(z)$ at any point $z = r(x+iy)$ on the unit circle of fixed radius $r = 1$ where $|\theta(z) - \pi/2| \leq \pi/4$ and $|x| \leq \Delta$ if either* 1. $\Delta \leq \frac{k}{e(n-k)}$ *or* 2. *$P(z)$ is $(k, \sqrt{2}/8)$-descending.*

The work is again $O(n \log^2 k) + O(m \log^c k) \leq O(n \log^2 k + m \log^c k)$. Also, the circuit depth can be seen to be $O(\log^c k)$ with this same work bound.

4.3. Dutt and Rokhlin's approximation of a polynomial on the unit circle. A further method for approximation of a polynomial over an interval of the unit circle is proposed by Dutt and Rokhlin [14], who give an approximation to a polynomial $P(z)$, as follows. Fix a real constant α, $0 < \alpha < \pi/3$. They define a mapping τ from complex numbers $z = e^{i\theta}$ on the unit circle to real $x = \tau(z) = 3 \tan\alpha \cot(\theta/2)$ and use this mapping to define the real function $f(x) = P(z)$. They use a degree $k-1$ polynomial $\widetilde{f}(x)$ to approximate $P(z)$ at angular positions θ, $6\alpha \leq \theta \leq 2\pi - 6\alpha$. The polynomial $\widetilde{f}(x)$ interpolates $f(x)$ at the shifted $(2k, k)$-Chebyshev points $a_j = -\cos((j - 1/2)\pi/k)$ and is defined, for $|x| \leq 1$, as $\widetilde{f}(x) = \sum_{j=1}^{k-1} f(a_j) \prod_{\ell \neq j} (\frac{x - a_\ell}{a_j - a_\ell})$. Note that $\widetilde{f}(a_j) = f(a_j)$ for $j = 1, \ldots, k-1$. Dutt and Rokhlin [14] prove that $\widetilde{f}(x)$ approximates $f_h(x) = P(z)$, with relative error $O(1/5^k)$ over some portion of this interval. Our Theorem 5.1 implies that the length of the interval where these error bounds can be obtained must be very small.

5. Approximation everywhere on the unit circle is not possible. Here we show that it is not possible to approximate an arbitrary polynomial by a small degree polynomial over a large portion of the unit circle. Fix an angular interval $[0, \Delta]$ for $0 \leq \Delta \leq 2\pi$ and an error ϵ, $0 \leq \epsilon < 1$. Let us define an (k, Δ) *circle ϵ-approx evaluation scheme* for a degree $n-1$ polynomial $P(z)$ to be a degree $k-1$ polynomial $\widetilde{f}(x)$ which ϵ-approx $P(z) = f(x)$ at all angular positions $\theta \in [0, \Delta]$, where τ is a mapping from complex numbers $z = e^{i\theta}$ on the interval of the unit circle, $\theta \in [0, \Delta]$ to real $x = \tau(z)$. We now prove that any (k, Δ) circle ϵ-approx evaluation scheme in general only can be convergent for angular positions $\theta \in [0, \Delta]$, where $\Delta \leq O(k/n)$.

THEOREM 5.1. *There is no (k, Δ) circle ϵ-approx evaluation scheme for every degree $n-1$ polynomial $P(z)$, where $k\lceil 2\pi/\Delta \rceil < n$ and $\epsilon(\epsilon + 2|P|) < 1$.*

Proof. Let $H = \lceil 2\pi/\Delta \rceil$. We can use the (k, Δ) circle ϵ-approx evaluation scheme to define for each $h, 0 \leq h < H$ a degree $k-1$ polynomial $f_h(x)$ that ϵ-approx $P(z)$ for $x = \tau(z\omega^{-2\pi h/H})$. For a complex number z, define $h(z)$ to be the integer h such that $2\pi h/H \leq |\text{angle}(z)| < 2\pi(h+1)/H$.

PROPOSITION 5.2. *Given a complex degree $n-1$ polynomial $P(z)$, and a (k, Δ) circle ϵ-approx evaluation scheme, we can construct $H = \lceil 2\pi/\Delta \rceil$ polynomials $(f_0(x), \ldots, f_{H-1}(x))$=POLY-APPROX$_k(P)$, each of degree $k-1$, such that for any z on the unit circle, $f_h(x)$ is an ϵ-approx of $P(z)$, for $h = h(z)$ and $x = \tau(z\omega^{-2\pi h/H})$.*

Polynomial Convolution is defined as follows:

Input: Complex coefficients $p_0, \ldots, p_{n-1}, q_0, \ldots, q_{n-1}$, defining degree $n-1$ polyno-

mials $P(z) = \sum_{j=0}^{n-1} p_j x^j$ and $Q(z) = \sum_{j=0}^{n-1} q_j x^j$.
Output: Complex coefficients $q_0, \ldots, q_{2n-2}$, defining degree $2n-2$ product polynomial $R(z) = P(z)Q(z) = \sum_{\ell=0}^{2n-2} r_\ell x^\ell$, where

$$r_\ell = \sum_{j=0, 0 \le \ell - j < n}^{n-1} p_j q_{\ell - j}.$$

Next we show the following.

LEMMA 5.3. *Suppose there is a (k, Δ) circle ϵ-approx evaluation scheme. Then, given complex n-vectors $\mathbf{u}$, $\mathbf{v}$, we can construct vectors $\widetilde{\mathbf{u}} = ROW\text{-}COMPRESS_{\,k}(\mathbf{u})$, $\widetilde{\mathbf{v}}^T = COLUMN\text{-}COMPRESS_k(\mathbf{v}^T)$ of size $O(k)$ (where $\widetilde{\mathbf{u}}$ depends only on $\mathbf{u}$ and not on $\mathbf{v}$, and furthermore $\widetilde{\mathbf{v}}$ depends only on $\mathbf{v}$ and not on $\mathbf{u}$) such that $\widetilde{\mathbf{u}}^T \widetilde{\mathbf{v}}$ is an ϵ_1-approx of $\mathbf{u}^T\mathbf{v}$ for $\epsilon_1 \le \epsilon(\epsilon + |\mathbf{u}| + |\mathbf{v}|)$.*

Proof. Given complex n-vectors $\mathbf{u} = (u_0, \ldots, u_{n-1})^T$, $\mathbf{v} = (v_0, \ldots, v_{n-1})^T$, we can compute the inner product

$$\mathbf{u}^T\mathbf{v} = \sum_{j=0}^{n-1} u_j v_j$$

by computing the nth coefficient

$$r_n = \sum_{j=0}^{n-1} p_j q_{n-j} = \sum_{j=0}^{n-1} u_j v_j = \mathbf{u}^T\mathbf{v}$$

of the product polynomial $R(z) = P(z)Q(z)$ as defined above and where $p_j = u_j$ and $q_j = v_{n-j}$, for $j = 0, \ldots, n-1$. Let $\omega = e^{i\pi/n}$ be the $(2n)$th root of unity, where $i = \sqrt{-1}$. By the convolution theorem, $\mathbf{u}^T\mathbf{v} = r_n = \frac{1}{2n} \sum_{j=0}^{2n-1} \omega^{-nj} P(\omega^j) Q(\omega^j)$. For each of $P(z), Q(z)$ defined above, by Proposition 5.2 we can construct $H = \lceil 2\pi/\Delta \rceil$ degree $k-1$ polynomials $(f_0(x), \ldots, f_{H-1}(x)) =$ POLY-APPROX$_k(P)$, and $(g_0(x), \ldots, g_{H-1}(x)) =$ POLY-APPROX$_k(Q)$, such that for any z on the unit circle, $\widetilde{P}_h(z) = f_h(x)$ is an ϵ-approx of $P(z)$, and $\widetilde{Q}_h(z) = g_h(x)$ is an ϵ-approx of $Q(z)$, for $h = h(z)$ and $x = \tau(z\omega^{-2\pi h/H})$. For $h = 0, \ldots, H-1$ let $f_h(x) = \sum_{a=0}^{k-1} f_{h,a} x^a$ and $g_h(x) = \sum_{b=0}^{k-1} g_{h,b} x^b$. Let $\delta = n/H$. For each integer $j, 0 \le j < 2n$, let h_j be the number $h \in \{0, \ldots, H-1\}$ such that $\delta h \le j < \delta(h+1)$. Then we have that $\widetilde{P}_{h_j}(\omega^j)$ is an ϵ-approx of $P(\omega^j)$, and $\widetilde{Q}_{h_j}(\omega^j)$ is an ϵ-approx of $Q(\omega^j)$. We will approximate $r_n = \frac{1}{2n} \sum_{j=0}^{2n-1} \omega^{-nj} P(\omega^j) Q(\omega^j)$ by $\widetilde{r}_n = \frac{1}{2n} \sum_{j=0}^{2n-1} \omega^{-nj}\ \widetilde{P}_{h_j}(\omega^j) \widetilde{Q}_{h_j}(\omega^j)$.

PROPOSITION 5.4. $|r_n - \widetilde{r}_n| \le \epsilon(\epsilon + |P| + |Q|)$.

Proof. Since $|\omega^{-nj}| = 1$ and for $h = h_j$,

$$|P(\omega^j)Q(\omega^j) - \widetilde{P}_h(\omega^j)\widetilde{Q}_h(\omega^j)| \le |P(\omega^j)(Q(\omega^j) - \widetilde{Q}_h(\omega^j)) + (P(\omega^j) - \widetilde{P}_h(\omega^j))\widetilde{Q}_h(\omega^j)|$$

$$\le |P(\omega^j)||Q(\omega^j) - \widetilde{Q}_h(\omega^j)| + |P(\omega^j) - \widetilde{P}_h(\omega^j)||\widetilde{Q}_h(\omega^j)| \le \epsilon(|P| + \epsilon + |Q|).$$

Thus we have

$$|r_n - \widetilde{r}_n| \le \frac{1}{2n} \sum_{j=0}^{2n-1} |\omega^{-nj}||P(\omega^j)Q(\omega^j) - \widetilde{P}_{h_j}(\omega^j)\widetilde{Q}_{h_j}(\omega^j)|$$

$$\leq \frac{2n}{2n}\epsilon(\epsilon + |P| + |Q|) = \epsilon(\epsilon + |P| + |Q|). \qquad \square$$

Surprisingly, we can contract the expansion of $\widetilde{r}_n$ as follows in this proposition.

PROPOSITION 5.5. *$\widetilde{r}_n = \sum_{h=0}^{H-1} \sum_{a=0}^{k-1} f_{h,a} s_{h,a}$, where $s_{h,a} = \sum_{b=0}^{k-1} g_{h,b} c_{h,a+b}$, and the $c_{h,\ell}$ are complex scalar constants dependent only on h, ℓ, n, and δ.*

Proof. The $(2n)$th root of unity is defined $\omega = e^{i\pi/n}$. For a fixed small constant α, each power ω^j, for $0 \leq j < n$, is mapped to the real $d_j = \tau(\omega^j)$ which is a constant (that is, the d_j need not be computed, and instead are provided by the algebraic circuit that we construct) for fixed n and α. For $h = h_j$, we have defined $\widetilde{P}_h(\omega^j) = f_h(d_{j-\delta h}) = \sum_{a=0}^{k-1} f_{h,a} d_{j-\delta h}^a$ and similarly,

$$\widetilde{Q}_h(\omega^j) = g_h(d_{j-\delta h}) = \sum_{b=0}^{k-1} g_{h,b} d_{j-\delta h}^b.$$

Thus for $h = h_j$, we can expand

$$\widetilde{P}_h(\omega^j)\widetilde{Q}_h(\omega^j) = \left(\sum_{a=0}^{k-1} f_{h,a} d_{j-\delta h}^a\right)\left(\sum_{b=0}^{k-1} g_{h,b} d_{j-\delta h}^b\right) = \sum_{a,b=0}^{k-1} f_{h,a} g_{h,b} d_{j-\delta h}^{a+b}.$$

Interchanging the order of summation (bringing the summation of j inside and a summation of h outside), we get

$$\widetilde{r}_n = \frac{1}{2n}\sum_{j=0}^{2n-1} \omega^{-nj} \widetilde{P}_{h_j}(\omega^j)\widetilde{Q}_{h_j}(\omega^j) = \sum_{h=0}^{H-1}\sum_{a=0}^{k-1} f_{h,a} s_{h,a},$$

where we define the *associated prefix sums* to be

$$\begin{aligned} s_{h,a} &= \frac{1}{2n}\sum_{b=0}^{k-1} g_{h,b} \sum_{j=\delta h}^{\delta(h+1)-1} \omega^{-nj} d_{j-\delta h}^{a+b} \\ &= \sum_{b=0}^{k-1} g_{h,b}\left(\frac{1}{2n}\sum_{j=\delta h}^{\delta(h+1)-1} \omega^{-nj} d_{j-\delta h}^{a+b}\right) = \sum_{b=0}^{k-1} g_{h,b} c_{h,a+b}, \end{aligned}$$

and where we define the *associated scalar constants* to be

$$c_{h,\ell} = \frac{1}{2n}\sum_{j=\delta h}^{\delta(h+1)-1} \omega^{-nj} d_{j-\delta h}^{a+b}.$$

Thus, Proposition 5.5 follows. $\square$

Note that the $s_{h,a}$ are defined independently of the $f_{h,a}$. To complete the proof of Lemma 5.3, we define $\widetilde{\mathbf{u}} =$ ROW-COMPRESS$_k(\mathbf{u})$ and $\widetilde{\mathbf{v}}^T =$ COLUMN-COMPRESS$_k(\mathbf{v}^T)$, where for each $h = 0, \ldots, H-1$ and for each $a = 0, \ldots, k-1$,

$$\widetilde{\mathbf{u}}_{hk+a} = f_{h,a}$$

and

$$\widetilde{\mathbf{v}}_{hk+a} = s_{h,a}.$$

Thus by the convolution theorem and Proposition 5.5,

$$\widetilde{r}_n = \sum_{h=0}^{H-1}\sum_{a=0}^{k-1} f_{h,a}s_{h,a} = \sum_{\ell=0}^{Hk-1} \widetilde{\mathbf{u}}_\ell \widetilde{\mathbf{v}}_\ell = \widetilde{\mathbf{u}}^T\widetilde{\mathbf{v}}.$$

Note that

$$|\mathbf{u}| = \sum |u_j| = \sum |p_j| = |P|$$

and

$$|\mathbf{v}| = \sum |v_j| = \sum |q_j| = |P|.$$

Now, given an error bound $\epsilon_1 > 0$, let

$$\epsilon = \epsilon_1/(\epsilon + |P| + |Q|) = \epsilon_1/(\epsilon + |\mathbf{u}| + |\mathbf{v}|).$$

By Proposition 5.4, approximation error $\epsilon_1 \le \epsilon(\epsilon + |\mathbf{u}| + |\mathbf{v}|)$, so we have proved Lemma 5.3. ∎

Given two $n \times n$ matrices A, B (with rows and columns indexed from 0 to $n-1$), we now approximate the inner product AB, where $(AB)_{\ell,m} = \sum_{j=0}^{n-1} A_{\ell,j}B_{j,m}$; this is the inner product of the ℓth row of A times the mth column of B. Now again fix an error bound $\epsilon_1 > 0$. Let

$$k = O(\log((\|A\|_\infty + \|B^T\|_\infty)/\epsilon_1),$$

where $\|A\|_\infty$ is the maximum, for any row of A, of the sum of the moduli of elements of the row, and $\|B^T\|_\infty$ is the maximum, for any column of B, of the sum of the moduli of elements of the column. We can precompute approximation polynomials of degree $k-1$ and their associated prefix sums for each of the rows of A and each of the columns of B. That is, we define an $n \times (Hk)$ matrix $\widetilde{A} =$ ROWS-COMPRESS$_k(A)$ and an $(Hk) \times n$ matrix $\widetilde{B} =$ COLUMNS-COMPRESS$_k(B)$ such that for each $\ell = 0, \ldots, n-1$ we define row $\widetilde{A}_\ell =$ ROW-COMPRESS$_k(A_\ell)$ (where $A_\ell, \widetilde{A}_\ell$ denote the ℓth row vectors of matrices $A, \widetilde{A}$, respectively) and for each $m = 0, \ldots, n-1$ we define column $\widetilde{B}_{-,m} =$ COLUMN-COMPRESS$_k(B_{-,m})$ (where $B_{-,m}, \widetilde{B}_{-,m}$ denote the mth columns of matrices $B, \widetilde{B}$, respectively). Then by Lemma 5.3, $\widetilde{A}_\ell\widetilde{B}_{-,m}$ is an ϵ-approx to $A_\ell B_{-,m}$. Hence, for each $0 \le \ell, m < n$, $(\widetilde{A}\widetilde{B})_{\ell,m}$ is an ϵ-approx to $(AB)_{\ell,m}$. Hence we have the following lemma.

LEMMA 5.6. *Suppose there is a (k, Δ) circle ϵ-approx evaluation scheme. Then given two $n \times n$ matrices A, B, we can ϵ_1-approximate the inner product AB by $\widetilde{A}\widetilde{B}$ for*

$$\epsilon_1 \le \epsilon(\epsilon + |\mathbf{u}| + |\mathbf{v}|).$$

PROPOSITION 5.7. *An $n \times n$ matrix of rank $< n$ cannot ϵ_1-approximate an $n \times n$ identity matrix, for any $\epsilon_1 < 1$.*

Proof. Suppose, for the sake of contradiction, that an $n \times n$ matrix $\widetilde{M}$ of rank $< n$ is an ϵ_1-approx of $n \times n$ matrix I, so $\|\widetilde{M} - I\|_\infty \le \epsilon_1$. Then since $\widetilde{M}$ has rank $< n$, there is an $x \ne 0$ such that $\|x\|_\infty = 1$ and $\widetilde{M}x = 0$. Thus

$$\|\widetilde{M}x - Ix\|_\infty = \|Ix\|_\infty = \|x\|_\infty = 1,$$

so

$$\|\widetilde{M} - I\|_\infty \geq \|\widetilde{M}x - Ix\|_\infty / \|x\|_\infty$$

$$= 1 > \epsilon_1,$$

a contradiction. □

We now consider the case where A, B are $n \times n$ identity matrices. Their product $M = AB$ is also an identity matrix. But $\widetilde{M} = \widetilde{A}\widetilde{B}$ is the product of an $n \times (kH)$ matrix and a $(kH) \times n$ matrix and thus has rank $\leq kH$. Recall that the statement of Theorem 5.1 makes the assumption that $kH = k\lceil 2\pi/\Delta \rceil < n$, so $\widetilde{M}$ has rank $< n$. But Lemma 5.6 states that $\widetilde{M}$ is an ϵ_1-approximation to $I = AB$, a contradiction of Proposition 5.7. Hence Theorem 5.1 follows. □

Appendix A. Proof of an alternative approximate evaluation on a circle.

Proof of Theorem 4.5. We give a proof of this alternative construction in stages, first considering a somewhat more complex algorithm.

An approach to approximate evaluation on a circle via approximate real polynomial evaluation. We will derive and prove the alternative algorithm given in subsection 4.2 by the use of classic real polynomial approximation techniques which provide our error analysis, but we observe at the end of this subsection that we do not need to explicitly construct the real approximation polynomials. Restrict z to the circle of radius r, so $z = r(x + iy)$, where x, y are real and $x + iy$ is on the unit circle, so $y = \sqrt{1 - x^2}$. Note that an expansion of $P(z)$ in terms of x, y gives $P(z) = R(x) + iS(x)y$, for polynomials $R(x), S(x)$ of degree $n - 1$. For simplicity, let us assume, w.l.o.g., $P(z)$ has real coefficients $p_0, \ldots, p_{n-1} \in \mathcal{R}$, so $R(x), S(x)$ are real polynomials. The exact evaluation of $P(z)$ at the evaluation points z_j, for $j = 0, \ldots, n-1$, might be done by an exact evaluation of $R(x), S(x)$ at the real points $x_0, \ldots, x_{n-1}$, where $z_j = x_j + iy_j$, so $P(z_j) = R(x_j) + iS(x_j)y_j$. Instead, we consider (see also Bini and Pan [3]) an approach for an ϵ-approximate evaluation of $P(z)$ at the evaluation points $z_0, \ldots, z_{m-1}$ using approximation of real polynomials (we will later show we can avoid explicit construction of these real polynomials). Fix a number k which divides n, to be determined below. We could construct degree $k-1$ polynomials $\widetilde{R}(x), \widetilde{S}(x)$, which give an ϵ'-approx of polynomials $R(x), S(x)$, where $\epsilon' = \epsilon/\sqrt{2}$. Then $\widetilde{P}(z) = \widetilde{R}(x) + i\widetilde{S}(x)y$ is an ϵ-approx of $P(z)$ over the circle of radius r, since

$$|P(z) - (\widetilde{R}(x) + i\widetilde{S}(x)y)| \leq \sqrt{(R(x) - \widetilde{R}(x))^2 + (S(x) - \widetilde{S}(x))^2} \leq \sqrt{2}\epsilon' = \epsilon.$$

To construct $\widetilde{R}(x), \widetilde{S}(x)$, we could do an exact evaluation of $R(x), S(x)$ at the shifted $(2k, k)$-Chebyshev points $x'_j = \Delta cos(j\pi/k)$, for $j = 0, \ldots, k-1$ over the real interval $I = [-\Delta, \Delta]$, for a positive real $\Delta \leq 1$. To do this first step, we can exactly evaluate $P(z)$ at

$$z'_j = r\left(x'_j + i\sqrt{1 - (x'_j)^2}\right),$$

for $j = 0, \ldots, k-1$, which costs work $O(n \log k)$ by Proposition 1.4. Then, we could exactly interpolate a degree $k-1$ real polynomial $\widetilde{R}(x)$ from the real parts $R(x'_j)$ of $P(z'_j)$, for $j = 0, \ldots, k-1$. We could also interpolate a degree $k-1$ real polynomial $\widetilde{S}(x)$ from the real values $s_0, s_1, \ldots, s_{k-1}$, where

$$is_j\sqrt{1 - (x'_j)^2}$$

is the complex part of $P(z'_j)$. By Lemma 3.1, the exact interpolation of polynomials $\widetilde{R}(x), \widetilde{S}(x)$ at these $(2k, k)$-Chebyshev points cost work $O(k \log k)$.

Exponential coefficient growth: The key difficulty with this approach. Next we need to bound the approximation errors for evaluation of $\widetilde{R}(x), \widetilde{S}(x)$ at all the evaluation points. To apply Proposition 3.2 we need to upper bound, the maximum moduli of $R^{(k)}(x) = \frac{d^k R(x)}{d^k x}$ (the kth derivative of $R(x)$ with respect to x), and $S^{(k)}(x) = \frac{d^k S(x)}{d^k x}$ over the interval $I = [-\Delta, \Delta]$. Unfortunately, straightforward application of the binomial expansion (in terms of x and y) to bound the moduli of coefficients of $R(x)$ and $S(x)$ give bounds on $|R|, |S|$ (the sums of the moduli of the coefficients of $R(x), S(x)$) that grow exponentially with the degree $n-1$. Thus application of Lemma 3.4 to evaluate $\widetilde{R}(x), \widetilde{S}(x)$ at all the evaluation points gives the work bound of the form

$$O(n \log \log(|R|/\epsilon')) = O(n \log(n + \log(1/\epsilon'))),$$

since in this case

$$\log(2^n/\epsilon') = n + \log(1/\epsilon').$$

The same difficulty occurs for ϵ'-approximate evaluation of the polynomial $S(x)$.

Evaluation at restricted intervals. To avoid this difficulty, we will do only an ϵ-approximate evaluation of $P(z)$ at a restricted interval on the radius r circle, where $|\cos(\theta(z))| \leq \Delta$ or, for the case of descending input coefficients,

$$|x| \leq \frac{1}{\sqrt{2}} \leq \Delta = \frac{1}{\max(\sqrt{2}, 8(n-1-k)/\sqrt{2})}.$$

Then we can do an ϵ-approximate evaluation of $P(z)$ over the entire circle of radius r by a number of separate evaluations of $P(z)$, each at a consecutive segment of the circle. The following lemma provides useful bounds on the k derivatives of $R(x), S(x)$.

LEMMA A.1. *Suppose $r = 1$ and either 1. $\Delta \leq \frac{k}{e(n-k)}$ or 2. $P(z)$ is $(k, \sqrt{2}/8)$-descending. For*

$$|x| \leq \Delta \leq \frac{1}{\sqrt{2}},$$

the modulus of the kth derivative (with respect to x) of $z = r(x + iy)$ (where $x + iy$ is on the unit circle) is bounded as

$$\left|\frac{d^k z}{d^k x}\right| \leq O(rk!8^k).$$

Also,

$$\left|\frac{d^k P(z)}{d^k x}\right|, \left|R^{(k)}(x)\right|,$$

and $|S^{(k)}(x)|$ are bounded in case 1 by $O(k!|P|\Delta^{-k})$ and in case 2 by $O(k!n^{O(1)}|P|\Delta^{-k})$.

Proof. We shall show that $\frac{dz}{dx} = r(1 - if_1(x))$ and, for $k \geq 2$, $\frac{d^k z}{d^k x} = -ir f_k(x)$, where $f_k(x)$ is a sum of at most 2^{k-1} terms, each of the form $c(k)\frac{x^a}{y^{2b+1}}$, where $|c(k)| \leq O(k!2^k)$ is the term's coefficient, and $0 \leq a, b \leq k$ are integers. We prove this by

construction of a *derivative tree*, which is a binary tree whose nodes are such terms. Since $y = \sqrt{1-x^2}$, $\frac{dy}{dx} = \frac{-x}{y^3}$, we have $\frac{dz}{dx} = 1 - if_1(x)$ where $f_1(x) = \frac{x}{y^3}$, so we define the root to be $f_1(x)$. Note that for $k > 1$,

$$\frac{d^k z}{d^k x} = -i\frac{d^{k-1} f_1(x)}{d^{k-1} x}.$$

Inductively, assume the label of a node is a term of form $c(k)\frac{x^a}{y^{2b+1}}$ labeling a node, since the derivative of $c(k)\frac{x^a}{y^{2b+1}}$ with respect to x is the sum of terms $c(k)(2b+1)$ $\frac{x^a}{y^{2(b+1)+1}}$ and $c(k)a\frac{x^{a-1}}{y^{2b+1}}$, we let its left and right children be labeled by these two terms, respectively. Note that since we take k derivatives, the tree has depth $k-1$, and so there are at most 2^{k-1} terms at the leaves of the tree. The leaves will be the terms of $\frac{d^{k-1} f_1(x)}{d^{k-1} x}$. The leaf with largest coefficient modulus is reached by a length $k-1$ path of left branches down the tree, so the maximum coefficient modulus of any leaf can be upper bounded as

$$|c(k)| \leq (2k-1)(2k-3)\cdots 3 \cdot 1 \leq k! 2^{k-1}.$$

Since $|x| \leq \frac{1}{\sqrt{2}}$,

$$y \geq \sqrt{1-x^2} \geq 1/\sqrt{2},$$

so

$$1/y^{2b+1} \leq (\sqrt{2})^{2k+1} \leq 2^{k+1}.$$

Thus the modulus of each leaf term $c(k)\frac{x^a}{y^{2b+1}}$ is upper bounded by $|c(k)|2^{k+1} \leq k!2^{k-1}2^{k+1}\ O(k!4^k)$. Hence $|f_k(x)| \leq$ the product of the number 2^{k-1} of terms at the leaves times their maximum modulus $O(k!4^k)$, so $|f_k(x)| \leq O(k!8^k)$. This implies

$$\left|\frac{d^k z}{d^k x}\right| \leq r(1 + |f_k(x)|) \leq O(rk!8^k).$$

Note that for $j > k$,

$$\frac{j!}{(j-k)!} \approx (j-k)^k$$

for $k = o(n)$, since by the Stirling approximation to factorial,

$$j! \approx (j)^j e^{-j}\sqrt{2\pi j},$$

$$(j-k)! \approx (j-k)^{j-k} e^{-(j-k)}\sqrt{2\pi(j-k)},$$

and

$$\left(\frac{j}{j-k}\right)^{j-k} = \left(1 + \frac{k}{j-k}\right)^{j-k} \approx e^k;$$

so

$$\frac{j!}{(j-k)!} \approx \frac{j^j}{(j-k)^{j-k}} e^{-k} \approx (j-k)^k e^{-k} e^k \approx (j-k)^k.$$

Since $|z| \le r$, we have $|\frac{d^k P(z)}{d^k z}| \le \sum_{j=k}^{n-1} \frac{j!}{(j-k)!} r^{j-k} |p_j|$. By the chain rule of derivatives,

$$\frac{d^k P(z)}{d^k x} = \frac{d^k z}{d^k x} \frac{d^k P(z)}{d^k z},$$

so

$$\left| \frac{d^k P(z)}{d^k x} \right| \le \left| \frac{d^k z}{d^k x} \right| \left| \frac{d^k P(z)}{d^k z} \right| \le O(rk!8^k) \sum_{j=k}^{n-1} \frac{j!}{(j-k)!} r^{j-k} |p_j|.$$

Thus

$$\left| \frac{d^k P(z)}{d^k x} \right| \le O(rk!8^k) \sum_{j=k}^{n-1} \frac{j!}{(j-k)!} r^{j-k} |p_j| \le O(rk!8^k) \sum_{j=k}^{n-1} (j-k)^k r^{j-k} |p_j|.$$

1. Now suppose $r = 1$ and $\Delta \le \frac{1}{8(n-k)}$. Then $\sum_{j=k}^{n-1} (8(j-k))^k |p_j| \le |P| \Delta^{-k}$, so

$$(8(j-k))^k |p_j| \le |P| (1/\sqrt{2})^k \le O(|P| \Delta^{-k}).$$

   Thus,

$$\left| \frac{d^k P(z)}{d^k x} \right| \le O(rk!8^k) \sum_{j=k}^{n-1} (j-k)^k r^{j-k} |p_j| \le O(k!) \sum_{j=k}^{n-1} (8(j-k))^k r^{j-k+1} |p_j|$$

$$\le O(k! |P| \Delta^{-k}).$$

   Since $P(z) = R(x) + iS(x)y$, we have

$$\frac{d^k P(z)}{d^k x} = R^{(k)}(x) + i \left(S^{(k)}(x) y + \frac{x S(x)}{y} \right).$$

   Hence for $|x| \le \Delta$,

$$|R^{(k)}(x)| \le \left| \frac{d^k P(z)}{d^k x} \right| \le O(k! |P| \Delta^{-k}).$$

   Also, $|\frac{xS(x)}{y}| \le |P(z)| \le |P|$, so

$$\begin{aligned} |S^{(k)}(x)| &\le \left| \frac{d^k P(z)}{d^k x} \right| + \left| \frac{x S(x)}{y} \right| \le \left| \frac{d^k P(z)}{d^k x} \right| + |S(x)| \left| \frac{x}{y} \right| \\ &\le \left| \frac{d^k P(z)}{d^k x} \right| + O(|P|) \\ &\le O(k! |P| \Delta^{-k}). \end{aligned}$$

2. Next suppose $r = 1$. If $P(z)$ is $(k, \sqrt{2}/8)$-descending, then, by definition, the magnitude of the coefficients of $P(z)$ drops as

$$|p_j| \le n^{O(1)} |P| \left(\frac{\sqrt{2}}{j8} \right)^k$$

for $j = k, \ldots, n-1$. Then $\sum_{j=k}^{n-1}(8(j-k))^k|p_j| \leq |P|\Delta^{-k})$, so

$$(8(j-k))^k|p_j| \leq n^{O(1)}|P|(1/\sqrt{2})^k \leq O(n^{O(1)}|P|\Delta^{-k}).$$

Thus in this case,

$$\left|\frac{d^k P(z)}{d^k x}\right| \leq O(rk!8^k)\sum_{j=k}^{n-1}(j-k)^k|p_j| \leq O(k!)\sum_{j=k}^{n-1}(8(j-k))^k|p_j|$$
$$\leq O(k!n^{O(1)}|P|\Delta^{-k}).$$

Also,

$$|S^{(k)}(x)| \leq O(k!n^{O(1)}|P|\Delta^{-k}).$$

Since $|I| = 2\Delta$, for $I = [-\Delta, \Delta]$, by Propositions 3.2 and A.1, the error in the approximation of $R(x)$ by $\widetilde{R}(x)$, for $|x| \leq \Delta$, is upper bounded in case 1 by

$$\frac{2}{k!}(|I|/4)^k \max_{x\in I} R^{(k)}(x) \leq O(|P|(\Delta/2)^k\Delta^{-k}) \leq O(|P|2^{-k})$$

and in case 2 by $O(n^{O(1)}|P|2^{-k})$. This error is upper bounded by $O(\epsilon')$ if we set $k = \log(|P|/\epsilon) + O(1)$ in case 1 or $k = O(\log(n|P|/\epsilon))$ in case 2. Also, the error in the approximation of $S(x)$ by $\widetilde{S}(x)$, for $|x| \leq \Delta$, is upper bounded by

$$\frac{2}{k!}(|I|/4)^k \max_{y\in I} S^{(k)}(y) \leq \epsilon'.$$

Hence

$$|P(z) - (\widetilde{R}(x) + i\widetilde{S}(x)y)| \leq ((R(x) - \widetilde{R}(x))^2 + (S(x) - \widetilde{S}(x))^2)^{1/2} \leq \sqrt{2}\epsilon' = \epsilon.$$

Thus we have proved the error bounds given in Theorem 4.5 for the case

$$\widetilde{P}(z) = \widetilde{R}(x) + i\widetilde{S}(x)y.$$

A simplification avoiding construction of real polynomials. Finally, we observe that we do not need to explicitly construct real approximation polynomials defined above. For simplicity, we assume, w.l.o.g., for each evaluation point z_j that $|\theta(z_j) - \pi/2| \leq \pi/4$. Instead, we do the following:

- In work $O(n\log^2 k)$, interpolate the degree $k-1$ complex polynomial $\widetilde{P}(z)$ from the values $P(z'_j)$, for $j = 0, \ldots, k-1$, where $z'_j = r(x'_j + i\sqrt{1-(x'_j)^2})$, and $x'_j = \cos(j\pi/k)/\sqrt{2}$, for $j = 0, \ldots, k-1$.
  Comment: Thus we construct $\widetilde{P}(z) = (\widetilde{R}(x) + i\widetilde{S}(x)y)$, without explicitly constructing $\widetilde{R}(x), \widetilde{S}(x)$.
- Evaluate polynomial $\widetilde{P}(z)$ exactly at the given evaluation points $z_0, \ldots, z_{m-1}$, within work $O(m\log^c k)$, where $c = 2$ by Proposition 1.1 (or $c = 1$ by Proposition 1.4 if these evaluation points are regularly spaced).
  Comment: Thus we evaluate $\widetilde{P}(z) = (\widetilde{R}(x)+i\widetilde{S}(x)y)$, at the given evaluation points, without explicitly evaluating $\widetilde{R}(x), \widetilde{S}(x)$.

Thus we have derived the algorithm as defined in subsection 4.2, which provides a somewhat simpler implementation of Theorem 4.5. ∎

Appendix B. Exact Chebyshev point evaluation in $O(n \log n)$ work.

Let $P(x)$ be a polynomial of degree $n-1$. Let $\omega = e^{i2\pi/n'}$ be the n'th root of unity over the complex numbers $\mathcal{C}$, for some $n' \geq n$, such that n divides n'. We wish to exactly evaluate $P(x)$ at the (n', n)-Chebyshev points, which we defined to be the real parts $x_{0,j} = \cos(j2\pi/n')$ of the powers $z_{0,j} = \omega^j = e^{ji2\pi/n'}$, for $j = 0, \ldots, n-1$.

For simplicity, we assume n is a power of 2 and $n' = 2n$. For each $\ell = 0, \ldots, \log n$,

- let $n_\ell = n/2^\ell, n'_\ell = n'/2^\ell$,
- let $z_{\ell,j+n_\ell} = -z_{\ell,j}$, and if $\ell > 0$ then let $z_{\ell,j} = 2z^2_{\ell-1,j} - 1$, and
- let $x_{\ell,j} = \text{ real}(e^{ji2\pi/n'_\ell}) = \cos(j2\pi/n'_\ell)$ denote the (n'_ℓ, n_ℓ)-Chebyshev points.

By the identities $\cos(2\theta) = 2\cos^2(\theta) - 1$ and $\cos(\theta + \pi) = -\cos(\theta)$, we have a recursive definition of these (n'_ℓ, n_ℓ)-Chebyshev points.

PROPOSITION B.1. *For each $\ell = 0, \ldots, \log n$ and all $j = 0, \ldots, n_\ell - 1$, $x_{\ell,j+n_\ell} = -x_{\ell,j}$ and if $\ell > 0$ then $x_{\ell,j} = 2x^2_{\ell-1,j} - 1$.*

The key idea behind our algorithm is as follows:

- Starting with the polynomial P to be evaluated, we define two polynomials P_0 and P_1 as the unique polynomials such that $P(x) = P_0(2x^2-1) + xP_1(2x^2-1)$.
- The map from x to $2x^2 - 1$ sends $\cos\theta$ to $\cos 2\theta$, that is, it maps Chebyshev points to Chebyshev points. If the original set of Chebyshev points contains both x and $-x$, then this map collapses them, so the number of points to be evaluated is halved. Thus the problem of evaluating P at the (n', n)-Chebyshev points reduces to evaluating P_0 and P_1 at $(n'/2, n/2)$-Chebyshev points.
- To apply this argument recursively, we compute the polynomials at each node of a logarithmic depth tree, by their evaluations on suitably defined points. This process can be initiated at the leaves, since there evaluation is the same as the coefficient itself.

We define a full binary tree of depth $\log n$ which we call the (n', n)-*Chebyshev evaluation tree*, whose nodes of depth $\ell = 0, \ldots, \log n$ are the set $\{0,1\}^\ell$ of binary strings of length ℓ. The root is labeled with the input polynomial $P_\lambda(x) = P(x)$, where λ is the empty string. Each node $\beta \in \{0,1\}^\ell$ of the tree is labeled with a degree $n_\ell - 1$ polynomial $P_\beta(x)$ and if $\ell < \log n$, node β has two children $\beta 0, \beta 1$ with labels consisting of the unique degree $n_{\ell+1} - 1$ polynomials $P_{\beta 0}(x), P_{\beta 1}(x)$ such that $P_\beta(x) = P_{\beta 0}(2x^2 - 1) + xP_{\beta 1}(2x^2 - 1)$. In contrast with the $O(n \log^2 n)$ algorithm of Pan [29], we will not actually compute these polynomials, but they will aid us in definition of our algorithm. However, we will compute for each node $\beta \in \{0,1\}^\ell$ two sets of values

$$\{F_{\beta,j} = P_\beta(z_{\ell,j}) | j = 0, \ldots, n_\ell - 1\}$$

and

$$\{\phi_{\beta,j} = P_\beta(x_{\ell,j}) | j = 0, \ldots, n_\ell - 1\}.$$

PROPOSITION B.2. *For each $\beta \in \{0,1\}^{\log n}$, $\phi_{\beta,0} = F_{\beta,0}$. For each $\ell = 0, \ldots, (\log n) - 1$, $\beta \in \{0,1\}^\ell$ and $j = 0, \ldots, n_\ell - 1$,*

1. *$F_{\beta,j} = F_{\beta 0,j} + z_{\ell,j}F_{\beta 1,j}$, $F_{\beta,j+n_\ell} = F_{\beta 0,j} - z_{\ell,j}F_{\beta 1,j}$, and so*
2. *$F_{\beta 0,j} = (F_{\beta,j} + F_{\beta,j+n_\ell})/2$, $F_{\beta 1,j} = (F_{\beta,j} - F_{\beta,j+n_\ell})/(2z_{\ell,j})$, and similarly*
3. *$\phi_{\beta,j} = \phi_{\beta 0,j} + x_{\ell,j}\phi_{\beta 1,j}$, $\phi_{\beta,j+n_\ell} = \phi_{\beta 0,j} - x_{\ell,j}\phi_{\beta 1,j}$, and so*

4. $\phi_{\beta 0,j} = (\phi_{\beta,j} + \phi_{\beta,j+n_\ell})/2$, $\phi_{\beta 1,j} = (\phi_{\beta,j} - \phi_{\beta,j+n_\ell})/(2x_{\ell,j})$.

Proof. For $\ell = |\beta| = \log n$, since P_β is a degree 0 polynomial, we have $\phi_{\beta,0} = P_\beta = F_{\beta,0}$.

1. By definition, for $\ell \geq 0$, $F_{\beta,j} = P_\beta(z_{\ell,j}) = P_{\beta 0}(z_{\ell+1,j}) + z_{\ell,j}P_{\beta 1}(z_{\ell+1,j}) = F_{\beta 0,j} + z_{\ell,j}F_{\beta 1,j}$. Also by definition, $z_{\ell,j+n_\ell} = -z_{\ell,j}$, so $F_{\beta,j+n_\ell} = P_\beta(z_{\ell,j+n_\ell}) = P_\beta(-z_{\ell,j}) = P_{\beta 0}(2(-z_{\ell,j})^2 - 1) - z_{\ell,j}P_{\beta 1}(2(-z_{\ell,j})^2 - 1) = P_{\beta 0}(z_{\ell+1,j}) - z_{\ell,j}P_{\beta 1}(z_{\ell+1,j}) = F_{\beta 0,j} - z_{\ell,j}F_{\beta 1,j}$.
2. The reverse formulas for $F_{\beta 0,j}, F_{\beta 1,j}$ follow from linear solution of the previous two equations.
3. By definition, for $\ell \geq 0$, $\phi_{\beta,j} = P_\beta(x_{\ell,j}) = P_{\beta 0}(x_{\ell+1,j}) + x_{\ell,j}P_{\beta 1}(x_{\ell+1,j}) = \phi_{\beta 0,j} + x_{\ell,j}\phi_{\beta 1,j}$. By Proposition B.1, $x_{\ell,j+n_\ell} = -x_{\ell,j}$, so $\phi_{\beta,j+n_\ell} = P_\beta(x_{\ell,j+n_\ell}) = P_\beta(-x_{\ell,j}) = P_{\beta 0}(2(-x_{\ell,j})^2 - 1) - x_{\ell,j}P_{\beta 1}(2(-x_{\ell,j})^2 - 1) = P_{\beta 0}(x_{\ell+1,j}) - x_{\ell,j}P_{\beta 1}(x_{\ell+1,j}) = \phi_{\beta 0,j} - x_{\ell,j}\phi_{\beta 1,j}$.
4. The reverse formulas for $\phi_{\beta 0,j}, \phi_{\beta 1,j}$ follow from linear solution of the previous two equations. □

Our algorithms for exact Chebyshev point evaluation and interpolation. We now show how to do (n', n)-Chebyshev point evaluation by a recursive algorithm and reduction to the DFT.

For each $\ell = 0, \ldots, (\log n) - 1$, we precompute $x_{\ell,j} = \cos(j2\pi/n')$, and $z_{\ell,j}$ for $j = 0, \ldots, n_\ell - 1$ as defined above. We then compute the $(2n, n)$-Chebyshev point evaluation of a given $P(x)$ as follows:

$(2n, n)$-Chebyshev point evaluation algorithm.

Input: The n coefficients $p_0, \ldots, p_{n-1} \in \mathcal{C}$ polynomial $P(x)$ of degree $\leq n - 1$.

1. In work $O(n \log n)$, compute the values $F_{\lambda,j} = P(\omega^j), j = 0, \ldots, n - 1$, where λ is the empty string (this can be done as stated in Proposition 1.3 within work $O(n \log n)$ by the chirp transform, which generalizes the DFT to this case).
2. Traverse the Chebyshev evaluation tree from root to leaves:
   For each $\ell = 0, \ldots, (\log n) - 1$ **do** apply Proposition B.2, part 2:
   For each $j = 0, \ldots, n_\ell - 1$ and $\beta \in \{0, 1\}^\ell$ **do**
   let $F_{\beta 0,j} = (F_{\beta,j} + F_{\beta,j+n_\ell})/2$, $F_{\beta 1,j} = (F_{\beta,j} - F_{\beta,j+n_\ell})/(2z_{\ell,j})$.
3. **For each** $j = 0, \ldots, n_\ell - 1$ and $\beta \in \{0, 1\}^\ell$ **do**
   let $\phi_{\beta,0} = F_{\beta,0}$, as given by Proposition B.2.
4. Traverse the Chebyshev evaluation tree from leaves to root:
   For each $\ell = (\log n) - 1, \ldots, 0$ **do** apply Proposition B.2, part 3:
   For each $j = 0, \ldots, n_\ell - 1$ and $\beta \in \{0, 1\}^\ell$ **do**
   let $\phi_{\beta,j} = \phi_{\beta 0,j} + x_{\ell,j}\phi_{\beta 1,j}$, $\phi_{\beta,j+n_\ell} = \phi_{\beta 0,j} - x_{\ell,j}\phi_{\beta 1,j}$.

Output: $\phi_{\lambda,j} = P(x_{\lambda,j}), j = 0, \ldots, n - 1$, which is the $(2n, n)$-Chebyshev evaluation of $P(x)$.

For each $\ell = 0, \ldots, \log n$, the work is $O(n_\ell) = O(n/2^\ell)$ for each of the 2^ℓ nodes $\beta \in \{0, 1\}^\ell$ of depth ℓ, so the total work is $O(n_\ell 2^\ell \log n) = O(n \log n)$, plus $O(n \log n)$ work for computing the chirp transform by Proposition 1.3.

We can apply this $(2n, n)$-Chebyshev point evaluation algorithm to compute an (n', n)-Chebyshev point evaluation of degree $n-1$ polynomial $P(x)$, for any $n' = O(n)$ such that n divides n'. In the case $n' \geq 2n$, we can view the polynomial $P(x)$ to be of degree $(n'/2) - 1$, and apply the $(n', n'/2)$-Chebyshev point evaluation algorithm. In the case $n' < 2n$, since n divides n', we can decompose $P(x) = \sum_{j=0}^{c-1} P_j(x)x^{jn}$, where $c = n'/(2n) = O(1)$ and each polynomial $P_j(x)$ is of degree $\leq (n/c) - 1$, and then apply the $(2n/c, n/c)$-Chebyshev point evaluation algorithm a constant c number of

times.

Thus we have proved that the (n', n)-Chebyshev point evaluation problem for a polynomial of degree $n-1$ can be solved within work $O(n \log n)$ for any $n' = O(n)$ such that n divides n'. Since the Chebyshev evaluation tree has depth $\log n$, and there are $2 \log n$ stages, each taking $O(1)$ time with n work, and the DFT takes $O(\log n)$ with $O(n \log n)$ work, it follows that the total circuit depth is $O(\log n)$ with $O(n \log n)$ work. This completes the first part of the proof of Lemma 3.1. We next show how to do (n', n)-Chebyshev point interpolation by reduction to the DFT. We compute the interpolation of a polynomial from a set of n values at $(2n, n)$-Chebyshev points, as follows:

$(2n, n)$-Chebyshev point interpolation algorithm.

Input: The values $v_0, \ldots, v_{n-1} \in \mathcal{C}$ at the $(2n, n)$-Chebyshev points: $x_{0,0}, \ldots, x_{0,n-1}$.

1. Let $\phi_{\lambda,j} = v_j), j = 0, \ldots, n-1$.
2. Traverse the Chebyshev evaluation tree from root to leaves:
   For each $\ell = 0, \ldots, (\log n) - 1$ **do** apply Proposition B.2, part 4:
   For each $j = 0, \ldots, n_\ell - 1$ and $\beta \in \{0,1\}^\ell$ **do**
   let
$$\phi_{\beta 0,j} = (\phi_{\beta,j} + \phi_{\beta,j+n_\ell})/2,$$
$$\phi_{\beta 1,j} = (\phi_{\beta,j} - \phi_{\beta,j+n_\ell})/(2x_{\ell,j}).$$
3. **For each** $j = 0, \ldots, n_\ell - 1$ and $\beta \in \{0,1\}^\ell$ **do**
   let $F_{\beta,0} = \phi_{\beta,0}$, as given by Proposition B.2.
4. Traverse the Chebyshev evaluation tree from leaves to root:
   For each $\ell = (\log n) - 1, \ldots, 0$ **do** apply Proposition B.2, part 1:
   For each $j = 0, \ldots, n_\ell - 1$ and $\beta \in \{0,1\}^\ell$ **do**
   let
$$F_{\beta,j} = F_{\beta 0,j} + z_{\ell,j} F_{\beta 1,j},$$
$$F_{\beta,j+n_\ell} = F_{\beta 0,j} - z_{\ell,j} F_{\beta 1,j}.$$
5. In work $O(n \log n)$, compute the inverse DFT of the values $F_{\lambda,j} = P(\omega^j), j = 0, \ldots, n-1$ (this can be done within work $O(n \log n)$ as stated in Proposition 1.3), giving:

Output: The n coefficients of the unique interpolated polynomial $P(x)$.

For each $\ell = 0, \ldots, \log n$, the work is again $O(n_\ell) = O(n/2^\ell)$ for each of the 2^ℓ nodes $\beta \in \{0,1\}^\ell$ of depth ℓ, so the total work is again $O(n_\ell 2^\ell \log n) = O(n \log n)$, plus $O(n \log n)$ work for computing the DFT as in Proposition 1.3.

We can also apply this $(2n, n)$-Chebyshev point interpolation algorithm to compute an (n', n)-Chebyshev point interpolation of degree $n-1$ polynomial $P(x)$ for any $n', n \le n' < 2n$, such that n divides n'. We apply the $(2\bar{n}, \bar{n})$-Chebyshev point interpolation algorithm, for $\bar{n} = n'/c$, to interpolate c polynomials of degree $\bar{n}-1$ from the given (n', n)-Chebyshev points, and then construct $P(x)$ in further work $O(n \log n)$ by the well-known divide-and-conquer interpolation methods of [16, 26, 5, 1]. Thus we have proved that the (n', n)-Chebyshev point interpolation problem for a polynomial of degree $n-1$ can be solved within work $O(n \log n)$ for any $n' \le 2n$ such that n divides n'. The shifted version of these evaluation and interpolation problems can

be solved within the same work bounds using the same techniques by initializing the recurrences at the leaves to the appropriate values corresponding to the shifts. Again, since the Chebyshev evaluation tree has depth $\log n$, and there are $2 \log n$ stages, each taking $O(1)$ time with n work, and the inverse DFT takes $O(\log n)$ with $O(n \log n)$ work, it follows that the total circuit depth is $O(\log n)$ with $O(n \log n)$ work. To complete the proof of Lemma 3.1, we apply the technique of Proposition 1.1 to solve the (n', m)-Chebyshev point evaluation problem, for $n' = O(n)$, in work $O(n + m \log \min(n, m))$. □

Acknowledgments. The author sincerely thanks Chris Lambert, Victor Pauca, Kathy Redding, Ken Robinson, Steven Tate, and especially Deganit Armon, Octavian Procopiuc, and D. Sivakumar for comments improving the presentation of the results in this paper. Don Coppersmith suggested the reduction to the approximation of a matrix by small rank used in the proof of Theorem 5.1. Donald Rose described to us a simple proof of Proposition 5.7.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, London, 1974.
[2] A. V. Aho, K. Steiglitz, and J. D. Ullman, *Evaluating Polynomials at Fixed Sets of Points*, SIAM J. Comput., 4 (1975), pp. 533–539.
[3] D. Bini and V. Pan, *Polynomial and Matrix Computations*, 1, Birkhauser, Boston, MA, 1994.
[4] A. B. Borodin and I. Munro, *Evaluating polynominals at many points*, Inform. Process. Lett., 1 (1971), pp. 660–68.
[5] A. B. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numerical Problems*, American Elsevier, New York, 1975.
[6] P. B. Callahan and S. R. Kosaraju, *A decomposition of multi-dimensional point sets with applications to k-nearest-neighbors and n-body potential fields*, J. ACM, 42 (1995), pp. 67–90.
[7] J. Carrier, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Comput., 9 (1998), pp. 669–686.
[8] J. M. Cooley, P. A. Lewis, and P. D. Welch, *History of the fast Fourier transform*, Proc. IEEE, 55 (1967), pp. 1675–1677.
[9] J. M. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297–301.
[10] G. Dahlquist and A. Björck, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
[11] G. C. Danielson and C. Lanczos, *Some improvements in practical Fourier analysis and their application to X-ray scattering from liquids*, J. Franklin Inst., 233 (1942), pp. 365–380 and pp. 435–452.
[12] A. Dutt, M. Gu, and V. Rokhlin, *Fast algorithms for polynomial interpolation, integration, and differentiation*, SIAM J. Numer. Anal., 33 (1996), pp. 1689–1711.
[13] A. Dutt, *Fast Fourier Transforms for Nonequispaced Data*, Technical Report 980, Department of Computer Science, Yale University, New Haven, CT, 1993.
[14] A. Dutt and V. Rokhlin, *Fast Fourier Transforms for Nonequispaced Data II*, in Applied and Computational Harmonic Analysis, Academic Press, 2 (1995), pp. 85–100.
[15] W. D. Elliott and J. A. Board, *Fast fourier transform accelerated fast multipole algorithm*, SIAM J. Sci. Comput., 17 (1996), pp. 398–415.
[16] C. M. Fiduccia, *Polynomial evaluation via the division algorithm—the fast Fourier transform revisited*, in Proceedings of the 4th Annual ACM Symposium on Theory of Computing, 1972, pp. 88–93.
[17] W. M. Gentleman and G. Sande, *Fast Fourier transforms for fun and profit*, in Proceedings of the AFIPS 1966 Fall Joint Computer Conference, 29, 1966, pp. 563–578.
[18] A. Gerasoulis, *A fast algorithm for the multiplication of generalized Hilbert matrices with vectors*, Math. Comp., 50 (1988), pp. 179–188.
[19] A. Gerasoulis, M. D. Grigoriadis, and Liping Sun, *A fast algorithm for Trummer's problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s135–s138.
[20] I. J. Good, *The interaction algorithm and practical Fourier series*, J. Roy. Statist. Soc. Ser.

A, 20 (1958), pp. 361–372. Addendum, 22 (1960), pp. 372–375.
[21] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.
[22] L. Greengard and V. Rokhlin, *On the efficient implementation of the fast multipole algorithm*, Technical Report RR-602, Dept. of Computer Science, Yale University, New Haven, CT, 1988.
[23] P. Henrici, *Applied and Computational Complex Analysis*, III, John Wiley, New York, 1986.
[24] M. S. Henry, *Approximation by polynomials: Interpolation and optimal nodes*, Amer. Math. Monthly, 91 (1984), pp. 497–499.
[25] E. Horowitz, *A fast method for interpolation using preconditioning*, Inform. Process. Lett., 1 (1972), pp. 157–163.
[26] H. T. Kung, *Fast evaluation and interpolation*, Department of Computer Science, Carnegie-Mellon University., Pittsburgh, PA, 1973, unpublished manuscript.
[27] R. Moenck and A. B. Borodin, *Fast modular transforms via division*, in Conf. Record, IEEE 13th Ann. Symposium on Switching and Automata Theory, 1972, pp. 90–96.
[28] A. C. R. Newbery, *Error analysis for polynomial evaluation*, Math. Comp., 28 (1979), pp. 789–793.
[29] V. Pan, *Fast Evaluation and interpolation at the Chebyshev sets of points*, Appl. Math. Lett., 2 (1989), pp. 255–258.
[30] V. Y. Pan, J. H. Reif, and S. R. Tate, *The power of combining the techniques of algebraic and numerical computing: Improved approximate multipoint polynomial evaluation and improved multipole algorithms*, in Proceedings of the 32nd Annual IEEE Symposium Foundations of Computer Science, Pittsburgh, PA, 1992, pp. 703–713.
[31] V. Pan, A. Sadikou, E. Landowne, and O. Tiga, *A new approach to fast polynomial interpolation and multipoint evaluation*, Comput. Math. Appl., 25 (1993), pp. 25–30.
[32] L. R. Rabiner and C. M. Rader, eds., *Digital Signal Processing*, IEEE Press, New York, 1972.
[33] J. H. Reif and S. R. Tate, *Fast Spatial Decomposition and Closest Pair Computation for Limited Precision Input*, Technical Report N-96-001, Department of Computer Science, University of North Texas, Denton, TX, 1996; also available online from http://www.cs.duke.edu/∼reif/paper/Sep.ps.
[34] J. H. Reif and S. R. Tate, *N-Body Simulation* I: *Fast Algorithms for Potential Field Evaluation and Trummer's Problem*, Technical Report N-96-002, Department of Computer Science, University of North Texas, Denton, TX, 1996; also available online from http://www.cs.duke.edu/∼reif/paper/Multipole.ps.
[35] V. Rokhlin, *A fast algorithm for the discrete Laplace transformation*, J. Complexity, 4 (1988), pp. 12–32.
[36] C. Runge and H. König, *Die Grundlehren der mathematischen Wissenschaften*, 11, Springer, Berlin, 1924.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2090–2102

OPTIMAL SEARCH IN TREES*

YOSI BEN-ASHER†, EITAN FARCHI‡, AND ILAN NEWMAN†

Abstract. It is well known that the optimal solution for searching in a finite total order set is binary search. In binary search we divide the set into two "halves" by querying the middle element and continue the search on the suitable half. What is the equivalent of binary search when the set P is partially ordered? A query in this case is to a point $x \in P$, with two possible answers: "yes" indicates that the required element is "below" x or "no" if the element is not below x. We show that the problem of computing an optimal strategy for search in posets that are tree-like (or forests) is polynomial in the size of the tree and requires at most $O(n^4 \log^3 n)$ steps.

Optimal solutions of such search problems are often needed in program testing and debugging, where a given program is represented as a tree and a bug should be found using a minimal set of queries. This type of search is also applicable in searching classified large tree-like databases (e.g., the Internet).

Key words. optimal search, search in trees, binary search, search in graphs, posets

AMS subject classification. 68P10

PII. S009753979731858X

1. Introduction.

Binary search is a well-known technique used for searching in total order sets. Let $S = \{s_1, \ldots, s_n\}$ be a total ordered set such that $s_i < s_{i+1}$ for all $i \in \{1, \ldots, n-1\}$. A binary search usually is used to find out whether a given element s is a member of S. However, a different interpretation can be used, namely, that one of the elements in S is "buggy" and needs to be located. In a binary search, we query the middle element $s_{\frac{n}{2}}$. If the answer is "yes," then the bug is in $S' = \{s_1, \ldots s_{\frac{n}{2}}\}$ and we continue the search on S'; otherwise we search on the complement of S', namely $\{s_{\frac{n}{2}+1}, \ldots, s_n\}$. It is well known that the binary search is optimal for this case.

In this work we consider the generalization of searching in partially ordered sets (poset). A query is to a point x in the poset, with two possible answers: "yes" indicates that the required element is "below" x (less than or equal to x) or "no" if the element is not below x. Equivalently, P can be represented as a directed acyclic graph; G, a query is to a node $x \in G$; a "yes"/"no" answer indicates that the answer is at a node $y \in G$ reachable/not reachable from x, correspondingly. We are interested in the problem of computing the optimal search algorithm for a given input poset. The complexity measure here is the number of queries needed for the worst-case input (buggy node). We concentrate on posets that are tree- (forest-) like (every element except one has one element that covers it). In this case, the partial order set is represented as a rooted tree T, whose nodes are the elements of S. A query can be made to any node $u \in T$. A "yes"/"no" answer indicates whether the buggy node is in the subtree rooted at u, or in its complement. Our results extend naturally to forest-like posets, too. Some comments are made for Cartesian product posets.

An example of a (optimal) search on a tree of 5 nodes is shown in Figure 1.1. The arrow points to the next query node. The search takes 3 queries in the worst case.

*Received by the editors March 24, 1997; accepted for publication (in revised form) September 30, 1997; published electronically June 23, 1999. A preliminary version of this paper appeared in SODA97.

http://www.siam.org/journals/sicomp/28-6/31858.html

†Department of Computer Science, Haifa University, Haifa 31905, Israel (yosi@mathcs.haifa.ac.il, ilan@mathcs.haifa.ac.il).

‡I.B.M. Research Center, Haifa 31905, Israel.

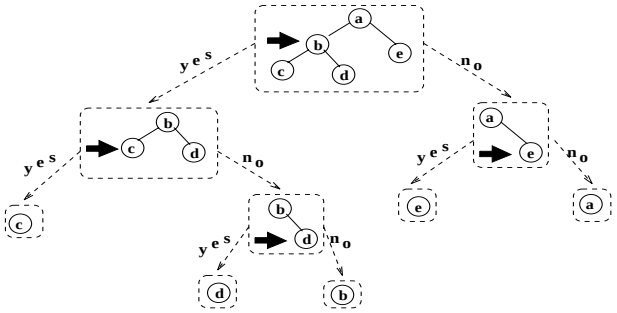


FIG. 1.1. *Searching in a tree.*

One motivation for the study of searches in trees (and posets in general) is that it forms a generalization of the search in linear orders to more complex sets: The known binary search is optimal for the path of length n with $\lceil \log n \rceil$ cost. Another extreme example is the search in a completely nonordered set of size n. This is equivalent to search in a star with n leaves. Here, as we must query each leaf separately, the search takes n queries. Searching in trees spans a spectrum between these two examples in terms of the cost function as well as poset type.

There is also a practical motivation: consider the situation where a large tree-like data structure is being transferred between two agents. Such a situation occurs when a file system (database) is sent across a network or a back/restore operation is done. In such cases, it is easy to verify the total data in each subtree by checksum-like tests (or randomized communication complexity equality testing). Such equality tests easily detect faults but give no information on which node of the tree is corrupted. Using a search on the tree (by querying the correctness of the subtrees) allows us to find the buggy node and avoid retransmitting the whole data structure.

Software testing is another motivation for studying search problems in posets (and in particular in trees). In general, program testing can be viewed as a two-person game, namely the tester and his "adversary." The adversary injects a fault into the program and the tester has to find the fault while minimizing the number of tests. A typical scenario in software testing is that the user tests his program by finding a "test bucket" (a set of inputs) that meets a certain coverage criteria, e.g., branch coverage or statement coverage [1, 2, 3]. It is plausible that in certain situations it might be possible to embed such a set of tests (e.g., the union over all test buckets that meet branch coverage) in a poset or in a tree such that the requirement for covering all tests can be replaced by a requirement for searching in this poset or tree. Finding an optimal search can save a lot of tests as the cost of a search might be considerably smaller than the size of the domain. For example, the syntactic structure of a program forms a tree; thus, if suitable tests are available, statement coverage might be replaced by a search in the syntactic tree of the program.

Finally, a possible motivation and direct application is in the area of information retrieval: Consider a Yahoo-like search scenario. The Yahoo contains an immense tree that classifies home pages (currently estimated as about 1–2% of the total number of web homepages). In a typical search, a node is reached and it exposes the next level of the tree (or part of it). The user chooses the appropriate branch according to the query he has in mind. As it turns out, this tree is quite deep, which often results in numerous queries before reaching the target. Clearly, such a top-down search might

be inefficient compared to the possibility of the optimal search of the Yahoo tree (e.g., searching in a chain of n nodes (a tree of depth n) requires n queries if we search top down and only $\log n$ queries if we allow to query arbitrary nodes).

A different notion of searching in posets was considered by Linial and Saks [5, 4]. They consider the case of a set of real numbers that are "stored" in a poset so that their natural order is consistent with the poset order. A search in this case is to determine if a real x is stored in the set. The possible queries in this case are, as in our case, the poset elements. The two possible answers for a query z is either "yes" (means $x \le e(z)$ where $e(z)$ is the element stored in z), or "no" ($x > e(z)$). The first answer results in excluding all elements greater than z from the poset and the later excludes all elements below z. Note that the difference between the two models is the resulting poset after a "yes" answer. It turns out that in spite of the similarity in definition the two models are quite different, e.g., the product of two paths (see section 5). Linial and Saks proved lower and upper bounds for the number of queries needed to search in posets in terms of some of the poset properties; however, they presented no algorithm to find the exact cost.

Our main result is a polynomial time algorithm (in the size of the tree) that finds the *optimal* search strategy for any tree (forest). Let $T(v)$ be the subtree of T rooted at v. The answer to a query $v \in T$ results in a search on either the subtree rooted at v or its complement $T - T(v)$. Thus, the optimal complexity, $w(T)$, of the search for T is defined by minimizing over all $v \in T$ the expression $1 + \max\{w(T(v)), w(T - T(v))\}$. Direct use of this formula to compute $w(T)$ would give an exponential time algorithm. Another possible approach is trying to compute $w(T)$ in a bottom-up manner; however, it seems that this, too, needs exponential time. Knowing the cost of the subtrees is not enough to compute the cost of the complete tree: a complement subtree produced by querying a node results in a new subtree whose cost has to be computed all over. Our approach is to get rid of this difficulty by using further relevant information on subtrees (rather than just the cost of the optimal search for them). We use a somewhat nonstandard decomposition of a tree into subtrees, so that the cost of a tree can be determined using the relevant information of its subtrees. Next, we generalize the results for forests and draw some conclusions for Cartesian product posets.

We note that for bounded degree trees, an approximation of the optimal strategy may be obtained by finding a "splitting" vertex that splits the tree into two parts that are not too big. However, such an approach totally fails for unbounded degree trees.

2. Basic definitions and preliminary facts. Let us start with some notations: The subtree of T that is rooted at u is denoted by $T(u)$. When it is clear from the context, we identify $T(u)$ with u. Deleting all nodes of a subtree T_1 from a tree T is denoted by $T - T_1$, in particular $T - u = T - T(u)$.

DEFINITION 2.1. *A search algorithm Q_T for a tree T with root r is defined recursively as follows. If $|T| = 1$, the search is trivial and gives as output the only node in T. For $|T| \ge 2$ a search algorithm is a triplet (v, Q_v, Q_{T-v}), where $v \in T - r$ (a "first query"), Q_v is a search algorithm for $T(v)$ (corresponds to a "yes" answer), and Q_{T-v} is a search algorithm for $T - v$ ("no" answer).*

Graphically, we denote Q_T as

$$Q_T = \begin{array}{ccc} v & \xrightarrow{\text{no}} & Q_{T-v}. \\ \downarrow & & \\ Q_v & & \end{array}$$

The cost of a search algorithm Q, denoted by $w(Q)$, is the number of queries needed to find any buggy node in the worst case. The cost of an optimal search algorithm for T, $w(T)$, is

$$w(T) = \min_{Q_T} w(Q_T).$$

An optimal algorithm is any search algorithm Q_T such that $w(Q_T) = w(T)$. Note that the above definition conforms with the convention that there is always a buggy node; thus the cost of a single node is zero since this node must be buggy and no query is needed. The case of a search in which there is a possible "un-found" answer is easily obtained from the above definition (with the expense of one additional query). See section 5 for more details.

The following is immediate from the definitions.

FACT 2.1. *For a tree T, with $|T| > 1$, $w(T) = 1 + \min_{v \in T} \max\{w(v), w(T-v)\}$.*

A useful property of the cost is that it is monotone.

OBSERVATION 2.1. *Let T_1 be a subtree of T_2 (namely, T_1 is obtained from T_2 by deleting some nodes), then $w(T_1) \leq w(T_2)$.*

Fact 2.1 suggests that we might as well start the search by querying a node $u \in T$ for which $w(u) < w(T)$ and $w(T-u)$ is minimal. Applying this idea further leads us to the definition of the "sequence of complements." We start by defining a lexicographic order on finite sequences.

DEFINITION 2.2. *Let $\mu = [\mu_1, \mu_2, \ldots, \mu_k]$ and $\rho = [\rho_1, \rho_2, \ldots, \rho_n]$ be two sequences of natural numbers. Then μ is lexicographically smaller than ρ ($\mu <_L \rho$) if there exists i such that $\mu_i < \rho_i$ and for every $j < i$, $\mu_j = \rho_j$.*

DEFINITION 2.3. *The sequence of complements $\mu(T) = [\mu_0, \mu_1, \mu_2, \ldots, \mu_l]$ for a given tree T is defined recursively as follows:*

$\mu_0 = w(T)$.

If $|T| = 1$, then $l = 0$ and $\mu(T) = [0]$.

For $|T| > 1$,

$$[\mu_1, \mu_2, \ldots, \mu_l] = \min \{\mu(T-u) |\ w(T(u)) < w(T)\},$$

where min *is taken according to the lexicographic order.*

For example, $\mu(L)$ of a path L with five nodes is $[3, 0]$, as its cost is 3; and the node u that achieves the smallest $\mu(L-u)$ is the son of the root for which $w(T-u) = 0$. The reader can verify that μ of a star with n leaves is $[n, n-1, \ldots, 2, 1, 0]$. The following properties are immediate from the definition: Let $\mu(T) = [\mu_0, \mu_1, \ldots, \mu_l]$. Then

1. $\mu_0 > \mu_1 > \cdots > \mu_l = 0$.
2. $l \leq w(T)$.

As the sequence of complements is always strictly decreasing, we restrict the order L to all *strictly decreasing* finite sequences of nonnegative integers with the last element being 0. The order L is a *discrete total* order on this set of sequences. In particular, every sequence has a rank in this total order, and every sequence has a successor. Thus, it also follows that $\mu(T)$ is a unique, well-defined sequence for every tree T.

DEFINITION 2.4. *For T with $|T| \geq 2$, $\lambda_T \in T$ is any node in T for which $\mu(T-\lambda_T)$ is the smallest and $w(T(\lambda_T)) < w(T)$. Namely λ_T is such that $\mu(T) = [w(T), \mu(T-\lambda_T)]$. Note that for $|T| = 1$, λ_T is not defined.*

It is immediate from the definition that λ_T is an optimal node, i.e., there exists an optimal search algorithm whose first query is λ_T. Figure 2.1 illustrates λ_T, $\mu(T)$, and an optimal search algorithm that starts in λ_T for a particular example. In the

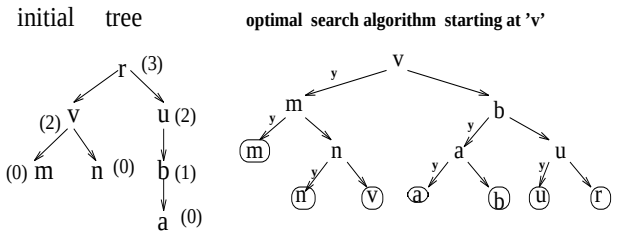


FIG. 2.1.

above tree (left side) the cost of each subtree is marked by its root. Here $\lambda_T = v$, since $\mu(T - v) = [2, 1, 0]$ compared (for example) to $\mu(T - u) = [3, 0]$.

3. The decomposition of T and the role of $\mu(T)$. As mentioned before, it is not obvious how to efficiently compute the cost of T using the formula of Fact 2.1 in a bottom-up manner. Our solution is to decompose T into subtrees, so that we can characterize $\mu(T)$ as a function of the μ's of its subtrees. We define the following operations:

T': "Rerooting" operation. T' is the tree obtained from T by adding a new node x whose only child is the root of T.

$T'_1 + \cdots + T'_k$: "grouping" operation. $T = T'_1 + T'_2 + \cdots + T'_k$ (also denoted by $\sum_{i=1}^{k} T'_i$) is obtained from disjoint trees $T_1, T_2, \ldots, T_k$ by attaching $T_1, T_2, \ldots, T_k$ as the children of a new root x. Note that we write it as an operation on the *rerooted* trees $T'_1, \ldots, T'_k$ rather than on $T_1, \ldots, T_k$, as we will relate $\mu(T)$ to $\mu(T'_1), \ldots, \mu(T'_k)$. In the grouping operation, we will always order $T_1, T_2, \ldots, T_k$ so that $\mu(T'_1) \leq_L \cdots \leq_L \mu(T'_k)$.

FACT 3.1. *Every rooted tree can be constructed from single nodes using the above operations.*

In the following two sections we will show how $\mu(T)$ is determined by the μ's of the subtrees used to decompose T according to the operations defined above.

3.1. Properties of the rerooting operation. We relate here the sequence of complements of any tree T to the sequence of complements of the tree T' that is obtained from T by rerooting. As it will turn out this can simply be characterized by the following theorem.

THEOREM 3.1. *Let T be a rooted tree and let T' be the tree obtained from T by rerooting. Then $\mu(T') = succ(\mu(T))$, where succ is the successor according to the order L on strictly decreasing nonnegative finite sequences, ending with* 0.

The key ingredient in the proof of the theorem is based on the observation that a "jump" in the coordinates of the sequence $\mu(T)$ is a necessary and sufficient condition for the cost of T' to remain the same. The term "jump" indicates that the difference between two consecutive coordinates of $\mu(T)$ is greater than 1. The intuition is that such a jump can compensate for the expected increase in $w(T')$ caused by the extra node added above the root of T. To demonstrate this on an example, consider a path P with three nodes, for which $\mu(P) = [2, 0]$; namely, it contains a jump. Indeed, P' is a path with 4 nodes and its cost remains 2. The reason why $w(P')$ did not increase is that we can start a search in P' by querying λ_P. For a "yes" answer we proceed as we would have done in P and the cost certainly won't change. For a "no" answer we are left with $P' - \lambda_P = (P - \lambda_P)'$ but $(P - \lambda_P)$ is a single node with zero cost. Thus $w(P' - \lambda_P) = 1$ which increased comparing to $P - \lambda_P$ but not enough to increase

$w(P')$. Another example is the rerooting T' of the tree T in Figure 2.1, showing that the cost does increase if $\mu(T)$ contains no jump. For T of Figure 2.1 $\lambda_T = v$ and $\mu(T) = [3, 2, 1, 0]$. Indeed, the reader can verify the cost increases, and $w(T') = 4$.

We proceed now with the proof of Theorem 3.1.

PROPOSITION 3.2. *For a given tree T, $w(T) \leq w(T') \leq w(T) + 1$*

Proof. The left inequality is immediate from Observation 2.1. The right inequality follows from the search that starts by querying the (old) root of T. □

PROPOSITION 3.3. *If $w(T') = w(T) + 1$, then $\lambda_{T'} = root(T)$ and $\mu(T') = [w(T'), 0]$.*

Proof. Querying $v = root(T)$ leads to $w(T' - v) = 0$ and thus $\mu(T') \geq [w(T'), 0]$, but $[w(T'), 0]$ is the smallest sequence $[\mu_0, \ldots, \mu_l]$ (according to $<_L$) for which $\mu_0 = w(T')$. □

LEMMA 3.4. *Let $\mu(T) = [\mu_0, \ldots, \mu_l]$. If there exists $i \geq 0$ such that $\mu_i < w(T) - i$, then $w(T') = w(T)$.*

Proof. Note that the condition of the lemma states that there is a jump between μ_i and μ_{i-1}. Let $\alpha = w(T)$. We will use induction on $|T|$ to show that there exists a search algorithm Q for T', with $w(Q) = \alpha$. The premise of the lemma is possible only for a tree with cost greater than 1; hence the induction base includes only a path P with three nodes or a star with two leaves. For the latter, the sequence of complements $\mu()$ is $[2, 1, 0]$ and the premise of the lemma is not met. For the path P, the sequence of complements is $[2, 0]$, and indeed $w(P') = w(P) = 2$.

Let T be a tree with $w(T) = \alpha \geq 3$, and consider the following search algorithm for T':

$$Q = \begin{array}{ccc} \lambda_T & \xrightarrow{\text{no}} & Q_1, \\ \downarrow & & \\ Q_2 & & \end{array}$$

where Q_1 is an optimal search algorithm for $T' - \lambda_T$ and Q_2 is an optimal search algorithm for $T(\lambda_T)$. Here, $w(Q) \leq 1 + \max(w(Q_2), w(T' - \lambda_T))$, but, by the definition of λ_T, $w(Q_2) \leq \alpha - 1$ and thus $w(Q) \leq \max(\alpha, 1 + w(T' - \lambda_T)) = \max(\alpha, 1 + w((T - \lambda_T)'))$. Hence, it is enough to show that $w((T - \lambda_T)') \leq \alpha - 1$.

Let us first consider the case where the jump is for $i = 1$, namely, $\mu_1 = w(T - \lambda_T) \leq \alpha - 2$. By Proposition 3.3 for $T - \lambda_T$ we have that $w((T - \lambda_T)') \leq w(T - \lambda_T) + 1 \leq \mu_1 + 1 \leq \alpha - 1$.

Consider the case where $i > 1$. Let $T_1 = T - \lambda_T$; then by the definition of μ we get that $\mu(T_1) = [\mu_1, \ldots, \mu_l]$, with $\mu_1 = w(T - \lambda_T) \leq \alpha - 1$. Thus, the premise of the lemma holds for T_1 (with $i' = i - 1$), so by the induction hypothesis $w(T_1') = w(T_1) \leq \alpha - 1$. □

The opposite of Lemma 3.4 is also true.

LEMMA 3.5. *Let $\mu(T) = [\mu_0, \ldots, \mu_l]$. If $w(T') = w(T)$, then there exists $i \in \{1, \ldots, l\}$ such that $\mu_i < w(T) - i$.*

Proof. Let $w(T) = w$. It is sufficient to show that if the condition is not met, i.e., $\forall i,\ \mu_i \geq w - i$, then $w(T') = w + 1$. We prove this by induction on $|T|$. The base case is a single node whose $\mu = [0]$ and for which T' is a path of length 1 with cost $= 1$.

For $|T| > 1$ the assumption above implies that $\mu(T) = [w, w - 1, w - 2, \ldots, 1, 0]$. Let Q be an optimal algorithm for T' starting with a vertex x. We show that Q must spend $w + 1$ queries. If $w(T(x)) \geq w$, then on a "yes" answer we are left with $T(x)$ (on which additional w queries are needed) and we are done. Thus we may assume that

$w(T(x)) \leq w-1$. Therefore by the definition of λ_T we get that $\mu(T-x) \geq_L \mu(T-\lambda_T)$. But this means that $\mu(T-x) = \mu(T-\lambda_T) = [w-1, w-2, \ldots, 1, 0]$ as the successor of $\mu(T-\lambda_T)$ has a first coordinate equal to w. Thus, by the induction hypothesis on $T-x$ (whose μ has no jump), $w(T'-x) \geq w(T-x)+1 = w-1+1 = w$. Hence $w(Q) \geq 1 + w(T'-x) = 1+w$. ∎

The above two lemmas yield the following theorem.

THEOREM 3.6. *Let $\mu(T) = [\mu_0, \ldots, \mu_l]$. Then $w(T') = w(T)$, iff there exists $i \geq 0$ such that $\mu_i < w(T) - i$.*

We now conclude with the proof of Theorem 3.1.

Proof. If for a tree T there is no jump, namely, $\mu(T) = [w, w-1, \ldots, 1, 0]$, where $w = w(T)$, then by Theorem 3.6 $w(T') = w+1$. Moreover, by Proposition 3.3, $\mu(T') = [w+1, 0]$, which is the successor of $[w, w-1, \ldots, 1, 0]$ in the order L.

Assume then that there is a jump in $\mu(T)$, and let i be the largest index for which $\mu_{i-1} \geq l-i+2$, namely,

$$\mu(T) = [\mu_0, \mu_1, \ldots, \mu_{i-1}, \overbrace{l-i}^{\mu_i}, l-i-1, \ldots, 2, 1, 0].$$

Then we prove by induction on $w(T)$ that $\mu(T') = succ(\mu(T))$ and that $\lambda_{T'} = \lambda_T$. By Theorem 3.6 $w(T') = w(T) = \mu_0$. Assume $\lambda_{T'} = x$ then $\mu(T') = [\mu_0, \mu(T'-x)]$. But $T'-x = (T-x)'$ and, by the induction hypothesis for $(T-x)'$, $\mu(T-x)' = succ(\mu(T-x))$. However, by the definition of λ_T,

$$\mu(T-x) \geq \mu(T-\lambda_T) = [\mu_1, \ldots, \mu_{i-1}, l-1, l-i-1, \ldots, 1, 0].$$

However,

$$succ([\mu_1, \ldots, \mu_{i-1}, l-i, l-i-1, \ldots, 1, 0]) = [\mu_1, \ldots, l-i+1, 0],$$

thus we get that $\mu(T'-x) = succ(\mu(T-x) \geq [\mu_1, \ldots, l-i+1, 0]$. On the other hand, by the definition of the node x in T', we have that $\mu(T'-x) \leq \mu(T'-\lambda_T) = \mu((T-\lambda_T)') = succ(\mu(T-\lambda_T)) = [\mu_1, \ldots, l-i+1, 0]$. Thus we conclude that

$$\mu(T') = [\mu_0, \mu_1, \ldots, l-i+1, 0] = succ([\mu_0, \mu_1, \ldots, \mu_{i-1}, l-i, l-i-1, \ldots, 1, 0]).$$

Moreover, as we get that $\mu(T'-x) = \mu(T'-\lambda_T)$ this also proves that $\lambda_T = \lambda_{T'}$. ∎

3.2. Properties of the grouping operation. We now relate the sequence of complements of grouping k rerooted trees $T_1'+T_2'+\cdots+T_k'$ to the individual sequences. First a simpler operation is defined as follows.

DEFINITION 3.7. *The "amalgamation" operation $T_1' + T_2$ on disjoint trees T_1', T_2 is the tree that is obtained by amalgamating the roots of T_1' and T_2 into one node.*

Note that in the amalgamation operation it is required that one of the trees be a rerooted tree. The amalgamation operation exhibits a certain monotonicity.

LEMMA 3.8. *Let T_1, T_2, T be trees; then $\mu(T_1') \leq_L \mu(T_2') \implies w(T_1'+T) \leq w(T_2'+T)$.*

Proof. This proof is by induction on the sum $n = |T| + k$ where $|T|$ is the number of nodes in T and k is the rank of $\mu(T_1')$ defined by the total order $<_L$.

The induction base is for $n = 2$, and T that has one node. Since the amalgamation of one node to any tree does not change the tree at all, the base of the induction is trivially true.

First case—$w(T) \le w(T_1')$: Let $w = w(T_1')$ and assume Q is any search algorithm for $T+T_2'$ with cost $w(Q)$. We give an algorithm for $T+T_1'$ with cost of at most $w(Q)$. Indeed let $u \neq root(T_2')$ be the first query of Q. If $u \in T$ then obviously $w(Q) \ge w+1$, as $w((T-u)+T_2') \ge w(T_2) \ge w$. However, the algorithm that first queries the root of T_1 and then proceeds optimally for T_1 if the answer is "yes" and optimally for T if the answer is "no" also achieves $w+1$ and we are done.

If $u \in T_2$, then we may assume that $w(T(u)) \le w-1$ (otherwise $w(Q) \ge w+1$ and we are done as before). We also may assume that $w(T_2') = w$, as if $w(T_2') \ge w+1$, then $w(Q) \ge w+1$, but as we have shown, we can clearly search $T+T'$ in $w+1$ queries. Thus $\mu(T_2' - \lambda_{T_2}) \ge \mu(T_1' - \lambda_{T_1})$ (as $\mu(T_2') \ge \mu(T_1')$ and the first coordinate in both sequences is equal to w). In this case by querying λ_{T_1} we get that $w(T+T_1') \le 1 + \max(w-1, w(T + (T_1 - \lambda_{T_1})'))$. However, by our assumption $\mu(T_2' - u) \ge_L \mu(T_2' - \lambda_{T_2'}) \ge_L \mu(T_1' - \lambda_{T_1'})$. Thus by induction $w(T+(T_2'-u)) \ge w(T+(T_1' - \lambda_{T_1'}))$, so that $w(Q) \ge 1 + w(T + T_2' - u) \ge 1 + w(T + T_1' - \lambda_{T_1})$. On the other hand, as $w(Q) \ge w$, it follows that $w(Q) \ge 1 + \max(w-1, w(T + (T_1' - \lambda_{T_1'})) = w(T+T_1')$.

Second case—$w(T) > w(T_1')$: Consider an optimal search algorithm Q for $T+T_2'$. We may assume that $w(Q) \le w(T)$; otherwise, a search algorithm for $T+T_1'$ that starts by querying the root of T_1 will use at most $w(T)+1$ queries (as $w(T) > w(T_1)$), so that $w(T+T_1') \le w(T+T_2')$. The lemma follows.

Let y be the first query of Q. It follows that $y \in T$. Consider a search algorithm Q' for $T'+T_1$ that also starts by querying y. We are left with two trees $(T-y)+T_1'$ and $(T-y)+T_2'$. Here, $|T-y| < |T|$ so that we can use the induction assumption and obtain that $w((T-y)+T_1') \le w((T-y)+T_2')$. The lemma follows. □

We now consider grouping of several trees.

PROPOSITION 3.9. *Let $T = \sum_{i=1}^{k} T_i'$ (i.e., $T = T_1' + \cdots + T_k'$) and assume that $\mu(T_1') \le \cdots \le \mu(T_k')$. Then $w(T_k') \le w(T) \le w(T_k') + k - 1$.*

Proof. The left inequality is immediate from Fact 3.1. The other inequality is obvious by the following search algorithm for T.

Start by querying $root(T_1), root(T_2), \ldots, root(T_{k-1})$. If the answer is "yes" on, say, the ith query, then proceed by an optimal search on T_i. Otherwise proceed by an optimal search in T_k. Clearly this search takes at most $w(T_k') + k - 1$ queries. □

The criterion for determining the exact cost of the grouping operation is given by the following lemma.

LEMMA 3.10. *Let $T = T_1' + \cdots + T_k'$. Then*

(a) $w(T) = w(T_k')$ *iff* $w(T_1' + \cdots + T_{k-1}' + (T_k' - \lambda_{T_k'})) \le w(T_k') - 1$;

(b) *for any $\alpha > w(T_k')$, we have that $w(T) \le \alpha$ iff $w(T_1' + \cdots + T_{k-1}') \le \alpha - 1$.*

Proof. The "if" direction for (a) is trivial since, if $w(T_1' + \cdots + T_{k-1}' + (T_k' - \lambda_{T_k'})) \le w(T_k')$, then by querying $\lambda_{T_k'}$ we are done. Similarly querying $root(T_k)$ if $w(T_1' + \cdots + T_{k-1}') \le \alpha - 1$ proves the "if" direction for (b).

For the other direction of (a)—assume by negation that $w(T_1' + \cdots + T_{k-1}' + (T_k' - \lambda_{T_k'})) \ge w = w(T_k')$—we show that $w(T_1' + \cdots + T_k') \ge w+1$. Let Q be a search algorithm for $T_1' + \cdots + T_k'$ that starts by querying x. If $x \notin T_k'$, then clearly $w(Q) \ge 1+w$. Assume that $x \in T_k'$ and $w(T(x)) \le w-1$ (otherwise, for a "yes" answer $w(Q) \ge 1 + w(T(x)) \ge w+1$). Then, by the definition of μ, $\mu(T_k' - x) \ge_L \mu(T_k' - \lambda_{T_k'})$.

Now, let $\tilde{T} = T_1' + \cdots + T_{k-1}'$. Then by Lemma 3.8 we get $w((T_k - x)' + \tilde{T}) \ge w((T_k - \lambda_{T_k'})' + \tilde{T}) \ge w$; hence

$$w(Q) = 1 + w((T_k - x)' + \tilde{T})) \ge 1 + w((T_k - \lambda_{T_k'})' + \tilde{T}) \ge 1 + w.$$

For (b), let $\alpha > w(T'_k)$ and assume that $w(T'_1 + \cdots + T'_{k-1}) \geq \alpha$. Let Q be a search algorithm for T that first queries x. If $x \in T'_k$, then $w(Q) \geq 1 + w(T'_1 + \cdots + T'_{k-1} + T'_k - \lambda_{T'_k}) \geq 1 + w(T'_1 + \cdots + T'_{k-1}) \geq \alpha + 1$. If $x \in T'_i$, where $i \neq k$, then let $\tilde{T} = T'_1 + \cdots + T'_{i-1} + T'_{i+1} + \cdots + T'_{k-1}$. By Lemma 3.8 $w(T'_1 + \cdots + T'_{k-1}) = w(\tilde{T} + T'_i) \leq w(\tilde{T} + T'_k)$. Considering the "no" answer after query x, this implies that $w(Q) \geq 1 + w(\tilde{T} + T'_k) \geq 1 + w(\tilde{T} + T'_i) \geq 1 + \alpha$. ∎

Finally we relate the sequences of complements.

LEMMA 3.11. *Let $T = T'_1 + \cdots + T'_k$. Then*

(a) *if $w(T) = w(T'_k)$, then $\lambda_T = \lambda_{T'_k}$, $\mu(T) = [\mu_0, \ldots, \mu_l]$, where $\mu_0 = w(T'_k)$ and $[\mu_1, \ldots, \mu_l] = \mu(T'_1 + \cdots + T'_{k-1} + (T'_k - \lambda_{T'_k}))$;*

(b) *if $w(T) > w(T'_k)$, then $\lambda_T = root(T'_k)$, $\mu(T) = [\mu_0, \ldots, \mu_l]$, where $\mu_0 = w(T)$ and $[\mu_1, \ldots, \mu_l] = \mu(T'_1 + \cdots + T'_{k-1})$.*

Proof. The proof immediately follows from the proof of Lemma 3.10 ∎

4. Constructing the optimal search algorithm. In this section we show that computing $\mu(T)$ (and in fact λ_T) for a given tree T, can be done in polynomial time (in the size of T). The algorithm constructs $\mu(T), \lambda_T$ bottom up using the rerooting and grouping operations.

We need the following notation: for a given tree T, let $\mu(T) = [\mu_0, \mu_1, \ldots, \mu_l]$; we denote $\partial\mu = [\mu_1, \ldots, \mu_l] = \mu(T - \lambda_T)$. Note that $\partial\mu$ is computed from μ just by dropping out the first entry. (Although formally it depends on λ_T, one does not need to know it explicitly.) Depending on the current operation in the decomposition, the algorithm performs the following computations.

Rerooting operation: Let $T = T'_1$.

1. Compute $\mu(T) = succ(\mu(T_1))$ (Theorem 3.1).
2. If $w(T) = w(T_1) + 1$, then $\lambda_T = root(T_1)$; otherwise (if $w(T) = w(T_1)$), $\lambda_T = \lambda_{T_1}$.

   This can be done in $O(w(T))$ steps.

Grouping operation: Let $T = T'_1 + \cdots + T'_k$.

1. The μ sequences of $T'_1, \cdots, T'_k$ are sorted so that $\mu(T'_1) \leq_L \cdots \leq_L \mu(T'_k)$. Each sequence is of length at most $w(T)$, and each entry is bounded by $w(T)$. Thus sorting can be done (treating each sequence as a number) by $min(w^2(T) \log w(T), |T| \log |T|)$.
2. For any α it can be determined whether $w(T'_1 + \cdots + T'_k) \leq \alpha$. This is done using the following recursive test as suggested by Lemma 3.10. The test is applied to the vectors of complement of subtrees $T'_1, \ldots, T'_k$, such that $test(\mu(T'_1), \ldots, \mu(T'_k), \alpha)$ returns true iff $w(T'_1 + \cdots + T'_k) \leq \alpha$.

   a) For $\alpha \geq \mu_0(T'_k) + k$

$$test(\mu(T'_1), \ldots, \mu(T'_k), \alpha) = true.$$

   b) For $w \leq \mu_0(T'_k) - 1$

$$test(\mu(T'_1), \ldots, \mu(T'_k), \alpha) = false.$$

   c) For $\alpha = \mu_0(T'_k)$

$$test(\mu(T'_1), \ldots, \mu(T'_k), \alpha) = test(\mu(T'_1), \ldots, \mu(T'_{k-1}), \partial\mu(T'_k), \alpha - 1),$$

$$\text{i.e., } test(T'_1 + \cdots + T'_k, \alpha) = test(T'_1 + \cdots + T'_{k-1} + T'_k - \lambda_{T'_k}, \alpha - 1).$$

d) For $\mu_0(T'_k) < w \leq \mu_0(T'_k) + k - 1$

$$test(\mu(T'_1), \ldots, \mu(T'_k), \alpha) = test(\mu(T'_1), \ldots, \mu(T'_{k-1}), \alpha - 1),$$

i.e., $test(T'_1 + \cdots + T'_k, \alpha) = test(T'_1 + \cdots + T'_{k-1}, \alpha - 1)$.

The validity of a) and b) is due to Proposition 3.9, and that of c) and d) is due to Lemma 3.11.

After every application of test it might be required to sort the remaining sequences of complements. The worst case is if the first coordinate of $\mu(T'_k)$ is deleted for which the correct place of $\mu(T'_k)$ among the k sequences can be found in $w(T) \log w(T)$ comparisons. The time needed to compute $test(T'_1 + \cdots + T'_k, \alpha)$ is therefore $O(w^2(T) \log w(T))$.

We compute the cost $w(T'_1 + \cdots + T'_k)$ by binary searching for the correct α in the range $w(T'_k), \ldots, w(T'_k) + k - 1$. This may require $\log k$ applications of $test()$, Hence the maximal number of steps needed to compute the exact α is $O(w^2(T) \log^2 w(T))$ (not including the sorting of phase 1).

Note that the computation of $w(T'_1 + \cdots + T'_k)$ also determines λ_T at the same time (as implied by Lemma 3.11).

3. After determining the exact cost $\alpha = w(T) = \mu_0(T)$, we proceed to find the next component of $\mu(T)$ according to the two cases of Lemma 3.11.
(a) if $w(T) = w(T'_k)$, then $\lambda_T = \lambda_{T'_k}$ and $\mu(T)_1 = w(T'_1 + \cdots + T'_{k-1} + (T'_k - \lambda_{T'_k}))$;
(b) if $w(T) > w(T'_k)$, then $\lambda_T = root(T'_k)$ and $\mu(T)_1 = w(T'_1 + \cdots + T'_{k-1})$.
In this way we find each entry of $\mu(T)$. Note that resorting $\mu(T'_1), \ldots, \mu(T'_k - \lambda_{T'_k})$ is done in $w(T) \log w(T)$ steps; therefore the whole process takes $O(w^3(T) \log^3 w(T))$ steps.

THEOREM 4.1. *Let T be a tree with n nodes; then $\mu(T)$ and λ_T can be computed in $O(n \cdot w^3(T) \log w^3(T)) = O(n^4 \log^3 n)$ steps.*

Proof. We compute $\mu(T)$ by constructing T from subtrees working bottom up, starting from the leaves to the root. At each intermediate step we compute $\mu(T_1)$ for a subtree T_1 in $O(w^3(T) \log^3 w(T))$ steps using the above algorithm. Thus the whole process for a tree T with $|T| = n$ takes at most $O(n \cdot w^3(T) \log^3 w(T))$ steps. For bounded degree trees where $w(T) = O(\log n)$ this gives $O(n \log^4 n)$. For general trees this might be $O(n^4 \log^3 n)$. Note that each time we compute $w(T^i)$ of a subtree T^i, we also determine λ_{T^i}. Thus λ_T is computed in the same time bound, and an optimal search algorithm can be constructed. ∎

An example of the main stages of the algorithm is given in the appendix.

5. Search in forests and Cartesian products posets.

Forests. Our results hold for forests as well. Recall that in our model we assumed that one of the nodes in the given tree must be buggy. Let $\tilde{w}(T_1, T_2, \ldots, T_k)$ denote the cost of searching in a forest with k trees, where we do not assume that one of the nodes in $T_1, \ldots, T_k$ is buggy. It is easy to see that $\tilde{w}(T_1, T_2, \ldots, T_k) = w(T'_1 + T'_2 + \cdots + T'_k)$, where $T'_1 + T'_2 + \cdots + T'_k$ is the grouping operation defined in section 3. The reason is that if none of the nodes in the forest is buggy, then an optimal search algorithm will identify the root of $(T'_1 + T'_2 + \cdots + T'_k)$ as the buggy node. On the other hand $T'_1 + T'_2 + \cdots + T'_k$ can be searched by applying a search algorithm for the forest $T_1, T_2, \ldots, T_k$; if this search gives as a result "no buggy element," then the answer for $T'_1 + T'_2 + \cdots + T'_k$ is its root.

Rooted Cartesian products. A rooted poset is a poset with unique greatest element.

CLAIM 5.1. *Let P_1, P_2 be two rooted posets and let $P = P_1 \times P_2$ be the (rooted) product poset of P_1 and P_2. Then $w(P) \leq w(P_1) + w(P_2)$.*

Proof. One may search P by first querying (a, x), where a is the greatest element of P_1 and x is optimal first query for P_2. This results in either searching in $P_1 \times P_2(x)$ or $P_1 \times (P_2 - P_2(x))$, where $P_2(x)$ is the poset of all elements below x in P_2. Going on in this way by following the optimal strategy for P_2 for at most $w(P_2)$ queries we are left with a sub-poset that is isomorphic to P_1 for which additional $w(P_1)$ queries is enough.

As it is clear that $w(P) \geq max(w(P_1), w(P_2))$, the above observation gives $w(P)$ up to a factor of two.

An interesting example (the simplest nontrivial) of a rooted product is a product of two chains which is a rectangular lattice. □

CLAIM 5.2. *Let P_1, P_2 be disjoint chains and let $P = P_1 \times P_2$. Then $w(P_1) + w(P_2) - 2 \leq w(P) \leq w(P_1) + w(P_2)$.*

Proof. The upper bound follows from Claim 5.1. The lower bound follows from the fact that $w(L) \geq \lceil \log |P| \rceil$. If one or both lengths of P_1, P_2 is a power of 2, then in fact $\lceil \log |P| \rceil$ is the exact answer.

It is interesting to confront the above with the result of [5] for the Cartesian products of chains for their model. Let P_n be the n element chain. In both models $w(P_n) = \lceil \log n \rceil$. However, in the Linial–Saks model $w(P_n \times P_n) = 2n - 1$, while in our model $w(P_n \times P_n) = 2\lceil \log n \rceil$. □

6. Conclusions. We have presented a polynomial time algorithm for computing the optimal search strategy for forest-like Posets. The crux of the proof, and the algorithm, is the observation that the structure of a given tree T can be represented by a sequence of numbers $\mu(T)$. As an example, the cost of an optimal search of a complete binary tree (T) with n nodes is $\log n + \log^* n + \theta(1)$. We do not know any direct method for proving this result rather than computing $\mu(T)$ recursively (following our algorithm).[1]

Some problems are left open.

1. The main open problem is to give an algorithm that determines the cost (optimal strategy) for general posets (namely, searching in directed acyclic graphs).
2. The randomized complexity is quite interesting, too. For a given tree T and any buggy node, can one do better on the average using a randomized search algorithm? (A randomized search algorithm can be viewed as a probability distribution on a set of deterministic search algorithms.) For example, let T be a rooted star with n leaves; then it is easy to see that any randomized search algorithm will pay, on the average, n queries. The reason is that an adversary will put the "bug" in the root of the star, using the fact that any deterministic algorithm must query all the leaves first (this corresponds to the fact that the nondeterministic complexity of the star of n leaves is n). Randomization can help for some trees. For example, for d regular tree of height h, $w(T) = h \cdot d$ while its randomized complexity is $\leq \frac{d \cdot (h+1)}{2}$. Our conjuncture is that for every tree T, its randomized complexity is at least $\frac{1}{2} w(T)$.
3. While our algorithm gives the exact cost for any tree, it would be interesting to prove tight upper and lower bounds as a function of some natural property

[1] Further details are left to the reader.

of the tree. For example, $\log size(T)$ and d are lower bounds, where d is the maximal degree of T. However, none of these is tight in general. Another example is that for trees with maximum degree d, $\log_{\frac{d+1}{d}} size(T)$ is an upper bound, simply demonstrated by querying a node that splits the tree into two parts of size $\leq \frac{d}{d+1} size(T)$ each.

4. For the product of rooted posets we have seen that the cost is at most the sum of the costs. Is this tight (up to an additive $O(1)$ term) for every rooted product?

7. Appendix.

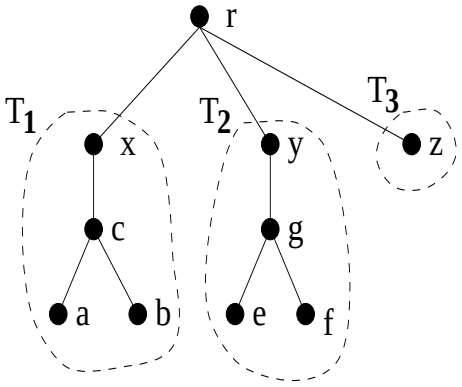


FIG. 7.1.

7.1. A detailed example of the algorithm. This section contains a detailed example showing the main phases of the algorithm combing three subtrees $T = T_3' + T_2' + T_1'$ as described in Figure 7.1. In early stages, the algorithms have computed $\mu(T_i'), \lambda_{T_i'},\ i = 1, 2, 3$, such that

$$\mu(T_1') = [3,1,0]\ \lambda_{T_1'} =' c',\quad \mu(T_2') = [3,1,0]\ \lambda_{T_2'} =' g',\quad \mu(T_3') = [1,0]\ \lambda_{T_3'} =' z'.$$

We proceed according to the three stages of the grouping operation in the algorithm to compute $\mu(T)$. After sorting the vectors of complements we obtain (the number of subtrees is $k = 3$) the possible cost of $w(T)$ is $3 \leq w(T) \leq 3 + 2 = 5$. Next, $test(T_3' + T_2' + T_1', \alpha)$ is applied for $\alpha = 4$ and is computed as follows (the label on the arrow $\xrightarrow{x)}$ denotes one of the four cases of $test()$ as described in section 4):

$$\begin{bmatrix} test(T_3' + T_2' + T_1', 4) \\ \mu(T_1') = [3,1,0] \\ \mu(T_2') = [3,1,0] \\ \mu(T_3') = [1,0] \end{bmatrix} \xrightarrow{d)} \begin{bmatrix} test(T_3' + T_2', 3) \\ \mu(T_2') = [3,1,0] \\ \mu(T_3') = [1,0] \end{bmatrix}$$

$$\xrightarrow{c)} \begin{bmatrix} test(T_3' + T_2' - \lambda_{T_2'}, 2) \\ \partial\mu(T_2') = [1,0] \\ \mu(T_3') = [1,0] \end{bmatrix} \xrightarrow{a)} TRUE.$$

We continue to check if $w = 3$ using

$$\begin{bmatrix} test(T_3' + T_2' + T_1', 3) \\ \mu(T_1') = [3,1,0] \\ \mu(T_2') = [3,1,0] \\ \mu(T_3') = [1,0] \end{bmatrix} \xrightarrow{c)} \begin{bmatrix} test(T_3' + T_2' + T_1' - \lambda_{T_1'}, 2) \\ \partial\mu(T_1') = [1,0] \\ \mu(T_2') = [3,1,0] \\ \mu(T_3') = [1,0] \end{bmatrix} \xrightarrow{b)} FALSE.$$

Hence, we have computed the first component in the vector of complements, namely, $\mu(T)_0 = 4$.

The second coordinate, $\mu(T)_1$, is obtained by computing $w(T'_3 + T'_2)$ since $\lambda_T = root(T_1)$ and the complement is $T'_3 + T'_2$. The possible range for $\mu(T)_1$ is $3 \leq \mu(T)_1 \leq 3 + 2 - 1 = 4$. However, "true" is obtained for the minimal value $\mu(T)_1 = 3$,

$$\begin{bmatrix} test(T'_3 + T'_2, 3) \\ \mu(T'_2) = [3, 1, 0] \\ \mu(T'_3) = [1, 0] \end{bmatrix} \xrightarrow{c)} \begin{bmatrix} test(T'_3 + T'_2 - \lambda_{T'_2}, 2) \\ \partial\mu(T'_2) = [1, 0] \\ \mu(T'_3) = [1, 0] \end{bmatrix} \xrightarrow{a)} TRUE,$$

and we get that $\mu(T)_1 = 3$. The third coordinate is computed in a similar way, namely, $1 \leq \mu(T)_2 \leq 1 + 2 - 1 = 2$, yet

$$\begin{bmatrix} test(T'_3 + T'_2 - \lambda_{T'_2}, 1) \\ \partial\mu(T'_2) = [1, 0] \\ \mu(T'_3) = [1, 0] \end{bmatrix} \xrightarrow{c)} \begin{bmatrix} test(T'_3, 0) \\ \mu(T'_3) = [1, 0] \end{bmatrix} \xrightarrow{b)} FALSE;$$

hence $\mu(T)_2 = 2$.

The set of nodes that were used in the different stages of $test(T'_3 + T'_2 + T'_1, 4)$ forms the backbone of the optimal search algorithm, i.e.,

$$\begin{array}{rccccccc} Q_T = & x & \xrightarrow{\text{no}} & g & \xrightarrow{\text{no}} & z & \xrightarrow{\text{no}} & y \xrightarrow{\text{no}} \quad r\ . \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ & Q_{T_1} & & Q_{T(g)} & & z & & y \end{array}$$

REFERENCES

[1] B. Beizer, *Software Testing Techniques*, Van Nostrand Reinhold, New York, 1990.

[2] P. G. Frankl and E. J. Weyker, *Provable improvements on branch testing*, IEEE Transaction on Software Engineering, 19 (1993), pp. 962–975.

[3] J. R. Horgan, S. London, and M. R. Lyu, *Achieving software quality with testing coverage measures*, IEEE Trans. Comput., 27 (1994), pp. 60–69.

[4] N. Linial and M. Saks, *Every poset has a central element*, J. Combin. Theory, 40 (1985), pp. 195–210.

[5] N. Linial and M. Saks, *Searching order structures*, J. Algorithms, 6 (1985), pp. 86–103.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2103–2116

WEAK RANDOM SOURCES, HITTING SETS, AND BPP SIMULATIONS*

ALEXANDER E. ANDREEV†, ANDREA E. F. CLEMENTI‡, JOSÉ D. P. ROLIM§, AND LUCA TREVISAN¶

Abstract. We show how to simulate any BPP algorithm in polynomial time by using a weak random source of r bits and min-entropy r^γ for any $\gamma > 0$. This follows from a more general result about *sampling* with weak random sources. Our result matches an information-theoretic lower bound and solves a question that has been open for some years. The previous best results were a polynomial time simulation of RP [M. Saks, A. Srinivasan, and S. Zhou, *Proc. 27th ACM Symp. on Theory of Computing*, 1995, pp. 479–488] and a quasi-polynomial time simulation of BPP [A. Ta-Shma, *Proc. 28th ACM Symp. on Theory of Computing*, 1996, pp. 276–285].

Departing significantly from previous related works, we do not use extractors; instead, we use the OR-disperser of Saks, Srinivasan, and Zhou in combination with a tricky use of hitting sets borrowed from [Andreev, Clementi, and Rolim, *J. ACM*, 45 (1998), pp. 179–213].

Key words. derandomization, imperfect sources of randomness, hitting sets, randomized computations, expander graphs

AMS subject classifications. 68Q10, 11K45

PII. S0097539797325636

1. Introduction.

Randomized algorithms are often the simplest ones that can be used to solve a given problem, or the most efficient, or both (see [MR95]). For some problems, including primality testing and approximation of #P-complete counting problems, only randomized solutions are known.

The practical applicability of such randomized methods depends on the effective possibility for an algorithm to access *truly random bits*. Since it is questionable whether truly random sources really exist, much research has been devoted in the last decade to finding weaker notions of randomness that are still sufficient to run BPP algorithms in polynomial time [VV85, SV86, V86, V87, CG88, Z90]. Several definitions of *weak random source* have been proposed in the literature, the most general being the following [CG88, Z90]: for $\gamma > 0$, an (r, r^γ)-source is a random source that outputs a string in $\{0,1\}^r$, and no string has probability of being output larger than 2^{-r^γ} (such an object is also called *random source of min-entropy* r^γ). An information-theoretic argument shows that a black-box simulation of BPP using an $(r, r^{o(1)})$-source is impossible when r is polynomial in the number of random bits used by the simulated algorithm.

Dispersers and extractors. The usual method of simulating a BPP algorithm using a weak random source is as follows. Say that, for a given input, the algorithm

*Received by the editors August 5, 1997; accepted for publication (in revised form) January 23, 1998; published electronically June 23, 1999. An extended abstract of this paper appears in the *Proceedings of the 38th IEEE Symp. on Foundations of Computer Science.*
http://www.siam.org/journals/sicomp/28-6/32563.html

†Advanced Development Laboratory, LSI LOGIC, Corp., 2091 Landings Drive, Mountain View, CA 94043 (andreev@lsil.com).

‡Dipartimento di Matematica, Università "Tor vergata" di Roma, Via della Ricerca Scientifica, I-00133 Roma, Italy (clementi@mat.uniroma2.it).

§Centre Universitaire d'Informatique, University of Geneva, 24 rue General Dufour, CH 1211, Geneve 4, Switzerland (rolim@cui.unige.ch).

¶Department of Computer Science, Columbia University, New York, NY 10027 (luca@cs.columbia.edu).

requires m (truly) random bits; then we ask the source r bits (note that only one access to the weak random is required), and we use them to produce a sample space (a set of m-bit strings). Such strings are fed into the algorithm, and then the majority rule is used to decide whether to accept or reject. The procedure that computes the sample space starting from the output of the source is *independent* of the algorithm that we want to derandomize. This simulation is basically equivalent [Z90, Z96, NZ96, SZ94, SSZ95, T96] to a bipartite graph $G = (V, W, E)$ having 2^r nodes in the left component V, 2^m nodes in the right component W, and degree d, and such that if we select a node v in the left component according to an (r, r^γ)-source and then a random neighbor of v, the induced distribution in W is ϵ-close to the uniform distribution over W. Such a graph is a $(2^r, 2^m, d, r^\gamma, \epsilon)$-*extractor*. The left nodes are seen as possible outcomes of the random source and the right nodes as possible random strings for the algorithm to be simulated. The simulation amounts to selecting a node on the left side according to the weak random source and then selecting as sample space the set of its neighbors. If, for some fixed γ, one could achieve d and r polynomial in m, then a polynomial time simulation of BPP would be possible, using an (r, r^γ)-source. However, the best present construction of extractors for fixed $\gamma > 0$ and $r = \text{poly}(m)$ has $d = n^{log^{(k)}n}$ [T96]. This implies a quasi-polynomial time simulation of BPP. A polynomial-time simulation of BPP, using weak random sources of min-entropy r^γ for any fixed $\gamma > 0$, was one of the major open questions in the field.

It is not difficult to show that, to simulate RP by means of a weak random source, *OR dispersers* [CW89] (from now on, we will simply call them *dispersers*) are sufficient. A $(2^r, 2^m, 2^{r^\gamma})$-disperser is again a bipartite graph $G = (V, W, E)$ with parameters r, m, and d as before, but now the property is that for any set $V' \subseteq V$ of at least 2^{r^γ} vertices on the left side and any set $W' \subseteq W$ of more than $2^m/2$ vertices on the right side, there is at least one edge joining V' and W'. This construction is somewhat easier to obtain, and indeed Saks, Srinivasan, and Zhou [SSZ95] give a disperser with $d = \text{poly}(n)$, for any constant $\gamma > 0$, allowing for a polynomial time simulation of RP.

See [N96] for a complete survey on extractors, dispersers, and weak random sources.

Pseudorandom generators and hitting sets. A more ambitious goal than simulating BPP with weak random sources is the *deterministic* simulation of BPP. Research on this subject tries to isolate reasonable complexity assumptions under which deterministic simulations of randomized algorithms are possible [Y82, BM84, N90, BFNW93, NW94, IW97, ACR97].

In some cases, combinatorial objects developed in the study of weak random sources have been used to give derandomization [NZ96]. Here we reverse this connection and use a derandomization method to take full advantage of a weak random source.

Two basic combinatorial objects are studied in the theory of derandomization: pseudorandom generators (whose efficient construction immediately implies a deterministic simulation of BPP) and hitting-set generators (whose efficient construction allows us to simulate RP algorithms). Informally speaking, in the context of derandomization, pseudorandom generators play the role of extractors and hitting-set generators play that of dispersers. A recent result of Andreev, Clementi, and Rolim [ACR98] shows how to deterministically simulate BPP algorithms by using hitting set generators. This suggests that perhaps dispersers could be used to simulate BPP with weak random sources.

A *quick δ-hitting set generator* (quick δ-HSG) is an algorithm that, given a pa-

rameter n, finds in poly(n) time a set $H_n \subseteq \{0,1\}^*$ such that, for any finite Boolean function f of circuit complexity n, if $\mathbf{Pr}_x[f(x) = 1] > \delta$, then $f(a) = 1$ for some $a \in H_n$,[1] where the probability is taken uniformly over $\{0,1\}^n$. The main technical result of [ACR98] can be stated as follows.

LEMMA 1 (see [ACR98]). *For any choice of constants $\epsilon, \delta > 0$, there is a deterministic algorithm that, given access to a quick δ-HSG and given in input any circuit C of size n, returns in* poly(n) *time a value D such that $|\mathbf{Pr}_x[C(x) = 1] - D| \leq \epsilon$, where the probability is taken uniformly over all possible $x \in \{0,1\}^n$.*

Lemma 1 immediately implies the following general derandomization result.

THEOREM 2. *If for some $\delta > 0$ a quick δ-HSG exists, then $P = BPP$.*

Andreev et al. [ACR98] prove Lemma 1 by constructing a set S of size poly(n) that is ϵ-discrepant for C, i.e., such that $\mathbf{Pr}_{x \in S}[C(x) = 1]$ approximates the value $\mathbf{Pr}_x[C(x) = 1]$ up to an additive error ϵ. A basic ingredient is the definition of a *discrepancy test* that, given a circuit C, a "candidate" set S, and a parameter ϵ, tests whether S is ϵ-discrepant for C. The test also needs an auxiliary set H in input, and, provided that H has a certain hitting property, the test is "sound;" that is, if the set S is accepted, then S is ϵ-discrepant for C. The fact that the test is sound only if the auxiliary set H is hitting is not a major restriction—since we are assuming that a hitting-set generator exists, we can use it to generate H. Thus, proving the theorem amounts to find a set S that passes the test. This task is solved in [ACR98] by means of a rather involved (and inherently sequential) algorithm. The algorithm indeed proves a somewhat stronger result than Lemma 1 and has also been used in [ACR97] in a different context. For the sake of proving Lemma 1, it might however be overkill.

Our results. We show how to use *dispersers* and weak random sources to simulate BPP in polynomial time and to even solve a more general *sampling* problem.

The sampling problem we are interested in is as follows: Given oracle access to a function $f : \{0,1\}^n \to \{0,1\}$ and a weak source of randomness, we want to find a set S of size poly(n) that with high probability is ϵ-discrepant for f. It should be clear that simulating a given BPP algorithm reduces to the above problem: the computation of a BPP algorithm on a fixed input is an (easy to compute) function f of the outcomes of the random coins. Being able to approximate the fraction of random coin outcomes that make f accept, allows us to decide whether or not the algorithm accepts the input.

We show that dispersers are sufficient for the above sampling problem. The starting point is the observation that, using a disperser and a weak random source, it is possible to generate polynomially many small sets $S_1, \ldots, S_k$ and $H_1, \ldots, H_k$ such that, with high probability, one of the S_i's is ϵ-discrepant for f and one of the H_i's has the hitting property required by the discrepancy test (see Theorem 21). Then, we define $H = \bigcup H_i$. Since the hitting property is *monotone* (adding elements to a set cannot decrease its hitting properties), we have that H will be a hitting set with high probability. We can thus run the discrepancy test on the sets $S_1, \ldots, S_k$, using H as the reference hitting set. We shall then prove that, with high probability, one of the S_i's will pass the test and thus be ϵ-discrepant for f as required.

The main difference between our method and the extractor-based method (mentioned at the beginning) is that the ϵ-discrepant set that is given in output *depends on* the specific function f that is accessed as oracle. The source of this nonobliviousness is the selection of a good set S_j among the candidates $S_1, \ldots, S_k$. As a result,

[1] In the next section we will give a seemingly weaker (but in fact equivalent) formal definition.

our sampling algorithm is not *oblivious* according to the definition of Bellare and Rompel [BR94]; however, it is *nonadaptive*. See [G97] for definitions of these notions and for a survey on sampling.

Our main result can be stated in the following way.

THEOREM 3 (main theorem). *For any $\gamma > 0$, there exist a polynomial p and a deterministic algorithm A such that the following holds. For any $\epsilon > 0$, $n > 0$, any $(p(n/\epsilon), p(n/\epsilon)^\gamma)$-source X, and any $f : \{0,1\}^n \to \{0,1\}$, on input (ϵ, n, X) and oracle access to f, A computes, in time polynomial in n/ϵ, a value D such that with probability at least $1 - 2^{-\mathrm{poly}(n)}$ over the outcomes of the source,*

$$|\mathbf{Pr}_x[f(x) = 1] - D| \leq \epsilon.$$

Note that since the algorithm runs in polynomial time it will make $\mathrm{poly}(n/\epsilon)$ queries to f.

COROLLARY 4. *For any $\gamma > 0$, any BPP algorithm can be simulated in polynomial time, using an (r, r^γ)-source.*

The idea of generating candidate discrepancy sets $S_1, \ldots, S_k$ and then applying the discrepancy test to them also yields a simple proof of Lemma 1. This simplified proof is presented in a preliminary version of this paper [ACRT97] and also in an appendix of the final version of [ACR98]. More recently, Fortnow has observed that an even simpler proof of Theorem 2 can be given by using a previous result of Lautemann [L83]. Fortnow's proof of Theorem 2 does not use the discrepancy test. To the best of our understanding, this new proof does not extend to the context of dispersers and weak random sources, and it seems that we still need the discrepancy test in order to prove Theorem 3. An additional, and fairly surprising result observed by Fortnow is that BPP can be simulated by an RP machine having oracle access to a promise-RP problem. We present both of Fortnow's results in section 5.

Overview of the paper. We give some definitions in section 2. In section 3 we describe the discrepancy test and its properties. In section 4 we prove Theorem 3. Fortnow's proof of Theorem 2 is presented in section 5. Section 6 is devoted to some concluding remarks.

2. Preliminaries. Unless otherwise stated, probabilities are with respect to the uniform distribution. For any positive integer n we denote by $\mathcal{F}_n$ the set of all n-ary Boolean functions $f : \{0,1\}^n \to \{0,1\}$. For a vector $a \in \{0,1\}^n$, and a function $f : \{0,1\}^n \to \{0,1\}$, we define a function $f^{\oplus a} : \{0,1\}^n \to \{0,1\}$ as $f^{\oplus a}(x) = f(x \oplus a)$.

We say that a Boolean function f *accepts* x if $f(x) = 1$.

DEFINITION 5 (weak random source). *A probability distribution D over the set $\{0,1\}^r$ is an (r, r^γ)-source* (weak random source of min entropy r^γ) *if, for any $x \in \{0,1\}^r$, $D(x) \leq 2^{-r^\gamma}$.*

For a vertex v of a graph $G = (V, E)$ we let $\Gamma(v) \subseteq V$ be the set of vertices that are adjacent to v. For a subset $S \subseteq V$, we define $\Gamma(S) = \bigcup_{v \in S} \Gamma(v)$. We give here a definition of dispersers which is more convenient than that given in section 1 to describe our results. It is easy to verify that the two definitions are in fact equivalent.

DEFINITION 6 (disperser). *A bipartite multigraph $G(V, W, E)$ with $|V| = R$ and $|W| = N$ is said to be an (R, N, T)-disperser if, for any subset $S \subseteq V$ such that $|S| \geq T$, it holds that $\Gamma(S) \geq N/2$.*

DEFINITION 7 (circuit complexity). *For a Boolean function $f : \{0,1\}^n \to \{0,1\}$ we denote by $L(f)$ the minimum size of a circuit computing f (here, for* circuit *we mean* a circuit whose gates have fan-in at most 2 and arbitrary fan-out*).*

DEFINITION 8 (Kolmogorov complexity). *Let us fix a universal Turing machine U with alphabet $\{0,1\}$ for programs allowing oracle queries. Given two Boolean functions $f : \{0,1\}^k \to \{0,1\}$ and $g : \{0,1\}^n \to \{0,1\}$, we define the* conditional Kolmogorov complexity of g given f, *denoted $K_U(g|f)$, as the length of the shortest program for U that evaluates g having oracle access to f.*

For example, $K_U(f|f) = O(1)$. As usual, if we fix another universal Turing machine U', it holds that $K_{U'}(g|f) = K_U(g|f) + \Theta(1)$. We will usually omit the subscript. See, e.g., [LV90] for an introduction to Kolmogorov complexity. In this paper we use only the obvious fact that, for any fixed f, the number of functions g such that $K(g|f) \leq k$ is at most 2^k.

DEFINITION 9 (hitting set). *A (multi)set $H \subseteq \{0,1\}^n$ is said to be δ-hitting for a family of functions $\mathcal{G} \subseteq \mathcal{F}_n$ if, for any $f \in \mathcal{G}$ with $\mathbf{Pr}_x[f(x) = 1] > \delta$, there exists $x \in H$ such that $f(x) = 1$.*

Recall that by our convention $\mathbf{Pr}_x(\cdot) = \mathbf{Pr}_{x \in \{0,1\}^n}(\cdot)$.

DEFINITION 10 (discrepancy set). *A (multi)set $S \subseteq \{0,1\}^n$ is said to be ϵ-discrepant for a family of functions $\mathcal{G} \subseteq \mathcal{F}_n$ if, for any $f \in \mathcal{G}$,*

$$|\mathbf{Pr}_{x \in S}[f(x) = 1] - \mathbf{Pr}_x[f(x) = 1]| \leq \epsilon.$$

Note that if a set is ϵ-discrepant for a family $\mathcal{G}$, then it is also ϵ-hitting for $\mathcal{G}$, but the converse is not necessarily true.

The definition below is a slight variant of the definition of *quick δ-HSG of price $O(\log n)$* given in [ACR98].

DEFINITION 11 (hitting-set generator). *A quick δ-HSG is a polynomial-time algorithm $\mathcal{H}$ that, on input a number n in unary, returns a multiset $\mathcal{H}(n) \subseteq \{0,1\}^n$ that is δ-hitting for the set $\{f : \{0,1\}^n \to \{0,1\} : L(f) \leq n\}$.*

It may seem awkward to restrict the above definition to functions having circuit complexity equal to the number of inputs. However, any n-ary function of circuit complexity N can be seen as a N-ary function of circuit complexity N whose value is independent of $N - n$ of its inputs. (This point of view does not change the fraction of satisfying inputs as long as we consider constant fractions as done below.) As a consequence of this observation, the set $\mathcal{H}(n)$ returned by the HSG hits *any* function of circuit complexity at most n.

Using straightforward amplification, it is easy to show the following useful property of HSGs.

LEMMA 12 (see [ACR98]). *Let $\delta(n)$ and $k(n)$ be polynomial-time computable functions such that $0 < \delta(n) < 1$ and $k(n) \leq \mathrm{poly}(n)$. Then if a quick $(1-\delta(n))$-HSG exists, there also exists a quick $(1 - (\delta((k(n)+1) \cdot n))^{1/k(n)})$-HSG. In particular, for any two constants $0 < \delta, \delta' < 1$, if there exists a quick δ-HSG, then there exists a quick δ'-HSG.*

Proof. We use the standard *sequential repetition* method. For an input n, $\mathcal{H}'$ first computes (using $\mathcal{H}$) a set $H \subseteq \{0,1\}^{(k(n)+1) \cdot n}$ that is $(1 - \delta((k(n)+1) \cdot n))$-hitting for all the functions $g : \{0,1\}^{(k(n)+1)n} \to \{0,1\}$ such that $L(g) \leq (k(n)+1)n$. Then, it generates a set $H' \subseteq \{0,1\}^n$ by "parsing" each element of H into $k(n)+1$ strings of length n.

We claim that H' is $(1 - (\delta((k(n)+1) \cdot n))^{1/k(n)})$-hitting for functions of circuit size n. Let $f : \{0,1\}^n \to \{0,1\}$ be such that $L(f) \leq n$ and

$$\mathbf{Pr}_x[f(x) = 1] \;>\; 1 - (\delta((k(n)+1) \cdot n))^{1/k(n)}.$$

DISC-TEST(f, S, H, ϵ)
begin
  $p_{\min} := \min\{p(a, f, S) : a \in H \cup \{\vec{0}\}\}$;
  $p_{\max} := \max\{p(a, f, S) : a \in H \cup \{\vec{0}\}\}$;
  if $p_{\max} - p_{\min} \leq \epsilon$ **then return** (1)
  else return (0)
end

FIG. 1. *The discrepancy test.*

Let $f^k : \{0,1\}^{(k(n)+1)n} \to \{0,1\}$ be the function that takes $k(n)+1$ strings of $\{0,1\}^n$ and whose value is 1 if and only if f evaluates to 1 on at least one of the first $k(n)$ strings. Note that $L(f^k) \leq k(n)L(f) + k(n) = k(n) \cdot (n+1)$. We have

$$\mathbf{Pr}_y[f^k(y) = 1] \;=\; 1 - (\mathbf{Pr}_x[f(x) = 0])^{k(n)} \;>\; 1 - \delta((k(n)+1) \cdot n).$$

Due to its hitting property, H contains an input that satisfies f^k and thus H' contains an input that satisfies f. The main claim follows.

For the second claim, if $\delta' \geq \delta$, then there is nothing to prove since, by definition, a δ-HSG is also a δ'-HSG for any $\delta' \geq \delta$. If $\delta' < \delta$, then we take a large enough k such that $\delta' \geq (1 - (1-\delta)^{1/k})$ and then we use the main claim. □

Observe that by applying Lemma 12 with $k(n) = \text{poly}(n)$, it is possible to show that, for any $0 < \epsilon < 1$, the existence of a quick $(1 - 2^{-n^{1-\epsilon}})$-HSG implies the existence of a quick $(1/\text{poly}(n))$-HSG. By using random walks on expander graphs instead of simple repetition, Andreev, Clementi, and Rolim [ACR97] show that, for $c > 1/2$, even the existence of a $(1 - 2^{-cn})$-HSG is an equivalent condition.

3. The discrepancy test. In this section we describe the discrepancy test from [ACR98]. We present a slight variation of the proof of [ACR98] that the test is sound and also prove a "completeness" property of the test.

For any vector $a \in \{0,1\}^n$, function $f : \{0,1\}^n \to \{0,1\}$, and set $S \subseteq \{0,1\}^n$, define

$$p(a, f, S) = \mathbf{Pr}_{x \in S}[f(x \oplus a) = 1]. \tag{1}$$

For any two subsets $S, H \subseteq \{0,1\}^n$, constant $\epsilon > 0$, and function $f : \{0,1\}^n \to \{0,1\}$, we define in Figure 1 a discrepancy test, denoted DISC-TEST(f, S, H, ϵ). In this test, the set S is tested to be ϵ-discrepant for f by using the auxiliary (hitting) set H.

THEOREM 13 (soundness of DISC-TEST [ACR98]). *A constant c_1 exists such that, for any $\epsilon > 0$, integer n, function $f : \{0,1\}^n \to \{0,1\}$, and sets $S, H \subseteq \{0,1\}^n$, if* DISC-TEST$(f, S, H, \epsilon) = 1$ *and H is δ-hitting for the set of functions g such that $K(g|f) \leq c_1 \cdot |S| \cdot n$, then S is $(\epsilon + \delta)$-discrepant for f.*

Theorem 13 is the core of the results of [ACR98]. Note that it says that a set H with a certain *one-sided* pseudorandom property (the hitting property) can be used to test S for a *two-sided* pseudorandom property (the discrepancy property). However, H has to be hitting for *a whole set* of functions while S is tested for discrepancy *on a single function* (i.e., f). Therefore, roughly speaking, the theorem trades off "globality" and "two-sidedness." The version of Theorem 13 proved in [ACR98] requires f to be computable by a small circuit and H to be δ-hitting for a family of functions of low circuit complexity. Here we have no requirement on f, and H is required to be hitting for a set of functions directly "related" to f.

```
function BAD(a)
constants
  p_min, p_max, m;
  s_1, ..., s_m;
begin
  count := 0;
  for i := 1 to m do
    count := count + f(a ⊕ s_i);
  if count > m p_max or count < m p_min then
    return (1)
  else
    return (0)
end.
```

FIG. 2. *How to compute* BAD. *The algorithm has oracle access to* f. *It first computes the number of* 1*s of* $f(a \oplus s_i)$ *for* $i = 1, \ldots, m$ *and then decides whether to accept or reject by comparing this number with* $p_{\min}$ *and* $p_{\max}$.

Proof of Theorem 13. Let f, $S = \{s_1, \ldots, s_m\}$, H, ϵ be fixed throughout the proof, and suppose H is δ-hitting for all gs with $K(g|f) \leq c_1 mn$. Let us define the function $\text{BAD} : \{0,1\}^n \to \{0,1\}$ as

$$\text{BAD}_{f,S,H}(a) \stackrel{\text{def}}{=} \text{BAD}(a) = \begin{cases} 0 & \text{if } p_{\min} \leq p(a,f,S) \leq p_{\max}, \\ 1 & \text{otherwise,} \end{cases}$$

where $p_{\min}$ and $p_{\max}$ are as defined in Figure 1.

CLAIM 14. $K(\text{BAD}|f) \leq mn + 2\log m + O(1)$.

Proof. We observe that BAD can be computed with the pseudocode depicted in Figure 2.

Let us bound the length of such a program. All the elements of S have to be defined explicitly, and this can be done by using mn bits; $p_{\min}$ and $p_{\max}$ have to be defined too, and since they are integral multiples of $1/m$, $\log m$ bits are sufficient to encode each of them. The rest of the program has constant length. We can conclude that the total length of the program is $mn + 2\log m + O(1)$. □

We fix c_1 large enough so that, using the hypothesis of the theorem, H is δ-hitting for BAD.

CLAIM 15. $\mathbf{Pr}_{a \in \{0,1\}^n}[\text{BAD}(a) = 1] \leq \delta$.

Proof. Assume, by contradiction, that $\mathbf{Pr}_{a \in \{0,1\}^n}[\text{BAD}(a) = 1] > \delta$. Then by the hitting property of H, there exists some $a \in H$ such that $\text{BAD}(a) = 1$, which is impossible by definition of BAD, $p_{\min}$ and $p_{\max}$ (as for any $a \in H$, we have $p_{\min} \leq p(a,f,S) \leq p_{\max}$). □

Let $\mathbf{E}[p(a,f,S)]$ be the average of $p(a,f,S)$ over all the choices of $a \in \{0,1\}^n$.

CLAIM 16. $\mathbf{E}[p(x,f,S)] = \mathbf{Pr}_x[f(x) = 1]$.

Proof.

$$\begin{aligned} \mathbf{E}[p(x,f,S)] &= \mathbf{Pr}_{x \in \{0,1\}^n,\ y \in S}[f(x \oplus y) = 1] \\ &= \frac{1}{|S|} \sum_{y \in S} \mathbf{Pr}_x[f(x \oplus y) = 1] \\ &= \frac{1}{|S|} \sum_{y \in S} \mathbf{Pr}_x[f(x) = 1] \end{aligned}$$

$$= \mathbf{Pr}_x[f(x) = 1] \,. \qquad \square$$

From Claim 15 we have the following inequalities (where the first term is due to a's for which $\text{BAD}(a) = 0$ and the second term is due to the rest):

$$\mathbf{E}\,[p(a, f, S)] \leq (1 - \delta) \cdot p_{\max} + \delta \cdot 1 \leq p_{\max} + \delta, \tag{2}$$

$$\mathbf{E}\,[p(a, f, S)] \geq (1 - \delta) \cdot p_{\min} \geq p_{\min} - \delta. \tag{3}$$

Recall that whenever the test accepts, $p_{\max} - p_{\min} \leq \epsilon$. Also, by definition,

$$p_{\min} \leq p(0, f, S) = \mathbf{Pr}_{x \in S}[f(x) = 1] \leq p_{\max}.$$

By Claim 16 and (2), we obtain

$$\mathbf{Pr}_x[f(x) = 1] - \mathbf{Pr}_{x \in S}[f(x) = 1] \leq (p_{\max} + \delta) - p_{\min} \leq \epsilon + \delta,$$

and, similarly,

$$\mathbf{Pr}_{x \in S}[f(x) = 1] - \mathbf{Pr}_x[f(x) = 1] \leq p_{\max} - (p_{\min} - \delta) \leq \epsilon + \delta.$$

Thus, S is indeed $(\epsilon + \delta)$-discrepant for f, and Theorem 13 follows. $\square$

We now give a sufficient condition for DISC-TEST to accept.

THEOREM 17 (completeness of DISC-TEST). *If S is $(\epsilon/2)$-discrepant for the set $\{f^{\oplus a} : a \in \{0,1\}^n\}$, then* $\text{DISC-TEST}(f, S, H, \epsilon) = 1$ *for any set $H \subseteq \{0,1\}^n$.*

Proof. Fix $H \subseteq \{0,1\}^n$ and let a_1 (respectively, a_2) be a point where $p_{\min} = p(a_1, f, S)$ (respectively, $p_{\max} = p(a_2, f, S)$).

$$\begin{aligned} p_{\min} &= \mathbf{Pr}_{x \in S}[f^{\oplus a_1}(x) = 1] \\ &\geq \mathbf{Pr}_x[f^{\oplus a_1}(x) = 1] - \epsilon/2 \\ &= \mathbf{Pr}_x[f(x) = 1] - \epsilon/2, \end{aligned}$$

where the first inequality is due to the discrepancy property of S. Similarly, we have

$$\begin{aligned} p_{\max} &= \mathbf{Pr}_{x \in S}[f^{\oplus a_2}(x) = 1] \\ &\leq \mathbf{Pr}_x[f^{\oplus a_2}(x) = 1] + \epsilon/2 \\ &= \mathbf{Pr}_x[f(x) = 1] + \epsilon/2, \end{aligned}$$

and thus $p_{\max} - p_{\min} \leq \epsilon$. $\square$

4. Proof of Theorem 3. The starting point of our proof is the following easy observation: If we have a set $I \subseteq \{0,1\}^N$ such that $\mathbf{Pr}_x[x \in I] > 1/2$, then using a weak random source and the dispersers of [SSZ95], we can generate a polynomial-sized (in N) set of vectors $x_1, \ldots, x_k$ such that, with high probability, $\{x_1, \ldots, x_k\} \cap I \neq \emptyset$. This is formalized in Corollary 19 below.

A naive way of using this fact would be to take the set I as the family of ϵ-discrepant sets S for f of size m. For large enough m ($m = O(1/\epsilon^2)$ would suffice) the set I will be such that $\mathbf{Pr}_{S \subseteq \{0,1\}^m}[S \in I] > 1/2$, and so we can use the weak random source and the disperser to generate a family of sets $S_1, \ldots, S_k$ such that, with high probability, one of them is ϵ-discrepant for f. But now the problem is that we are not able to recognize which of these sets has the discrepancy property

(note that an efficient Las Vegas algorithm to test the discrepancy property would imply $ZPP = BPP$). Theorem 13 gives indeed a way to test for discrepancy, provided that we have a hitting set at hand.

We thus define $I \subseteq \{0,1\}^{(m+M)\cdot n}$ as the family of pairs of sets (H, S) such that H has M elements and the hitting property as in the hypothesis of Theorem 13 and S has m elements and the discrepancy property as in the hypothesis of Theorem 17. As shown in Lemma 20, for an appropriate choice of m and M, the set I is such that $\mathbf{Pr}_{(H,S)}[(H,S) \in I] > 1/2$. Using the weak random source, we can thus obtain a set of pairs $(H_1, S_1), \ldots, (H_k, S_k)$ such that, for some j, the set S_j has the required discrepancy property and H_j the required hitting property (with high probability). The next important observation is that the hitting property is *monotone*, that is, if a set H has a certain hitting property and J is any set, then $H \cup J$ has at least the same hitting property of H (the reader may note that the discrepancy property is *not* monotone). As a consequence, the set $\bigcup_i H_i$ has (with high probability) the hitting property required by Theorem 13.

We start by quoting the disperser construction of Saks, Srinivasan, and Zhou.

THEOREM 18 (construction of dispersers [SSZ95]). *For any $0 < \lambda < \alpha \leq 1$, for any sufficiently large r, and for any $2^{r^\alpha} \leq T \leq 2^r$, there exists an efficient construction of a $(2^r, 2^{r^\lambda}, T)$-disperser $G = (V, W, E)$ of degree* $\mathrm{poly}(r)$.

In Theorem 17, by "efficient construction" we mean the existence of an algorithm that for any vertex of V finds its neighbors in time $\mathrm{poly}(r)$.

COROLLARY 19. *For any choice of constants $0 < \gamma < 1$ and $c > 0$ there exist a polynomial p and an algorithm A such that the following property holds. For any $n > 0$, any set $I \subseteq \{0,1\}^n$ with $|I| > 2^{n-1}$, and any $(p(n), p(n)^\gamma)$-source X, algorithm A, on input (n, X), outputs a set $C \subseteq \{0,1\}^n$ of size* $\mathrm{poly}(n^{c/\gamma})$ *such that*

$$\mathbf{Pr}[C \cap I = \emptyset] \leq 2^{-n^c},$$

where the probability is taken over the outcomes of the source.

We will use Corollary 19 by taking I as the set of pairs (S, H) such that S has a certain discrepancy property and H has a certain hitting property. Observe that algorithm A computes the set of "candidates" C without "knowing" which set I has been fixed.

Proof of Corollary 19. Fix constants α and λ such that $0 < \lambda < \alpha < \gamma$ and $n^{c\lambda} \leq n^\gamma - n^\alpha$. Let $r = n^{1/\lambda}$. Consider a $(2^r, 2^n, 2^{r^\alpha})$-disperser $G = (V, W, E)$ which can be efficiently constructed, as in Theorem 18. We identify V with $\{0,1\}^r$ and W with $\{0,1\}^n$. Let $B \subseteq V$ be the set of "bad" vertices v such that $\Gamma(v) \subseteq W - I$. We claim that $|B| < 2^{r^\alpha}$; otherwise, we reach a contradiction since, by definition of disperser, we would have $|\Gamma(B)| \geq 2^n/2$ while $|W - I| < 2^n/2$. Let us select an element v of V using an (r, r^γ)-source, and let C be the set of its neighbors. On the one hand, the probability that we picked a bad vertex $v \in B$ is at most $2^{r^\alpha} \cdot 2^{-r^\gamma} = 2^{n^{\alpha/\lambda} - n^{\gamma/\lambda}} \leq 2^{-n^c}$. On the other hand, if $v \notin B$, then $C \cap I \neq \emptyset$; the corollary thus follows. □

As preparation for using Corollary 19, we show that, for a randomly chosen pair of sets (S, H) of sufficiently large sizes, with high probability S has a certain discrepancy property and H has a certain hitting property.

LEMMA 20. *There exist two constants c_2 and c_3 such that for any $\epsilon > 0$, $n > 0$, $f : \{0,1\}^n \to \{0,1\}$, $c > 0$ and for $m = c_2 n/\epsilon^2$ and $M = c_3 cmn/\epsilon$, for a randomly chosen element (v, u) (where $v \in \{0,1\}^{Mn}$ and $u \in \{0,1\}^{mn}$) the following hold with probability larger than* $1/2$*:*

1. *v, regarded as a multiset of $\{0,1\}^n$ of size M, is $\epsilon/2$-hitting for the set of functions g such that $K(g|f) \leq cmn$;*

2. *u, regarded as a multiset of $\{0,1\}^n$ of size m, form a set that is $\epsilon/4$-discrepant for $\{f^{\oplus a} : a \in \{0,1\}^n\}$.*

Proof. It suffices to prove that each event holds with probability larger than 3/4.

Regarding the first event, the number of functions $g \in \mathcal{F}_n$ such that $K(g|f) \leq cmn$ is clearly at most 2^{cmn}. If one such g has $\mathbf{Pr}_x[g(x) = 1] \geq \epsilon/2$, then the probability that M randomly chosen elements from $\{0,1\}^n$ do not hit g is at most

$$(1 - \epsilon/2)^M \leq e^{-\epsilon M/2}.$$

Since $M = c_3 cmn/\epsilon$, it follows that, for an appropriate choice of c_3, the probability that all the functions g are hit is at least

$$1 - 2^{cmn} e^{-\epsilon M/2} > 3/4.$$

For the second claim, observe that a set of m randomly chosen elements from $\{0,1\}^n$ is not $\epsilon/4$-discrepant for a given function with probability at most $2^{-\Omega(m\epsilon^2)}$ (this follows from the Chernoff bound). Since

$$|\{f^{\oplus a} : a \in \{0,1\}^n\}| \leq 2^n,$$

we have that the probability that a randomly chosen set of $m = (1/\epsilon^2)c_2 n$ elements of $\{0,1\}^n$ is not $\epsilon/4$-discrepant for $\{f^{\oplus a} : a \in \{0,1\}^n\}$ is at most

$$2^n \cdot 2^{-\Omega(m\epsilon^2)} \leq 1/4$$

for an appropriate choice of c_2. ∎

The next theorem gives a method for generating (with high probability) a hitting set and a sequence of candidates for the discrepancy test by using the output of a weak random source.

THEOREM 21. *For any $\gamma > 0$, there exist a polynomial p and an algorithm which for any $\epsilon > 0$, $c > 0$, $n > 0$, and $(p(cn/\epsilon), p(cn/\epsilon)^\gamma)$-source X, given in input (ϵ, c, n, X) and having oracle access to a function $f : \{0,1\}^n \to \{0,1\}$, computes, in time polynomial in n/ϵ, sets $H, S_1, \ldots, S_k \subseteq \{0,1\}^n$ such that the following hold with probability at least $1 - 2^{-\mathrm{poly}(n)}$:*

1. *$|S_1| = |S_2| = \cdots = |S_k|$;*

2. *H is $\epsilon/2$-hitting for the set of functions g such that $K(g|f) \leq c|S_1|n$;*

3. *for some $j \in \{1, \ldots, k\}$, S_j is $\epsilon/4$-discrepant for the set of functions $\{f^{\oplus a} : a \in \{0,1\}^n\}$.*

We will use Theorem 21 by taking c as the constant c_1 introduced in the statement of Theorem 13 (soundness of DISC-TEST).

Proof of Theorem 21. Let us apply Corollary 19 to the set I of binary strings (u, v) satisfying properties 1 and 2 in Lemma 20. Then we can use a weak random source to generate sets $S_1, \ldots, S_k$ and $H_1, \ldots, H_k$ such that, with probability at least $1 - 2^{-\mathrm{poly}(n)}$, for some j the set S_j (respectively, H_j) has the required discrepancy (respectively, hitting) property item 2 (respectively, 1) of Lemma 20. Since the hitting property is *monotone*, we also have that, with at least the same probability, $H = \bigcup_j H_j$ is $\epsilon/2$-hitting for the set of functions g with $K(g|f) \leq c|S_1|n$. ∎

We are now ready to prove Theorem 3.

Proof of Theorem 3. We generate a set H and sets $S_1, \ldots, S_k$ as in Theorem 21. With probability at least $1 - 2^{-\mathrm{poly}(n)}$ these sets satisfy properties 1–3 of Theorem 21. From now on we assume that this is the case. We then run DISC-TEST$(f, S_i, H, \epsilon/2)$ for $i = 1, \ldots, k$, and we return S_j where j is the smallest index such that DISC-TEST(f, S_j, H, ϵ) accepts. From Theorem 17 (completeness of DISC-TEST) and condition 3 of Theorem 21 we have that at least one such index exists, and from Theorem 13 (soundness of DISC-TEST) we have that the selected set is ϵ-discrepant for f. We then output $D = \mathbf{Pr}_{x \in S}[f(x) = 1]$. □

5. A new proof of Theorem 2 and more (by Fortnow). In this section we will present a simple proof of Theorem 2 from Fortnow. We first have to introduce some new notation.

For a set S and a property Π we denote by $\exists^+ x \in S.\Pi(x)$ the statement "at least half the elements of S have property Π." A promise problem [ESY84] is a pair of disjoint sets of strings (Y, N). An algorithm A solves a promise problem (Y, N) if A accepts any element of Y and rejects any element of N. Languages can be seen as a special case of promise problems where N is the complement of Y. We denote by prRP the promise version of the class RP. That is, a promise problem (Y, N) belongs to prRP if and only if there is a polynomial-time algorithm $A(\cdot, \cdot)$ and a polynomial $p(\cdot)$ such that for any x of length n

$$x \in Y \Rightarrow \exists^+ y \in \{0,1\}^{p(n)}.A(x, y) = 1,$$
$$x \in N \Rightarrow \forall y \in \{0,1\}^{p(n)}.A(x, y) = 0.$$

We will use the following result of Lautemann [L83] (which is an improvement on a previous result by Sipser [S83]).

THEOREM 22 (Lautemann [L83]). *If $L \in BPP$, then there exists a polynomial time computable Boolean function $A(\cdot, \cdot, \cdot)$ and two polynomials $p(\cdot)$ and $q(\cdot)$ such that for any x of length n*

$$x \in L \Rightarrow \exists^+ y \in \{0,1\}^{p(n)}.\forall z \in \{0,1\}^{q(n)}.A(x, y, z) = 1,$$

$$x \notin L \Rightarrow \forall y \in \{0,1\}^{p(n)}.\exists^+ z \in \{0,1\}^{q(n)}.A(x, y, z) = 0.$$

It has been observed by Fortnow that Theorem 22 implies that $BPP \subseteq RP^{prRP}$, where we denote RP^{prRP} as the class of languages that are decidable by RP oracle machines having access to a prRP oracle.

THEOREM 23 (Fortnow). $BPP \subseteq RP^{prRP}$.

Proof. Let L be a BPP language, and let A, p, and q be as in Theorem 22. Consider the following promise problem (Y, N):

$$Y = \{(x, y) : |y| = p(|x|) \wedge \exists^+ z \in \{0,1\}^{q(n)}.[A(x, y, z) = 0]\};$$

$$N = \{(x, y) : |y| = p(|x|) \wedge \forall z \in \{0,1\}^{q(n)}.A(x, y, z) = 1\}.$$

By definition $(Y, N) \in prRP$. In Figure 3 an RP oracle algorithm that solves L by using one query to (Y, N) is described.

We now prove the correctness of the algorithm. If $x \in L$, then for at least half the choices of y we have that $(x, y) \in N$; thus the algorithm accepts with probability at least half. If $x \notin L$, then for any y we have $(x, y) \in Y$, so the algorithm accepts with probability 0. □

Theorem 2 follows from Theorem 23, since it is easy to see that if a δ-HSG exists for some constant $0 < \delta < 1$, then any RP problem and any prRP promise problem is solvable in P.

```
input : x;
begin
  Pick a random y ∈ {0,1}^{p(|x|)};
  Ask the oracle query (x, y);
  if the oracle answers YES then reject
  else accept;
end.
```

FIG. 3. *The RP^{prRP} algorithm solving a generic BPP problem.*

6. Conclusions. We have demonstrated how to simulate BPP algorithms in polynomial time by using weak random sources of r bits and min-entropy r^γ for any $\gamma > 0$.

The main novelty in our result has been the use of *dispersers* in a context where *extractors* seemed to be necessary. Extractors have other applications besides the use of weak random sources (see, e.g., [N96, Z96:stoc]). It could be the case that techniques similar to ours can give stronger results or simplified proofs in these other applications as well. It remains an open question whether it is possible, for any $\gamma > 0$ and any m, to efficiently construct a $(2^r, 2^m, d, r^\gamma, 1/7)$-extractor with r and d polynomial in m. Such a construction would provide an alternative proof of the main result of this paper and would have other interesting applications.

We also emphasize that our simulation runs in NC. This is due to the parallel nature of our construction and to the fact that it is possible to give an NC construction of the SSZ-dispersers [SSZ97]. Thus, our method provides also an efficient simulation of $BPNC$ algorithms using weak random sources.

Likewise, the proof of Lemma 1 as appeared in a preliminary version of this paper [ACRT97], as well as the proof of Theorem 2 described in section 5, implies an NC simulation of randomized algorithms when both the algorithm and the hitting set are given as oracles. In contrast, the proof of Lemma 1 that appeared in [ACR97] seems to be inherently sequential. Andreev, Clementi, and Rolim [ACR97] have recently used the NC proof of Theorem 2 in order to provide sufficient conditions (in terms of worst-case circuit complexity) for $NC = BPNC$.

Our main result (Theorem 3) can be generalized to the case where the function f that we want to sample is not Boolean but takes real values in the range $[0, 1]$. The proof of Theorem 3 contained in this paper can be easily generalized to the case of such functions. We choose, however, to state and prove only the case of Boolean functions since proofs are cleaner and since, as proved in [G97], sampling real-valued functions is reducible to sampling Boolean functions. We can thus get the following result as a corollary of Theorem 3 and of [G97, Theorem 5.5].

COROLLARY 24. *For any $\gamma > 0$, there exist a polynomial p and a deterministic algorithm A such that the following holds. For any $\epsilon > 0$, $n > 0$, any $(p(n/\epsilon), p(n/\epsilon)^\gamma)$-source X, and any $f : \{0,1\}^n \to [0,1]$, on input (ϵ, n, X) and oracle access to f, A computes, in time polynomial in n/ϵ, a value $\tilde{f}$ such that with probability at least $1 - 2^{-\mathrm{poly}(n)}$ over the outcomes of the source,*

$$|\bar{f} - \tilde{f}| \leq \epsilon,$$

where $\bar{f} = 2^{-n} \sum_x f(x)$ is the average of f.

Acknowledgments. We are grateful to Oded Goldreich for several valuable comments and suggestions on preliminary versions of this paper. In particular, the use of a counting argument à la Kolmogorov is due to Oded. We thank Lance Fortnow for his permission to mention his results in this paper. We thank Madhu Sudan and Avi Wigderson for helpful discussions on Fortnow's results and Michael Saks, Aravind Srinivasan, and Shiyu Zhou for showing us that OR-dispersers can be obtained by an NC construction, and for other helpful conversations.

REFERENCES

[ACR98] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim, *A new general de-randomization method*, J. ACM, 45 (1998), pp. 179–213.

[ACR97] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim, *Worst-case hardness suffices for derandomization: A new method for hardness vs randomness trade-offs*, in Proc. 24th International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 1256, Springer-Verlag, New York, 1997, pp. 177–187.

[ACRT97] A.E. Andreev, A.E.F. Clementi, J.D.P. Rolim, and L. Trevisan, *Weak random sources, hitting sets, and BPP simulations,* in Proc. 38th IEEE Symposium on Foundations of Computer Science, 1997, pp. 264–272.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson, *BPP has subexponential time simulations unless EXPTIME has publishable proofs,* Comput. Complexity, 3 (1993), pp. 307–318.

[BM84] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits,* SIAM J. Comput., 13 (1984), pp. 850–864.

[BR94] M. Bellare and J. Rompel, *Randomness-efficient oblivious sampling,* in Proc. 35th IEEE Symposium on Foundations of Computer Science, 1994, pp. 276–287.

[CG88] B. Chor and O. Goldreich, *Unbiased bits from sources of weak randomness and probabilistic communication complexity,* SIAM J. Comput., 17 (1988), pp. 230–261.

[CW89] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification, and weak random sources,* in Proc. 30th IEEE Symposium on Foundations of Computer Science, 1989, pp. 14–19.

[ESY84] S. Even, A. Selman, and Y. Yacoby, *The complexity of promise problems with applications to public-key cryptography,* Inform. and Control, 2 (1984), pp. 159–173.

[G97] O. Goldreich, *A sample of samplers—A computational perspective on sampling,* Electronic Colloquium on Computational Complexity, 1997, TR97-020.

[IW97] R. Impagliazzo and A. Wigderson, *P= BPP if E requires exponential circuits: Derandomizing the XOR lemma*, in Proc. 29th ACM Symposium on Theory of Computing, 1997, pp. 220–229.

[L83] C. Lautemann, *BPP and the polynomial hierarchy*, Inform. Proc. Lett., 17 (1983), pp. 215–217.

[LV90] M. Li and P. Vitany, *Kolmogorov complexity and its applications,* in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, New York, 1990, pp. 187–254.

[MR95] R. Motwani and P. Raghavan, *Randomized Algorithms,* Cambridge University Press, Cambridge, UK, 1995.

[N90] N. Nisan, *Using Hard Problems to Create Pseudorandom Generators*, ACM Distinguished Dissertations, MIT Press, Cambridge, MA, 1990.

[N96] N. Nisan, *Extracting randomness: How and why,* in Proc. 11th IEEE Conference on Computational Complexity, 1996, pp. 44–58.

[NW94] N. Nisan and A. Wigderson, *Hardness vs randomness*, J. Comput. System Sci., 49 (1994), pp. 149–167.

[NZ96] N. Nisan and D. Zuckerman, *Randomness is linear in space,* J. Comput. System Sci., 52 (1996), pp. 43–52.

[S83] M. Sipser, *A complexity theoretic approach to randomness*, in Proc. 15th ACM Symposium on Theory of Computing, 1983, pp. 330–335.

[SSZ95] M. Saks, A. Srinivasan, and S. Zhou, *Explicit dispersers with polylog degree,* in Proc. 27th ACM Symposium on Theory of Computing, 1995, pp. 479–488.

[SSZ97] M. Saks, A. Srinivasan, and S. Zhou, personal communication, March 1997.

[SV86] M. Santha and U. Vazirani, *Generating quasi-random sequences from slightly random*

sources, J. Comput. System Sci., 33 (1986), pp. 75–87.

[SZ94] A. Srinivasan and D. Zuckerman, *Computing with very weak random sources,* in Proc. 35th IEEE Symposium on Foundations of Computer Science, 1994, pp. 264–275.

[T96] A. Ta-Shma, *On extracting randomness from weak random sources,* in Proc. 28th ACM Symposium on Theory of Computing, 1996, pp. 276–285.

[V86] U. Vazirani, *Randomness, Adversaries and Computation*, Ph.D. thesis, University of California, Berkeley, CA, 1986.

[V87] U. Vazirani, *Efficiency considerations in using semi-random sources,* in Proc. 19th ACM Symposium on Theory of Computing, 1987, pp. 160–168.

[VV85] U. Vazirani and V. Vazirani, *Random polynomial time is equal to slightly random polynomial time*, in Proc. 26th IEEE Symposium on Foundations of Computer Science, 1985, pp. 417–428.

[Y82] A.C. Yao, *Theory and applications of trapdoor functions,* in Proc. 23rd IEEE Symposium on Foundations of Computer Science, 1982, pp. 80-91.

[Z90] D. Zuckerman, *General weak random sources,* in Proc. 31st IEEE Symposium on Foundations of Computer Science, 1990, pp. 534–543.

[Z96:stoc] D. Zuckerman, *Randomness-optimal sampling, extractors and constructive leader election,* in Proc. 28th ACM Symposium on Theory of Computing, 1996, pp. 286–295.

[Z96] D. Zuckerman, *Simulating BPP using a general weak random source,* Algorithmica, 16 (1996), pp. 367–391.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2117–2132

DOMINATORS IN LINEAR TIME*

STEPHEN ALSTRUP†, DOV HAREL‡, PETER W. LAURIDSEN†,
AND MIKKEL THORUP†

Abstract. A linear-time algorithm is presented for finding dominators in control flow graphs.

Key words. control flow analysis, dominators, algorithms

AMS subject classifications. 68Q25, 68N20

PII. S0097539797317263

1. Introduction. Finding the dominator tree for a control flow graph is one of the most fundamental problems in the area of global flow analysis and program optimization [2, 3, 4, 6, 12, 17]. The problem was first raised in 1969 by Lowry and Medlock [17], where an $O(n^4)$ algorithm for the problem was proposed (as usual, n is the number of nodes and m the number of edges in a graph). The result has been improved several times (see, e.g., [1, 2, 19, 22]), and in 1979 an $O(m\alpha(m,n))$ algorithm was found by Lengauer and Tarjan [16]. Here α is the inverse Ackermann's function. Finally, in 1985 Dov Harel [13] announced a linear-time algorithm. Based on Harel's results, linear-time algorithms have been found for many other problems (see, e.g., [4, 6, 12]). Harel's description was, however, incomplete. In this paper, we give a complete description of a different and simpler linear-time dominator algorithm.

The paper is divided as follows. In section 2 the main definitions are given. In section 3 we outline the Lengauer–Tarjan algorithm. In section 4 we give a linear-time dominator algorithm. A detailed comparison with Harel's approach [13] is given in section 4.7. Finally, in section 5 we briefly discuss dominators in the simpler case of reducible control flow graphs. The appendix contains implementation details of the algorithm in section 4.

2. Definitions. A *control flow graph* is a directed graph $G = (V, E)$, with $|V| = n$ and $|E| = m$, in which $s \in V$ is a start node, from which all nodes in V are reachable through the edges in E (see, e.g., [2]). If $(v, w) \in E$, we say that node v is a *predecessor* of node w and w is a *successor* of v. Node v *dominates* w if and only if all paths from s to w pass through v. Node v is the *immediate dominator* of node w, denoted $idom(w) = v$ if v dominates w and every other node that dominates w also dominates v. The tree T induced by the edges $(idom(w), w)$ is called the *dominator tree* of G. Hence, the root of T is the start node s, and for all other nodes v, its parent in T is its immediate dominator.

3. Lengauer and Tarjan's algorithm. In this section we outline Lengauer and Tarjan's algorithm, since the idea behind our algorithm is to optimize subroutines used in this algorithm.

Lengauer and Tarjan's algorithm [16] runs in $O(m\alpha(m,n))$ time. Initially a depth-first search (DFS) [21] is performed in the graph, resulting in a DFS-tree $\mathcal{T}$, in which

*Received by the editors February 26, 1997; accepted for publication (in revised form) July 1, 1998; published electronically June 29, 1999.
http://www.siam.org/journals/sicomp/28-6/31726.html
†Department of Computer Science, University of Copenhagen, Copenhagen, Denmark (stephen@diku.dk, waern@diku.dk, mthorup@diku.dk).
‡Microsoft Israel Ltd., R & D Center, Matam Haifa, 31905 Israel (dovh@microsoft.com).

the nodes are assigned a DFS-number. In this paper we will not distinguish between a node and its DFS-number. The nodes are thus ordered such that $v < w$ if the DFS-number of v is smaller than the DFS-number of w.

The main idea of the Lengauer–Tarjan algorithm is first to compute the so-called *semidominators*, $sdom(v)$, for each node $v \in V \backslash \{s\}$ as an intermediate step for finding dominators. The semidominator of a node v is an ancestor of v defined as

$$sdom(v) = \min\{u | \text{ a path} u, w_1, \ldots, w_k, v \text{ exists, where } w_i > v \text{ for all } i = 1, \ldots, k\}.$$

The semidominators are found by traversing the tree $\mathcal{T}$ in decreasing DFS-number order while maintaining a dynamic forest, $\mathcal{F}$, which is a subgraph of the DFS-tree $\mathcal{T}$. Hence, an edge $(v, w) \in \mathcal{T}$, included in $\mathcal{F}$, is in $\mathcal{F}$ connecting the parent v with its child w. The following operations should be supported on $\mathcal{F}$:

- LINK(v, w): Adds the edge $(v, w) \in \mathcal{T}$ to $\mathcal{F}$. The nodes v and w are root nodes of trees in $\mathcal{F}$.
- EVAL(v): Finds the minimum *key* value of nodes on the path from r to v, where r is the root of the tree in $\mathcal{F}$ to which v belongs.[1]
- UPDATE(v, k): Sets $key(v)$ to be k, where the node v must be a singleton tree.

We will now give a more detailed description of the Lengauer–Tarjan algorithm. The forest $\mathcal{F}$ initially contains all nodes as singleton trees and the computation of semidominators is done as follows:

- Initially we set $key(v) = v$ for all nodes $v \in V$.
- The nodes are then visited in decreasing DFS-number order, i.e., v is visited before w if and only if $v > w$. When visiting a node v, we call UPDATE(v, k), where $k = \min\{\text{EVAL}(w) | (w, v) \in E\}$.
- After visiting v, a call LINK(v, w) is made for all children w of v.

After running this algorithm we have $key(v) = sdom(v)$. The correctness of the algorithm (i.e., that when a node is updated, it is with the correct *sdom*-value) follows from the following theorem given by Lengauer and Tarjan [16, Theorem 4]:

THEOREM 3.1. *For any node* $v \neq s$, $sdom(v) = \min(S_1 \bigcup S_2)$ *where* $S_1 = \{w | (w, v) \in E \ \wedge w < v\}$ *and* $S_2 = \{sdom(u) | u > v \ \wedge (w, v) \in E \ \wedge u$ *is an ancestor of* $w\}$. □

To see the connection between Theorem 3.1 and the algorithm above consider the visit of node v in the algorithm. Since EVAL$(w) = w$ for $w < v$, $S_1 = \{\text{EVAL}(w) | (w, v) \in E \wedge w < v\}$. To see that $S_2 = \{\text{EVAL}(w) | (w, v) \in E \wedge w > v\}$, note that if $u > v$, $(w, v) \in E$, and u is an ancestor of w in $\mathcal{T}$, then u and w have already been visited. Thus u is an ancestor of w in a tree in $\mathcal{F}$, so EVAL(w) includes $sdom(u)$.

Tarjan and Lengauer show that having found the semidominators, the immediate dominators can be found within the same complexity.

The EVAL-LINK operations in the algorithm are performed using a slightly modified version of Tarjan's UNION-FIND algorithm for disjoint sets [23]. Since n LINK and m EVAL operations are performed, the complexity is $O(m\alpha(m, n))$. Thus a linear-time algorithm can be obtained if the EVAL and LINK operations can be performed in $O(n + m)$ time.

[1] In [16], EVAL is defined to consider the root only if the root is the argument of the EVAL. In the process of finding a dominator tree, this happens only if the tree is a singleton node. We have given the definition above to avoid confusion, as it is this definition which will be used in our algorithm. The Lengauer–Tarjan algorithm presented here is therefore a slight modification of the original algorithm. More specifically, the modification consists of performing the LINK(v, w) operation when v is visited instead of when w is visited.

4. A linear-time algorithm. In this section we present a linear-time dominator algorithm. The overall idea is to convert the on-line EVAL-LINK algorithm to an off-line algorithm by exploiting the fact that the tree resulting from LINK operations is known in advance. The inspiration for this stems from the linear UNION-FIND algorithm for disjoint sets by Gabow and Tarjan [11]. In the Gabow–Tarjan algorithm, the tree, T, resulting from all UNION operations is known in advance. More specifically this means that a UNION(v, w) operation is only permitted if the edge (v, w) is in T. The FIND queries are then defined as usual, whereas UNION(v, w) is defined as the union of the sets to which v and w belong. The linear time is achieved by tabulating the behavior of UNION-FIND within small *microtrees* of size $O(\log n)$.

The original approach of Harel was to convert this linear UNION-FIND algorithm into an EVAL-LINK algorithm [13]. Roughly speaking, the basic idea was to define a new parameter of nodes, referred to as pseudodominator, which satisfy the following two conditions: (a) pseudodominators can be propagated in linear time, and (b) using pseudodominators, we can compute semidominators in linear time. This approach had a couple of drawbacks, further elaborated upon in section 4.7. Here we do not involve pseudodominators, but calculate the semidominators directly. Not only do we know the structure of the tree resulting from the links, we also know that the LINKS occur in reverse-DFS order. Instead of converting the linear UNION-FIND algorithm, we end up using it as a black box. Moreover, the information needed for tabulating EVAL is found using Fredman and Willard's Q-heaps [9], which were not available at the time of [13]. Finally, our choice of microtrees leads to simpler calculations.

4.1. An $O(n \log n + m)$algorithm. We consider a forest, $\mathcal{F}$, of trees. Recall that to each node a *key* is associated, which initially contains the DFS-number of the node. Let T_v denote the tree in $\mathcal{F}$ to which v belongs. We will use the term *self-contained* for nodes, for which EVAL$(v) = key(v)$. Hence a node v is self-contained if all ancestors of v in T_v have key values $\geq key(v)$. Note that the definition implies that all root nodes in $\mathcal{F}$ are self-contained. A node v stops being self-contained when T_v is linked to a root node u, for which $key(u) < key(v)$.

LEMMA 4.1. *Let $nsa(v)$ denote the nearest self-contained ancestor of v.*

(a) *For any node $v \in V$, we have $EVAL(v) = key(nsa(v))$.*

(b) *For any node pair $u, v \in V$, if $nsa(v) = u$ at some point in the Lengauer–Tarjan algorithm, then $nsa(v) = nsa(u)$ in the remainder of the algorithm.*

Proof.

(a) By the definition of self-contained nodes, $key(nsa(v))$ is the least *key* value of nodes on the path from the root of T_v to $nsa(v)$. By the same definition, if nodes with *key* values $< key(nsa(v))$ were on the path from $nsa(v)$ to v in T_v, the node with least depth among these nodes would be self-contained.

(b) By definition, $nsa(u)$ is the nearest self-contained ancestor of u. The fact that $nsa(v) = u$ implies that $T_v = T_u$ and that all nodes on the path from u to v in T_v have *key* values $> key(u)$. By the definition of UPDATE none of these nodes will change *key* values again. The node $nsa(v)$ will therefore always be the nearest self-contained ancestor of u. □

By Lemma 4.1(b) we can represent the *nsa*-relation efficiently by using disjoint sets. Let each self-contained node, u, be the *canonical element* of the set $\{v | nsa(v) = u\}$. By Lemma 4.1(a), an EVAL(v) operation is then reduced to finding the canonical element of the set to which v belongs; hence EVAL$(v) = key(SetFind(v))$.

When a LINK(u, v) operation is performed, the node v will no longer be the root of T_v. Therefore a set A of nodes in T_v may stop being self-contained. A node $w \in A$

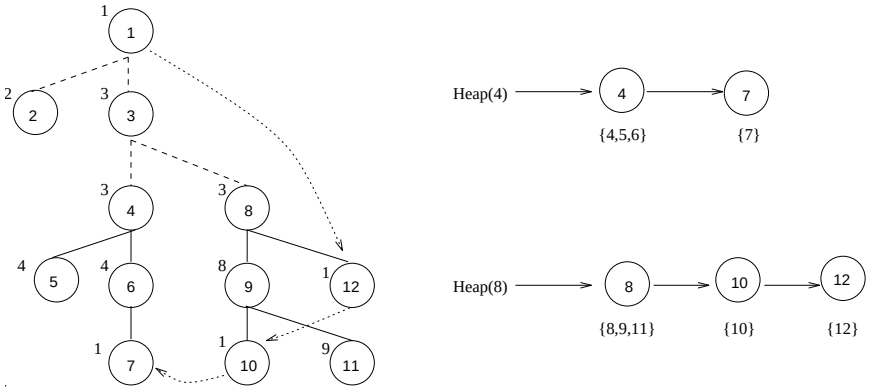


FIG. 4.1. *To the left a sample DFS-tree in which the nodes are labeled by their DFS-numbers is given. The full lines indicate the part of the tree which has been linked, hence the node to be processed is node "3." The dotted arrows are graph edges. The numbers beside the nodes are their key values. To the right the two nontrivial heaps containing self-contained nodes are illustrated as lists. Below the self-contained nodes the sets associated with them are listed.*

is the canonical element of the set of nodes whose EVAL values change from $key(w)$ to $key(u)$. We can thus maintain the structure by unifying the sets associated with nodes in A with the set associated with u.

To find the set A, a heap, supporting HeapFindMax, HeapExtractMax, and Heap Union (e.g., [8, 24]), is associated with each root of a tree in $\mathcal{F}$. Each heap contains the self-contained nodes in the tree (see Figure 4.1). The set A can then be found by repeatedly extracting the maximum element from the heap associated with v until the maximum element of this heap is $\leq key(u)$.

The algorithm LINK(u, v) is as follows:

- **While not** $Empty(Heap(v))$ **and** $key(FindMax(Heap(v))) > key(u)$ **do**
- $w := ExtractMax(Heap(v))$;
- $SetUnion(u, w)$; /* The canonical element of the resulting set is u */
- **od**;
- $Heap(u) := HeapUnion(Heap(u), Heap(v))$;

LEMMA 4.2. *The algorithm presented performs the n LINK and UPDATE operations interspersed with m EVAL operations in $O(m + n \log n)$ time.*

Proof. At most $O(n)$ $HeapExtract$, $HeapFindMax$, and $HeapUnion$ operations are performed. Each of these operations can be done in $O(\log n)$ time using an ordinary heap (e.g. [8, 24]). Since the tree structure is known in advance, the set operations can be computed in linear time using the result from [11].[2] It will, however, suffice to use a simple disjoint set algorithm which rearranges the smallest of the two sets. □

4.2. Decreasing roots. In section 4.4 we will need the ability to decrease the *key* value of a node, while it is the root of a tree. We will therefore extend the algorithm from the previous section to handle the $DecreaseRoot(v, k)$ operation, which sets $key(v) = k$, where v is the root of T_v. The $DecreaseRoot(v, k)$ operation should be done in constant time.

[2] If this result is used, the $SetUnion$ operation should be changed according to the description given earlier in this section. More specifically, the call would be $SetUnion(parent(w), w)$ and the canonical element of the resulting set would be the canonical element of the set that includes $parent(w)$.

Assume that a $DecreaseRoot(v, k)$ operation has been performed. Analogous to the LINK operation from the previous section, this may imply that some self-contained nodes in T_v are no longer self-contained. We should therefore remove such nodes from the heap and unify the sets associated with them with the set associated with v, as was done in the LINK operation. However, in the algorithm from the previous section, the root node v is the maximum element in the heap associated with it. In order to remove nodes from the heap, we would first have to remove v, which would require $O(\log n)$ time. We should note that since the heap returns maximum values the usual *decreasekey* operation for heaps cannot be used. We can, however, take advantage of the fact that the root node will always be the maximum element in the heap it belongs to. It is therefore not necessary to explicitly insert the root into the heap until it is linked to its parent. The $DecreaseRoot(v, k)$ operation is performed as follows.

- **While not** $Empty(Heap(v))$ **and** $key(FindMax(Heap(v))) > k$ **do**
- $w := ExtractMax(Heap(v))$;
- $SetUnion(v, w)$;
- **od**;
- $key(v) := k$;

LEMMA 4.3. *We can perform d DecreaseRoot and n LINK and UPDATE operations interspersed with m EVAL operations in $O(n \log n + m + d)$ time.*

Proof. We change the algorithm from the previous section by postponing the insertion of a root node, r, into $Heap(r)$, until r is linked to its parent. This has no effect on the complexity of EVAL and UPDATE operations stated in Lemma 4.2. Since each node may be deleted from a heap at most once, the total number of $ExtractMax$ and $SetUnion$ operations invoked by LINK and $DecreaseRoot$ is still limited by the number of nodes in the tree. The cost of these operations can therefore be charged to the LINK operations. Since the remaining operations invoked by $DecreaseRoot$ are each done in constant time, the additional complexity of the d $DecreaseRoot$ operations is $O(d)$. □

4.3. A linear-time algorithm for paths. We consider the situation in which the tree $\mathcal{T}$ is a path. Recall that in the algorithms from the previous two subsections we needed a heap to order self-contained nodes. The property which distinguishes paths from trees in this context is that this ordering is induced by the path. More specifically, any pair u, v of self-contained nodes on the part of the path which has been linked are ordered, such that $key(v) \geq key(u)$ if and only if $depth(v) \leq depth(u)$. To perform LINK operations on a path, we can therefore use the algorithm from the previous section, where the heap is replaced by a stack. The algorithm for the operation LINK(u, v) on a path is as follows.

- **While not** $StackEmpty$ **and** $key(StackTop) > key(u)$ **do**
- $w := StackPop$;
- $SetUnion(u, w)$;/* The canonical element of the resulting set is u */
- **od**;
- $StackPush(u)$;

An EVAL operation on a path is performed in analogy with the previous section, hence EVAL$(v) = key(SetFind(v))$.

LEMMA 4.4. *If the tree $\mathcal{T}$ is a path, we can perform the n LINK and UPDATE operations and the m EVAL operations in $O(n + m)$ time.*

Proof. The stack operations are done in linear time since each node will only be on the stack once. By using the result from [11] the set operations are performed in amortized constant time. □

4.4. A faster algorithm for trees with few leaves. We can take advantage of the linear time algorithm from the previous section by using it on the paths in $\mathcal{T}$. More specifically, we define *I-paths* to be maximal paths of $\mathcal{T}$ consisting of at least two unary nodes, i.e., nodes with at most one child. We now define the forest R as the forest $\mathcal{F}$ with each maximal segment of an I-path from $\mathcal{T}$ contracted into a single node, called an *I-node*. In the process of linking, the correspondence between R and the forest $\mathcal{F}$ is as follows.

- When the node with largest depth on an I-path is linked to its child, c, in $\mathcal{F}$, the *I-node* is linked to c in R.
- When the node with least depth on an I-path is linked to its parent, p, in $\mathcal{F}$, the *I-node* is linked to p in R.

We will use the result from section 4.2 for nodes in R and the result from the previous section for nodes on I-paths. The above correspondence means that EVAL queries on nodes in R correspond to EVAL queries in $\mathcal{F}$ if, for any I-path P, $key(I\text{-}node(P))$ is the least *key* value on the part of P which has been linked. In other words, we use $I\text{-}node(P)$ to represent the minimum self-contained node on P in R. During the processing of an I-path P, the *key* value of $I\text{-}node(P)$ should thus be properly updated. This is done by invoking a $DecreaseRoot(I\text{-}node(P), k)$ operation each time a new minimum *key* value k is found on P.

The EVAL queries on nodes on an I-path, P, will be correct as long as the node with least depth on P has not yet been linked to its parent. We can therefore construct an interface between R- and I-paths as follows. We associate a pointer, *I-root*, with each node on an I-path. The pointer is initially set to be NULL and when the node with least depth on an I-path is linked to its parent p, we set $I\text{-}root(v) = p$, for all nodes v belonging to the I-path. The algorithm for EVAL(v) is as follows (we use subscripts to distinguish between the structures EVAL operations are performed in):

- **if** v belongs to an I-path P **then**
- **if** $I\text{-}root(v)$ =NULL **then return** $\text{EVAL}_P(v)$
- **else return** min $\{\text{EVAL}_P(v), \text{EVAL}_R(I\text{-}root(v))\}$
- **else return** $\text{EVAL}_R(v)$.

LEMMA 4.5. *Let l denote the number of leaves in $\mathcal{T}$. We can perform m EVAL and n LINK and UPDATE operations in $O(l \log l + m + n)$ time.*

Proof. The I-paths are processed in linear time by Lemma 4.4 and since the I-root pointer is updated only once, this can also be done in linear time. Since the I-paths have been contracted R contains $O(l)$ nodes. Thus by Lemma 4.3, R can be processed in time $O(l \log l + m + d)$, where d is the number of *DecreaseRoot* operations. The number of *DecreaseRoot* operations is, however, bounded by the number of nodes on I-paths. □

4.5. Reducing to small subtrees. By lemma 4.5 we have that the EVAL-LINK algorithm can be performed effectively on trees with few leaves. However, the number of leaves is only bounded by the number of nodes. To reduce the number of leaves in $\mathcal{T}$, subtrees of size $\leq \log n$ can be removed from the bottom of $\mathcal{T}$ (a technique also used in [7]). We will refer to such subtrees as *S-trees*. Assume that all S-trees have been removed from the tree $\mathcal{T}$. Then each leaf in the remaining tree must be a node in $\mathcal{T}$ with at least $\log n$ descendants. Thus the remaining tree has at most $n/\log n$ leaves. By Lemma 4.5 we can therefore perform amortized constant-time EVAL, LINK, and UPDATE operations in the remaining tree.

We now show how to process the S-trees. Recall that the LINK operations are performed in decreasing DFS-number order. This implies that EVAL operations of

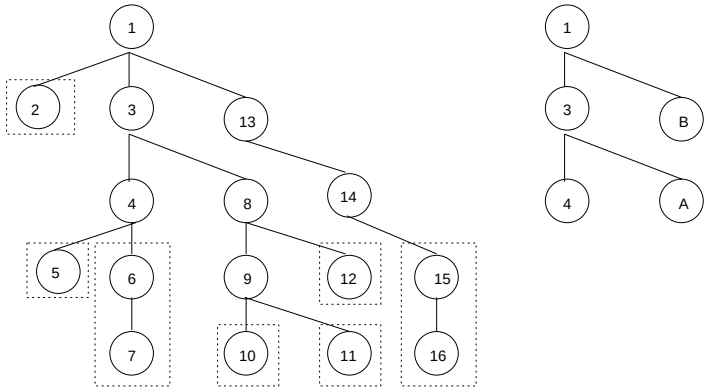


FIG. 4.2. *To the left a sample DFS-tree is given. The boxes indicate S-trees of size* ≤ 2. *To the right the reduced tree is given. The nodes "A" and "B" are replacing I-paths. Note that if S-trees of size* $\leq \log_2 16 = 4$ *were removed, the tree would be reduced to a single node representing the I-path* $1, 3, 8$.

nodes in S-trees, induced by nodes outside, will only take place at a time when all links have been performed inside the structure. Furthermore, the links inside S-trees are performed successively; hence, each S-tree can be processed independently. Analogously with I-paths, we can associate a pointer *S-root* with each node in each S-tree, which points to the parent of the root of the S-tree after the LINK between the root and its parent has been performed. Then an EVAL operation on a node v in an S-tree becomes

$$\begin{array}{ll} \text{EVAL}_S(v), & \text{if } S\text{-}root(v)\text{=NULL}, \\ min\{\text{EVAL}_S(v), \text{EVAL}(S\text{-}root(v))\} & \text{otherwise.} \end{array}$$

To perform EVAL, LINK, and UPDATE operations inside an S-tree we could use lemma 4.2. Alternatively, we could repeat the removal of subtrees on the S-trees because of the independent nature of S-trees. Let $T(m, n, a)$ denote the time it takes to support the m EVAL and n LINK and UPDATE operations in the Lengauer–Tarjan algorithm within subtrees of $\mathcal{T}$ each of size $\leq a$. For example, the construction of Lemma 4.2 gives $T(m, n, a) = O(m + n \log a)$.

LEMMA 4.6. $T(m, n, a) = O(m + n) + T(m, n, \log a)$.

Proof. We process each subtree S as follows. Remove maximal subtrees of size at most $\log a$ from S. Then, in the upper tree of S we have at most $|S|/\log a$ leaves. Since $a \geq |S|$, by Lemma 4.5 the cost of the LINKs and UPDATEs in S is $O(|S|)$, resulting in a total cost of $O(n)$ over all S-trees. An EVAL query in the upper tree of S has constant cost by Lemma 4.5. An EVAL query to an subtree may propagate to the root via the root-pointer in the subtree, but it takes only additional constant time. □

Since $T(m, n, 1) = O(m + n)$, repeating the above recurrence $\log^* n$ times, we immediately get

$$T(m, n, n) = O((m + n) \log^* n).$$

However, in this paper we need only to repeat it twice, giving the following corollary.

COROLLARY 4.7. $T(m, n, n) = O(m + n) + T(m, n, \log \log n)$. □

In the next subsection, we will show that $T(m, n, \log \log n) = O(m+n)$, implying a linear-time algorithm for finding dominators.

Figure 4.2 illustrates the division of a tree into I-paths and S-trees on one level.

4.6. Tabulation of small trees. In this section we show how to perform constant-time EVAL, UPDATE, and LINK operations on trees of size $\leq \log\log n$, henceforth denoted as *microtrees*. We will do this by constructing a table containing EVAL values for all possible forest permutations. We first show how to compute such a table assuming that a superset of *sdom* values is known for each microtree. Following that we show how to choose this superset. Combining these results we show that the microtrees can be processed in linear time. Finally we give the theorem, which completes the dominator algorithm. We start out by giving a lemma by Fredman and Willard [9].

LEMMA 4.8. *The Q-heap performs insertion, deletion, and search operations in constant time and accommodates as many as* $(\log n)^{1/4}$ *items given the availability of* $O(n)$ *time and space for preprocessing and word size* $\geq \log n$. □

For a set M' of different values we define the rank of a value $x \in M'$ as the number of values $< x$ in M'.

THEOREM 4.9. *Assume that to each microtree* M, $|M| \leq \log\log n$, *we are given a set of values* M', *where* $|M'| = O(|M|)$, *and that for all UPDATE*(v, k) *operations,* $v \in M \Rightarrow k \in M'$. *Assume also that the order in which LINK operations occur is known. It is then possible to perform constant-time EVAL, LINK, and UPDATE operations, given the availability of* $O(n)$ *time and space for preprocessing and word size* $\geq \log n$.

Proof. First we sort the set M' of size $O(\log\log n)$ in linear time using Lemma 4.8. The *key* value of each node is replaced by its rank in M', which simply is an index into the sorted set. In order to perform UPDATE and EVAL operations efficiently, we need a table which maps *key* values to ranks and vice versa. Since all *key* values are $< n$, this table requires only $O(n)$ space. Given this table we can now maintain the microtree using the rank of the node instead of its *key* value. Thus an EVAL output from the microtree should be mapped to a *key* value using the above table. Since the size of the rank is exponentially smaller than the *key* value, we can now use tabulation for microtrees as follows. We construct each possible tree of size $\leq \log\log n$. Since in general there are at most $O(2^k)$ trees of size k (all trees of size k can be uniquely represented by a bitmap of size $2k$), there are at most $\log n$ such trees. Due to the reverse-DFS order of LINKs, in each of the trees there are only $\log\log n$ ways the nodes can be partially linked. Finally for each of these forests of partially linked trees, we construct copies holding all possible permutations of ranks to nodes. In each of these forests we compute the EVAL value for each node. We then construct a table which outputs the computed EVAL values. This computation can be done in a time proportional to the number of nodes in the trees: For each tree with root r, traverse the tree top-down and set EVAL(r)=r, and for each node $v \neq r$ set EVAL(v)=min($key(v)$,EVAL($parent(v)$)). The number of nodes is the product of the number of trees ($\log n$), the number of LINK's ($\log\log n$), the number of rank permutations (($C_1 * \log\log n)^{\log\log n}$), and the number of nodes in each tree ($\log\log n$); thus the number of nodes is (C_i are constants):

$$\begin{aligned}
&\log n * \log\log n * (C_1 * \log\log n)^{\log\log n} * \log\log n \\
&= \log n^{C_2} * (\log\log n)^2 * \log\log n^{\log\log n} \\
&\leq \log n^{C_3} * \log\log n^{\log\log n} \\
&= \log n^{C_3} * \log n^{\log\log\log n} \\
&= \log n^{C_3 + \log\log\log n} = O(n).
\end{aligned}$$

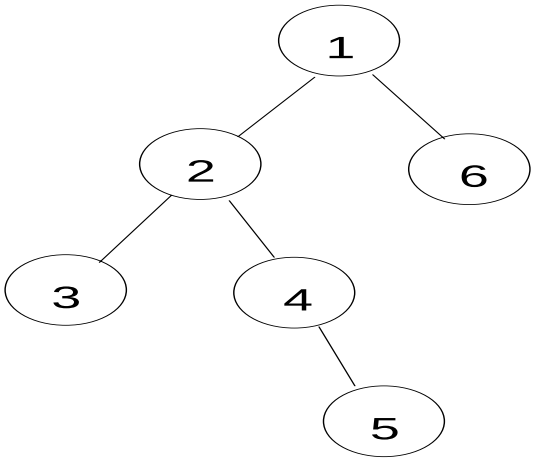


FIG. 4.3. *A sample tree labeled by DFS-numbers. The bitmap of the tree is "*11011000100*."*

To store each forest, the forest table from [11], which requires $\log\log n$ space, can be used. The rank of each node requires $\log\log\log n$ space. If we attach a new number to each node inside the forest we can identify each node using $\log\log\log n$ space. Hence, each entry to the table requires $\log\log n + \log\log\log n * \log\log n + \log\log\log n$ space, which will fit into a computer word of size $\geq \log n$. The size of the table is thus $O(n)$. To carry out the operations given a microtree, we first compute the table entry for the tree without any links. The EVAL operations are done by looking up the table and the LINK and UPDATE operations are done by updating the entry. Finally, in order to perform UPDATE and EVAL operations, we use the table which maps *key* values to ranks and vice versa. Next we will be a little more specific on how to maintain the forest table. The forest table from [11] supports any ordering of the LINK operations, whereas in the dominator algorithm the links are performed in decreasing DFS-number order. We can therefore simplify the representation by using the DFS-traversal to represent each tree. More specifically, we start at the root and use a bitmap in which "1" means that an edge is followed down in the tree and "0" means that we move to the parent of the current node. The tree traversal is finished when "0" is encountered while the root is the current node. As a special case this means that a single node tree is represented by the bitmap "0." The mapping is illustrated in Figure 4.3. Instead of representing the LINK's explicitly we can save the number of nodes in the tree, which at some point in time has been processed by the algorithm. Since the size of the forests can differ, we also need to save the size of each tree. Finally the *key* and EVAL values of the nodes can be saved in order of the DFS-traversal. The bitmap of an entry can thus have the following configuration [SIZE‖TREE‖KEYS‖EVAL‖LINK], where SIZE and LINK are blocks of $\log\log\log n$ bits, EVAL and KEYS use SIZE bits, and TREE uses (2*SIZE-1) bits.

To construct the entry of a microtree we traverse it in DFS-order and set the bits of TREE and SIZE accordingly. The KEYS are initialized to the rank of the DFS-numbers and LINK is initialized to 0. A microLINK operation is performed by incrementing the LINK value and the microUPDATE(v, k) operation is done by replacing the value of v in the entry with k. □

Theorem 4.9 requires a superset M' of *sdom* values for nodes in a microtree M. The next lemma shows how M' can be chosen.

LEMMA 4.10. *Let* $M_1 = \bigcup_{v \in M} \min\{(EVAL(w))|\ (w,v) \in E \wedge w \notin M\}$ *and* $M' = M \bigcup M_1$. *For all* $v \in M$ *we have that* $sdom(v) \in M'$.

Proof. The lemma is obviously true in case $sdom(v) \in M$. Assume therefore that $u = sdom(v) \notin M$ and that $u \notin M'$. By the definition of semidominators, a path $u = w_0, w_1, \ldots, w_{k-1}, w_k = v$ exists where $w_i > v$ for $i = 1, \ldots, k-1$. Let w_j be the last node on the path not in M. Since $u \notin M'$, the node w_{j+1} must have a predecessor x for which $\text{EVAL}(x) < u$. This means that a path exists from a node u', with $u' < u$, to w_{j+1} on which all nodes except u' are $> v$. This path can be concatenated with the path $w_{j+1}, \ldots, w_k, v$, contradicting that $sdom(v) = u$. □

We now complete the microtree algorithm by showing how to compute the sets M' of Lemma 4.10.

THEOREM 4.11. *Let* M *be a microtree of size* $\leq \log\log n$. *Each EVAL, UPDATE, and LINK operation inside* M *in the Lengauer–Tarjan algorithm can be performed in constant time, given the availability of* $O(n)$ *time and space for preprocessing and word size* $\geq \log n$.

Proof. By Theorem 4.9 and Lemma 4.10 we need only to show how to compute the sets M' defined in Lemma 4.10 in $O(|M'|)$ time. We will show this by induction on the visits of microtrees. Recall that the Lengauer–Tarjan algorithm visits nodes in decreasing DFS-number order. When the first microtree is reached all nodes with larger DFS-numbers have thus been processed. By Corollary 4.7, EVAL queries on processed nodes outside microtrees can be done in constant time. Furthermore, all nodes with smaller DFS-numbers will at this stage be singleton trees. The EVAL queries required in Lemma 4.10 can thus be performed in constant time for the first microtree. Given an arbitrary microtree M we can therefore assume that constant time EVAL queries can be performed in microtrees containing nodes with larger DFS-numbers than the nodes in M. For nodes not in microtrees, we can compute the EVAL values needed in Lemma 4.10 in constant time by the same arguments as above. By induction this is also the case for nodes in previously visited microtrees.

Finally we should note that in the proof of Theorem 4.9, $O(n)$ space was used for the table, which maps *key* values to ranks for a microtree. Since the microtrees are computed independently, this space can be reused, so that the overall space requirement is $O(n)$. □

We can now combine the results of this section in the following theorem.

THEOREM 4.12. *The EVAL, LINK, and UPDATE operations in the Lengauer–Tarjan algorithm can be performed in linear time.*

Proof. Follows directly from Corollary 4.7 and Theorem 4.11. □

4.7. Relation to Harel's algorithm. The proof of Theorem 1 of [13], which is omitted, employs a linear–time-table construction using a variant of dynamic programming. The details of this construction are beyond the scope of this paper.

Harel's original value propagation required the construction of supersets of the sets of *sdom* values for all microtrees in a separate phase, in order to presort the values. The main drawback of the technique is that it leads to a rather complicated case analysis, and checking correctness is pretty tedious. In fact, the original value propagation algorithm in [13] contains an error (more precisely Theorem 3b in [13] is false as stated and a concrete counterexample is given in [15, Section 4, p. 12]).

The algorithm described in this paper avoids the above problems [13] by using the following technique. We change the construction so that microsets are removed only from the bottom of the tree, which simplifies value propagation. The tables required to prove Theorem 1 of [13] are replaced by the use of Fredman and Willard's priority

queues. This technique is more general and allows us to propagate semidominator values on a per microset basis, just prior to computing the exact semidominator values for all members of a microset.

5. Algorithms for reducible graphs. The problem of finding dominators in reducible graphs has been investigated in several papers (e.g., [1, 18, 20]). The reason why reducible graphs are considered is that the control flow graphs of certain programming languages (e.g., Modula-2 [25]) are reducible. A graph is reducible if the edges can be partitioned into two disjoint sets E' and E'' so that

- the graph induced by the edges in E' is acyclic
- for all edges $(v, w) \in E''$, w dominates v.

Since the edges E'' have no influence on the dominance relation, the problem of finding dominators in reducible graphs is analogous to finding dominators in acyclic graphs. In this section we therefore assume that graphs are acyclic.

5.1. The former algorithm is not linear. In 1983, Ochranova [18] gave an algorithm which is claimed to have complexity $O(m)$[3]. Unfortunately the paper does not contain a complexity analysis. In order to disprove the complexity of the algorithm, it is therefore necessary to outline the behavior of the algorithm. For an acyclic graph we have the following facts:

(a) If a node, x, has a single predecessor, y, then $idom(x) = y$.
(b) If each of the successors of a node x has more than one predecessor then no node is dominated by x.

Since at least one successor of the start node s will satisfy the condition in (a), the dominators can be found by starting at s and using the two facts interchangeably as follows:

1. If (a) is true for a successor, v, of the current node, w, then set $idom(v) = w$ and the current node to v.
2. If (b) is true for all successors of the current node w then merge w and $idom(w)$ (by unifying their successor and predecessor sets respectively). Set the current node to be the merged node.

In order for the algorithm to be linear, the detection of whether (a) is true in 1 should have constant-time complexity. Furthermore, the merging of two nodes in 2, which involves union of two sets which are not disjoint, should also have constant-time complexity. The authors are not aware of a general algorithm with the above properties.

5.2. A linear-time algorithm. In this section we give a simple linear-time algorithm for finding dominators in reducible graphs. The algorithm is constructed by combining new techniques [10] with previously presented ideas (see, e.g., [1, 20]). In other words, the algorithm is a compilation.

LEMMA 5.1. *Let T be the dominator tree and let $W(w) = \{v|(v, w) \in E'\}$ be the set of predecessors of a node $w \neq s$ in G. Furthermore, let x be the node in T with the largest depth which is an ancestor in T to all the nodes in $W(w)$. Then $idom(w) = x$.*

Proof. x is an ancestor to the nodes in $W(w)$ and therefore dominates w. By definition of dominators a path from x to $idom(w)$ must exist in T. Conversely, $idom(w)$ is an ancestor to all nodes in $W(w)$; thus a path from $idom(w)$ to x must exist. Hence $idom(w) = x$. □

The computation is divided into two main steps as follows:

[3]Citation: "At least no counterexample was found."

1. The graph $G = (V, E')$ is acyclic and can therefore be topologically sorted [14] ensuring that if $(v, w) \in E'$, then v has a lower topological number than w.
2. Now the dominator tree T can be constructed dynamically. Set s to be the root of the dominator tree T and process each node $w \in (V \backslash \{s\})$ in increasing topological order as described below. Note that the part of T built so far, is used for determining *idom* for the rest of the nodes.
   - Let $B = W(w)$. From lemma 5.1 we have that $idom(w)$ now can be computed by repeatedly deleting two arbitrary nodes from B and inserting the nearest common ancestor (nca) of these nodes into the set B until the set contains only one node.
   - After computing $idom(w)$ the edge $(idom(w), w)$ is added to T.

The only unspecified part of the algorithm is the computation of *nca* in a tree T which grows under the addition of leaves. In [10] an algorithm is given which processes *nca* and the addition of leaves in constant time per operation.

THEOREM 5.2. *The algorithm above computes the dominator tree for a reducible control flow graph with n nodes and m edges in $O(n + m)$ time.*

Proof. Step 1 in the algorithm has complexity $O(n + m)$. In step 2 each node is visited and each edge can result in a query about *nca* in T, so at most m *nca*-queries are performed, which establishes the complexity. □

6. Concluding remarks. A linear-time algorithm has been presented for finding dominators. The result, as presented, is purely theoretical, in the sense that Fredman and Willard's Q-heaps require that $n \geq 2^{12^{20}}$ [9]. Some of our ideas may still be of practical relevance. If, for example, we take Corollary 4.7, giving a rather simple linear-time reduction to subtrees of size at most $\log \log n$, and then use Lemma 4.2 within each of these, we get a simple $O(m + n \log \log \log n)$ algorithm, which in practice may be competitive with that of Lengauer and Tarjan [16].

The problem with the large constant using RAM Q-heaps has recently been overcome by a linear-time pointer-machine algorithm to find dominators, due to Buchsbaum et al. [5].

In the following appendices we present the pseudocode of our algorithm.

Appendix A. Implementation details. We assume that a DFS-search has been performed in the graph. The I-paths are removed from the tree in the following way: The child pointer of the parent to the first node and the parent pointer of the child of the last node are removed. An *I-node* is inserted in its place (see Figure A.1). The *I-node* is numbered by a unique number larger than n. Furthermore, the I-paths are numbered by a number > 0.

The algorithm uses the following arrays in which the DFS-number of nodes are used as indices (the arrays marked * are also used in the Lengauer–Tarjan algorithm):

- $pred(v)^*$: The set of nodes w such that $(w, v) \in E$.
- $parent(v)^*$: The parent of v in the DFS-tree. To simplify the EVAL operation we set $parent(v) = 0$ if $v = 0$.
- $child(v), sibling(v)$: Pointer to the first child and first sibling of v, respectively, in the DFS-tree.
- $I\text{-}path(v)$: If v does not belong to an I-path, $I\text{-}path(v) = 0$. Otherwise $I\text{-}path(v)$ contains the number of the I-path to which v belongs.
- $S\text{-}tree(v)$: True if v belongs to an S-tree.
- $microtree(v)$: True if v belongs to a microtree.
- $root(v)$: This field is defined for nodes in S-trees, microtrees, and I-paths. Before the root of the structure has been linked to its parent, p, $root(v) = 0$.

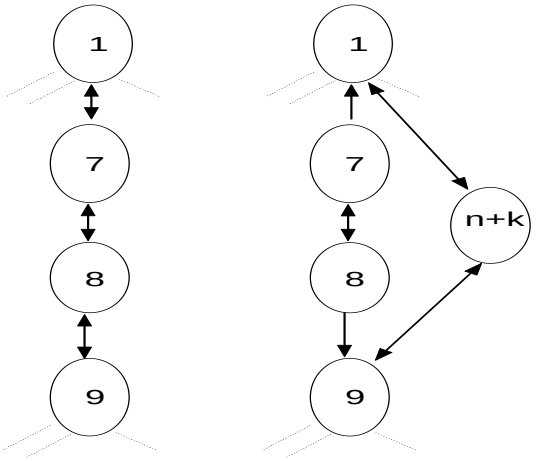


FIG. A.1. *An I-path and the representation of the I-path in the tree. Both child and parent pointers are illustrated.*

Afterwards $root(v)$ contains the number of p.

- $first(v)$: If v belongs to an I-path, this field contains the number of the first node on the I-path.
- $stack(v)$: If v is the first node on an I-path, this field contains the stack used for the I-path.
- $microroot(v)$: If v belongs to a microtree then $microroot(v)$ is the number of the root of the microtree.
- $key(v)$*[4]: After the semidominator of v has been computed, $key(v)$ is the number of the semidominator of v. Initially $key(v) = v$.
- $bucket(v)$*: The set of nodes whose semidominator is v.
- $dom(v)$*: A number which will eventually be the number of the immediate dominator of v.

The main algorithm is a slight modification of the Lengauer–Tarjan algorithm:

```
Begin
 constructmicrotable; /* This procedure computes the microtable */
 for v := 1 to n do bucket(v) := ∅;
 v := n;
 While v > 1 do begin
  if microtree(v) then begin
   microdominator(v, microroot(v)); /* see below */
   v := microroot(v) − 1;
  end else begin
   For each w ∈ pred(v) do begin
    k :=EVAL(w);
    if k < key(v) then UPDATE(v, k);
   end;
   /* The remainder of the algorithm computes dominators */
   /* from semidominators and is analogous to [16] */
   For each child w of v do LINK(v, w);
   bucket(key(v)) := bucket(key(v)) ⋃ {v};
   For each w ∈ bucket(parent(v)) do begin
```

[4]In the Lengauer–Tarjan algorithm, this array is called *semi*.

```
      bucket(parent(v)) := bucket(parent(v))\{w};
      k :=EVAL(w)
      if k < key(w) then dom(w) := k
      else dom(w) := parent(v);
    end;
    v := v − 1;
  end; /* While /*
 end; /* While /*
 for v := 2 to n do
  if dom(v) ≠ key(v) then dom(v) := dom(dom(v));
  else dom(v) := key(v);
End;
```

Procedure *microdominator*$(v, root : integer)$;
The *microdominator* procedure is analogous to the main algorithm. The only real difference is that EVAL, LINK, and UPDATE operations are replaced by microEVAL, microLINK, and microUPDATE operations. Furthermore there are no I-paths in a microtree.

For the EVAL and LINK operations we need the following additional fields:

- $heap(v)$: A heap associated with v.
- $I\text{-}node(v)$: If v is the root of an I-path then $I\text{-}node(v)$ is the number of the node which represents the I-path.

```
Function EVAL(v: integer):integer;
begin
 if v = 0 then EVAL:=∞
 else if microtree(v) then EVAL:=min(EVAL(root(v),microEVAL(v)))
 else if I-path(v) or S-tree(v) then EVAL:=min(EVAL(root(v)), key(SetFind(v)))
 else EVAL:=key(SetFind(v));
end;

Procedure LINK(v, w: integer);
begin
 if microtree(w) then /* w is the root of a microtree */
  For each u in the microtree to which w belongs do root(u) := v
 else if S-tree(w) and not S-tree(v) then /* w is the root of an S-tree */
  For each u in the S-tree to which w belongs do root(u) := v
 else if I-path(v) > 0 then begin
  if first(v) = v then Init-I-path(v, w) /*see below */
  else if I-path(w) = I-path(v) then begin
   S := stack(first(v));
   While not StackEmpty(S) and key(StackTop(S)) > key(v) do begin
    u := StackPop(S);
    SetUnion(v, u);
   end;
   if key(I-node(v)) > key(v) then DecreaseRoot(I-node(v), key(v));
   StackPush(v, S);
  end;
 end else if I-path(w) > 0 then begin /* the path is fully linked */
  For each u on the I-path do root(u) := v;
  /* Add I-node(w) to heap(I-node(w)) */[5]
  LINK(v, I-node(w));
 end else begin /* Neither v nor w is on an I-path */
  While not Empty(heap(w)) and key(HeapFindMax(heap(w))) > key(v) do begin
   u := HeapExtractMax(heap(w));
```

[5]The *I-node* has not been a member of the heap while the I-path has been processed. The pseudocode for this operation is omitted to improve program clarity, as it involves creating a dummy heap and performing a *HeapUnion* operation on the dummy heap and $heap(I\text{-}node(w))$.

```
      SetUnion(v, u);
    end;
    HeapUnion(heap(v), heap(w));
  end;
end;

Procedure Init-I-path(v, w: integer);
/* v is the first node on an I-path and should be linked to its child w */
begin
  CreateStack(S);
  stack(v) := S;
  StackPush(v, S);
  While not Empty(heap(w)) and key(HeapFindMax(heap(w))) > key(v) do begin
    w := HeapExtractMax(heap(w));
    SetUnion(I-node(v), w);
  end;
  heap(I-node(v)) := heap(w);
  key(I-node(v)) := key(v);
end;

Procedure DecreaseRoot(v, k: integer);
begin
  While not Empty(heap(v)) and key(HeapFindMax(heap(v))) > k do begin
    w := HeapExtractMax(heap(v));
    SetUnion(v, w);
  end;
  key(v) := k;
end;

Procedure UPDATE(v, k: integer);
begin
  key(v) := k;
end;
```

The pseudocode of the microalgorithm is rather tedious and therefore is omitted.

Acknowledgment. We wish to thank a referee from *SIAM Journal on Computing* for some very good and thorough comments. Further, Dov Harel wishes to thank Eli Dichtermann for recent discussions.

REFERENCES

[1] A. Aho, J. Hopcroft, and J. Ullman, *On finding lowest common ancestors in trees*, in Fifth Annual ACM Symposium on the Theory of Computing, 1973, pp. 115–132.

[2] A. Aho and J. Ullman, *The Theory of Parsing, Translation and Compiling*, vol. II, Prentice-Hall, Englewood Cliffs, N.J., 1972.

[3] A. Aho and J. Ullman, *Principles of Compiler design*, Addison-Wesley, Reading, MA, 1979.

[4] G. Bilardi and K. Pingali, *A framework for generalized control dependence*, in ACM SIGPLAN Conference on Programming Language Design and Implementation, 1996, pp. 291–300.

[5] A. Buchsbaum, H. Kaplan, A. Rogers, and J. Westbrook, *Linear-time pointer-machine algorithms for lca's, mst verification, and dominators*, in Annual ACM Symposium on the theory of computing (STOC), vol. 30, 1998.

[6] R. Cytron, J. Ferrante, B. Rosen, M. Wegman, and F. Zadek, *Efficiently computing static single assignment form and the control dependence graph*, in ACM Trans. Programming Language Systems, 13 (1991), pp. 451–490.

[7] B. Dixon and R. Tarjan, *Optimal parallel verification of minimum spanning trees in logarithmic time*, Algorithmica, 17 (1997), pp. 11–18.

[8] J. Driscoll, H. Gabow, R. Shrairman, and R. Tarjan, *Relaxed heaps: An alternative to fibonacci heaps with application to parallel computation*, Comm. ACM, 31 (1988), pp. 1343–

1354.
[9] M. Fredman and D. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. System Sci., 48 (1994), pp. 533–551.
[10] H. Gabow, *Data structure for weighted matching and nearest common ancestors with linking*, in First Annual ACM-SIAM Symposium on Discrete Algorithms, 1990, pp. 434–443.
[11] H. Gabow and R. Tarjan, *A linear-time algorithm for a special case of disjoint set union*, J. Comput. System Sci., 30 (1985), pp. 209–221.
[12] G. Gao and V. Sreedhar, *A linear time algorithm for placing ϕ-nodes*, in ACM SIGPLAN-SIGACT Symposium on the Principles of Programming Languages, 1995, pp. 62–73.
[13] D. Harel, *A linear time algorithm for finding dominators in flow graphs and related problems*, in 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 185–194.
[14] D. Knuth, *The Art of Programming*, vol. 1, Addison-Wesley, Reading, MA, 1968.
[15] P. Lauridsen, *Dominators*, Master's Thesis, Department of Computer Science, University of Copenhagen, Copenhagen, Denmark, 1996.
[16] T. Lengauer and R. Tarjan, *A fast algorithm for finding dominators in a flowgraph*, in ACM Trans. Programming Languages Systems, 1 (1979), pp. 121–141.
[17] E. Lowry and C. Medlock, *Object code optimization*, Comm. ACM, 12 (1969), pp. 13–22.
[18] R. Ochranova, *Finding dominators*, in Foundations of Computation Theory, *Lecture Notes in Comput. Sci.*, 4 (1983), pp. 328–334.
[19] P. Purdom and E. Moore, *Immediate predominators in a directed graph*, Comm. ACM, 15 (1972), pp. 777–778.
[20] G. Ramalingam and T. Reps, *An incremental algorithm for maintaining the dominator tree of a reducible flowgraph*, in 21st Annual ACM Symposium on Principles of Programming Languages, 1994, pp. 287–298.
[21] R. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.
[22] R. Tarjan, *Finding dominators in directed graphs*, SIAM J. Comput., 3 (1974), pp. 62–89.
[23] R. Tarjan, *A class of algorithms which require nonlinear time to maintain disjoint sets*, J. Comput. System Sci., 18 (1979), pp. 110–127.
[24] J. Vuillemin, *A data structure for manipulating priority queues*, Comm. ACM, 21 (1978), pp. 309–315.
[25] N. Wirth, *Programming in modula*-2(3rd corr.ed), Springer–Verlag, Berlin, New York, 1985.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2133–2149

APPROXIMATING CAPACITATED ROUTING AND DELIVERY PROBLEMS*

PRASAD CHALASANI† AND RAJEEV MOTWANI‡

Abstract. We provide approximation algorithms for some capacitated vehicle routing and delivery problems. These problems can all be viewed as instances of the following k-delivery TSP: given n source points and n sink points in a metric space, with exactly one item at each source, find a minimum length tour by a vehicle of finite capacity k to pick up and deliver exactly one item to each sink. The only known approximation algorithm for this family of problems is the 2.5-approximation algorithm of Anily and Hassin [*Networks*, 22 (1992), pp. 419–433] for the special case $k = 1$. For this case, we use matroid intersection to obtain a 2-approximation algorithm. Based on this algorithm and some additional lower bound arguments, we devise a 9.5-approximation for k-delivery TSP with arbitrary finite k. We also present a 2-approximation algorithm for the case $k = \infty$.

We then initiate the study of dynamic variants of k-delivery TSP that model problems in industrial robotics and other applications. Specifically, we consider the situation where a robot arm (with finite or infinite capacity) must collect n point-objects *moving* in the Euclidean plane, and deliver them to the origin. The point-objects are moving in the plane with known, identical velocities—they might, for instance, be on a moving conveyor belt. We derive several useful structural properties that lead to constant-factor approximations for problems of this type that are relevant to the robotics application. Along the way, we show that maximum latency TSP is implicit in the dynamic problems, and that the natural "farthest neighbor" heuristic produces a good approximation for several notions of latency.

Key words. capacitated vehicle routing, capacitated delivery, maximum latency problem, matroid intersection, approximation algorithms, traveling salesperson problem, NP-hard

AMS subject classification. 68Q25

PII. S0097539795295468

1. Introduction.

This paper considers two variations on the classical traveling salesperson problem (TSP): (a) the permissible routes are constrained by the requirement that objects must be delivered from sources to sinks by a vehicle of finite capacity k, and (b) the points to be visited may be moving with a known velocity. We define the k-delivery TSP: Given n source points and n sink points in some metric space, with exactly one item placed at each source, compute a minimum length route for a vehicle of capacity k to deliver exactly one item to each sink, *starting and ending at a fixed location.* Note that sources and sinks need not lie at distinct locations. This problem is an instance of vehicle routing or scheduling problems that have been the subject of intensive study in the literature [14, 23]. The problem is easily seen to be NP-hard via a reduction from TSP: place a source and a sink very close to each point of the TSP problem, and set $k = 1$; now, an optimal solution to the 1-delivery TSP is an optimal solution to the TSP instance. Similar reductions can be devised for arbitrary finite k and for infinite k.

*Received by the editors December 1, 1995; accepted for publication (in revised form) December 23, 1997; published electronically June 29, 1999. This paper is a revised and expanded version of *Approximation algorithms for robot grasp and delivery*, in Proceedings of the 2nd International Workshop on Algorithmic Foundations of Robotics, Toulouse, France, 1996, pp. 347–362.

http://www.siam.org/journals/sicomp/28-6/29546.html

†Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 (chal@cs.cmu.edu).

‡Department of Computer Science, Stanford University, Stanford, CA 94305 (rajeev@cs.stanford.edu). The research of this author was supported by the Alfred P. Sloan Research Fellowship, the IBM Faculty Partnership Award, ARO MURI grant DAAH04-96-1-0007, and NSF Young Investigator Award CCR-9357849 with matching support from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

We provide what appear to be the first known polynomial-time constant-factor approximation algorithms for this problem. Motivated by applications in areas such as robotics, we formulate a novel dynamic version of TSP where the points are moving in the plane, and we partially extend our results to that case. In the process we obtain an approximation algorithm for maximum latency TSP. All our algorithms run in time polynomial in n and, where meaningful, k. Some of our approximation factors have been improved upon by Charikar, Khuller, and Raghavachari [12].

In section 2, we begin by considering 1-delivery TSP, or bipartite TSP. This is closely related to the *swapping problem* for which Anily and Hassin [2] present a 2.5-approximation algorithm. We use matroid intersection to obtain a 2-approximation algorithm for this problem. In section 2.1, based on the 1-delivery approximation algorithm and additional lower bound arguments, we devise a 9.5-approximation algorithm for k-delivery TSP with arbitrary finite k. It turns out that for infinite capacity k, the resulting problem is a special case of the "TSP with delivery and backhauls" for which Anily and Mosheiov [3] obtain a factor-2 approximation.

The k-collect TSP is a well-studied vehicle routing problem [1, 23]: it is the special case of the k-delivery TSP where all sinks are at the starting location of the vehicle. Altinkemer and Gavish [1] have shown a 2.5-factor approximation algorithm for this problem and also established its NP-hardness (for $k \geq 2$). In this paper we motivate and present approximation algorithms for the dynamic k-collect TSP: n point-objects are moving in the Euclidean plane with fixed, identical velocities and a robot arm (starting at the origin) with capacity k must pick up and deliver these objects to the origin. Observe that the dynamic ∞-collect TSP, or simply dynamic TSP, is a generalization of standard TSP to the case of moving points. There does not appear to have been any prior theoretical work on dynamic TSP. As described in section 4, the dynamic problem arises in industrial robotics in the context of rapid deployment automation [10]. This problem also has some features of the *time-dependent* TSP [18, 32] wherein the distance function varies with time. We restrict ourselves to the case where the moving points' velocities are sufficiently smaller than that of the robot arm. This is essential to ensure that the robot arm is able to retrieve all objects and is certainly a valid assumption in the motivating application. An interesting variant, which we do not explore here, concerns the model where the velocities are unrestricted and the goal is to maximize the number of points visited, with possible restrictions on the total time available. Clearly, this would involve generalizing the *prize-collecting TSP* [5] to the case of moving points.

Two complications arise in the case of moving points. First, we lose symmetry in the distance matrix; fortunately, though, since the points are all moving at the same velocity, there is some structure in this asymmetry. The second problem is that the distance of points to the origin is time-dependent, although the distances between moving points are time-independent. In addition, the dynamic variants have some interesting and counterintuitive aspects. For instance, suppose $k = 1$ and there are two points, initially at $(10, 0)$ and $(15, 0)$. The robot is initially at the origin. Assuming that the robot can move at speed 1 and the points move at speed $1/2$ in the negative x direction, which point should be visited first? It is easy to check that visiting the *farther* point first produces a smaller total time (20) than the other way round (roughly 22)! In general, the more time we spend visiting points early in the tour, the closer the later points would have moved to the origin, and so we would spend less time visiting them from the origin.

Thus, the dynamic problem has some features of maximum latency TSP: given a set of n points $P = \{p_1, \ldots, p_n\}$ and a symmetric distance matrix (d_{ij}) satisfying the

triangle inequality, find a path starting at p_1 and visiting all other points so as to *maximize* the total latency of the points, where the latency of a point p_i is the length of the path to that point. In section 3, we give a 1/2-approximation algorithm for maximum latency TSP. We also study some variants that arise implicitly in the dynamic settings. While approximation algorithms for *minimum* latency TSP are known [7, 22], there does not appear to be any prior work on our version of the problem.

In section 4.1 we establish some basic properties of travel times involving moving points (such as the triangle inequality) that are not as obvious as they might seem. In section 4.6, we show that an optimal dynamic 1-collect tour must visit points in decreasing order of distances to the origin and that the case $k = \infty$ is an asymmetric TSP with bounded asymmetry. We then present a constant factor approximation for the dynamic k-collect problem for arbitrary finite k. Finally, some extensions are mentioned in section 4.8.

In the rest of the paper we will refer to the sources as blue points and the sinks as red points. We will also assume for convenience that n is a multiple of k; our results can be extended to the general case by introducing a few dummy sources and sinks close to one of the original points.

1.1. Other related work. Capacitated delivery problems have been studied extensively in the literature [14, 23], although the focus has mostly been on finding the optimal solution (using sophisticated branch and bound techniques, for example) and not on finding provably good approximations. Examples of such work include the following.

(a) *The capacitated vehicle routing problem* (CVRP) [1, 25, 23]: given n points in a metric space and an infinite fleet of vehicles of capacity k, find a collection of vehicle routes starting at origin such that every point is visited by exactly one vehicle. Typical objective functions to be minimized included number of vehicles, total distance traveled, maximum distance traveled by a vehicle, or some combination thereof.

(b) *The dial-a-ride problem* [6, 34, 27]: compute an optimal route for a k-capacity van to pick up and drop off n persons between different origin–destination pairs. Note that this problem differs from the k-delivery TSP in that each person must be dropped off at a specific destination.

(c) *The precedence-constrained TSP* [6]: for each vertex i there is a set $P(i)$ of vertices that must be visited before visiting i, and we are required to find an optimal TSP satisfying these constraints.

Although there does not appear to be any prior theoretical work on TSP with moving points, heuristics for related problems have been studied in the robotics literature (see, for example, Li and Latombe [30]). The *static* k-collect TSP has received considerable attention in the literature and the best-known result is the 2.5-approximation algorithm of Altinkemer and Gavish [1]. Also, the *k-person TSP* is a related problem and has a 1.5-approximation algorithm due to Frieze [20] (see also Frederickson, Hecht, and Kim [19]).

2. The k-delivery TSP. We begin by presenting an approximation algorithm for the 1-delivery TSP that underlies our algorithm for the general case. When the vehicle has unit capacity, any delivery route must alternately pick up and deliver one item at a time. The corresponding graph problem is the following.

BIPARTITE TRAVELING SALESPERSON PROBLEM. *Given an edge-weighted graph G satisfying the triangle inequality, with n blue vertices (sources) and n red vertices (sinks), find the optimal* bipartite *tour starting and ending at a designated blue vertex*

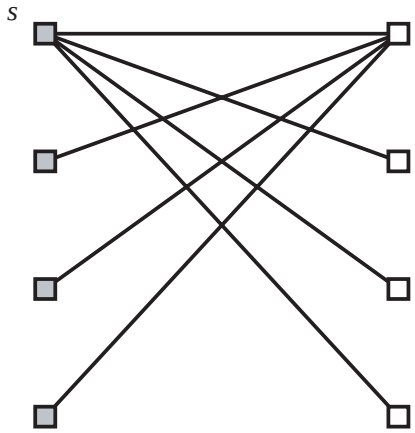


FIG. 1. *A bipartite spanning tree for which no depth-first traversal yields a bipartite tour.*

s and visiting all vertices. A tour is bipartite if no two consecutively visited vertices have the same color.

Anily and Hassin [2] have shown a 2.5-approximation algorithm for a generalization of this problem, known as the swapping problem. Their algorithm finds a perfect matching M consisting of edges that connect red and blue vertices, and it uses Christofides's heuristic [13] to find a tour T of the blue vertices. The final delivery route consists of visiting the blue vertices in the sequence specified by the tour T, using the matching edges in M to deliver an item to a sink and return to the blue vertex (or "shortcut" to the next blue vertex on T). If OPT is the optimal delivery tour, clearly $T \leq 1.5\text{OPT}$ and $M \leq 0.5\text{OPT}$, whereas the total length of the delivery tour is at most $T + 2M \leq 2.5\text{OPT}$. We exploit some combinatorial properties of bipartite spanning trees and matroid intersection to improve this factor to 2.

A naive approach toward a 2-approximation is to mimic the well-known 2-approximation algorithm for the TSP problem: pick a *bipartite* spanning tree of G and then perform a depth-first traversal followed by short-cutting. A spanning tree of G is bipartite if each edge connects a red and blue vertex. Given a bipartite spanning tree T, we can think of it as a tree rooted at s and do a depth-first traversal of T with short-cuts (there may in general be several ways to short-cut) and obtain a tour of G. However, such a tour may not be bipartite; there are bipartite spanning trees that do not yield a bipartite tour regardless of how we do the depth-first traversal and short-cuts (see Figure 1).

Our 2-approximation algorithm is based on the following very simple observations (these observations were communicated to us by Hassin and helped simplify our argument considerably).

Observations. Let T be a bipartite spanning tree where each blue vertex has degree at most 2; then, T has exactly one blue vertex v_1 of degree 1. If T is rooted at v_1 then every blue vertex has exactly 1 (red) child. Clearly, if we traverse this rooted tree T in (any) depth-first order, then the sequence of vertices visited are of alternating color.

Clearly, the OPT bipartite tour contains a bipartite spanning tree where all blue vertices have degree at most 2. Therefore the weight of the minimum-weight bipartite spanning tree whose blue vertices have degree at most 2 is a lower bound on OPT.

Again, if we can find (in polynomial time) the minimum-weight bipartite spanning tree T whose blue vertices have degree at most 2, then a depth first traversal of T with short-cuts will yield a tour whose length is at most twice OPT.

We now claim that the problem of finding T can be viewed as that of finding the minimum-weight, maximum-cardinality subset in the *intersection* of two matroids [29, 15, 16]. The matroids in this case are M_1, the matroid of all bipartite forests, and M_2, the matroid of all bipartite subgraphs whose blue vertices have degree at most 2. For completeness we review the definition of a matroid, following the standard text [29].

DEFINITION. *A* matroid $M = (E, \mathcal{I})$ *is a structure in which* E *is a finite set of elements and* $\mathcal{I}$ *is a family of subsets (called* independent sets*) of* E*, such that* $\phi \in \mathcal{I}$ *and all proper subsets of a set* $I \in \mathcal{I}$ *are in* $\mathcal{I}$*; and if* I_p *and* I_{p+1} *are sets in* $\mathcal{I}$ *containing* p *and* $p+1$ *elements, respectively, then there exists an element* $e \in I_{p+1}$ *such that* $I_p \cup \{e\} \in \mathcal{I}$*.*

An example of a matroid is the *graphic matroid* $M = (E, \mathcal{I})$, where E is the set of edges of an undirected graph, and a subset $I \subset E$ is in $\mathcal{I}$ if and only if I is cycle-free. Another example is the *matrix matroid* $M = (C, \mathcal{I})$, where C is the set of columns of a fixed matrix A, and a subset S of columns is in $\mathcal{I}$ if and only if the columns of S are linearly independent. A maximal-cardinality independent subset of a matroid is called a *base* of a matroid; all bases of a matroid have the same cardinality.

Returning to our problem, let E be the set of all edges that connect red vertices to blue vertices. Let $\mathcal{F}$ denote the collection of all subsets of E that are cycle-free, and let $\mathcal{D}$ denote the collection of subsets S of E such that no more than two edges of S are incident on any blue vertex. Then it is easily shown that $M_1 = (E, \mathcal{F})$ and $M_2 = (E, \mathcal{D})$ are matroids. In addition, the problem of finding *a minimum-weight bipartite spanning tree where the blue vertices have degree at most two* is equivalent to the problem of finding *a minimum-weight common base of* M_1 *and* M_2*.*

This is a special case of the *matroid intersection* problem, which was first solved in polynomial time by Edmonds [15, 16]. Other authors [9] have exploited the special structure of problems such as ours to improve running times. We obtain the following theorem.

THEOREM 2.1. *There is a polynomial-time* 2*-approximation algorithm for the bipartite TSP.*

2.1. Extension to finite capacity vehicles. We will now show how to obtain a constant-factor approximation for the case of arbitrary finite k using the algorithm for the unit capacity case. First, however, we will establish some lower bounds on the optimal solution. Let C_k denote the (length of the) optimal k-delivery tour, and let C^r and C^b denote the (length of the) optimal tours on the red and blue points, respectively. Let A denote the weight of the minimum-weight perfect matching in the bipartite graph with red vertices on one side and blue vertices on the other. To keep the notation simple, we will often use the same symbol to denote a graph and its weight; the context will make it clear which one is intended.

LEMMA 2.2. (a) $A \leq C_1/2$.

(b) $\frac{1}{2k} C_1 \leq C_k$.

Proof. Part (a) is easy to see since C_1 consists of two perfect matchings and each is at least as heavy as A. To see part (b), start with an optimal k-delivery tour C_k: this defines an ordering $r_1, r_2, \ldots, r_n$ on the red points and an ordering $b_1, b_2, \ldots, b_n$ on the blue points. We then construct a 1-delivery tour T starting at the blue vertex b_1 as follows. Consider the blue vertices in the order imposed by C_k, connecting the ith blue vertex b_i to the earliest red vertex in the C_k-ordering that has not already

been connected to a blue vertex; then add another edge connecting this red vertex to the next blue vertex b_{i+1} (if this red vertex is the last one, connect it to the starting blue vertex b_1). By the triangle inequality, each edge e of T is no longer than the sum of the C_k-edges connecting the endpoints of e; we can thus "charge off" each edge of T to a collection of edges of C_k. Since there is never a surplus of more than k blue points in the tour C_k, it follows that no edge of T is charged more than $2k$ times. Thus $T \leq 2kC_k$, from which part (b) follows since $C_1 \leq T$. □

The following lemma is straightforward.

LEMMA 2.3. (a) $C^r \leq C_k$. (b) $C^b \leq C_k$.

We now use the lower bounds just presented to design a constant-factor approximation algorithm for the k-delivery problem. We first use Christofides's heuristic to obtain a 1.5-approximate tour T_r of the red vertices and a 1.5-approximate tour T_b of the blue vertices. Next, we decompose T_r and T_b into paths of k vertices each, by deleting a set of edges that are spaced along the tour at intervals of length k. In fact, there are k such sets of edges in a tour, and we delete the set of maximum weight. It will be convenient to view each k-path as a "supernode" in the following. We now overlay the minimum-weight perfect matching of cost A on this graph. Note that any (red or blue) supernode now has degree exactly k and that there may be several edges between two given supernodes. Thus we obtain a k-regular bipartite multigraph. The following result due to König [26, 31] is crucial to the design of our algorithm:

LEMMA 2.4. *The edges of a d-regular bipartite multigraph can be partitioned into d perfect matchings.*

Using this result, we can partition the perfect matching A into k perfect matchings on the supernodes. We pick the least-weight matching M out of these and delete all other edges of A. Clearly, $M \leq A/k \leq \frac{1}{2k}C_1$. At this stage we have a collection of n/k subgraphs $H_1, H_2, \ldots, H_{n/k}$, each consisting of a red supernode connected via an edge of M to a blue supernode. Now we reintroduce the edges of T_b that were removed when breaking T_b into k-paths; this imposes a cyclic ordering on the subgraphs H_i; let us relabel them $H_1, H_2, \ldots, H_{n/k}$ with this cyclic ordering, where H_1 contains the start blue vertex. We now traverse the subgraphs $H_1, H_2, \ldots, H_{n/k}$ in sequence as follows. Within each subgraph H_i, first visit all the blue vertices and then use the edge of M to go to the red side and visit all the red vertices; then return to the blue side, go to the blue vertex that is connected via an edge e of T_b to the next subgraph H_{i+1}, and use the edge e to go to H_{i+1} (or H_1 if $i = n/k$).

We claim that this tour T is within a constant factor of the optimal k-delivery tour. To verify this, notice that in short-cutting, by triangle inequality, we "charge" each edge of T_b (that was not deleted) no more than three times, each edge of T_r (that was not deleted) no more than two times, and each edge of M at most two times. Notice also that the set of deleted edges in T_b and T_r have total weight at least a $1/k$ fraction of the tour's weight. Thus, we obtain

$$\begin{aligned} T &\leq 3\left(1-\frac{1}{k}\right)T_b + 2\left(1-\frac{1}{k}\right)T_r + 2M \\ &\leq 3\left(1-\frac{1}{k}\right) \times 1.5C^b + 2\left(1-\frac{1}{k}\right) \times 1.5C^r + \frac{2}{2k}C_1 \\ &\leq 7.5\left(1-\frac{1}{k}\right)C_k + 2C_k \\ &\leq 9.5\left(1-\frac{7.5}{9.5k}\right)C_k. \end{aligned}$$

Note that, while for large k the approximation ratio is 9.5, for small k we do much better (e.g., 5.75 for $k = 2$).

THEOREM 2.5. *The above algorithm gives a* $9.5\left(1 - \frac{7.5}{9.5k}\right)$*-approximation to the optimal* k*-Delivery TSP.*

3. Maximum latency TSP. A variant of the following problem arises in the moving points case and is also of independent interest. Given a set of n points $\{p_1, \ldots, p_n\}$ in a metric space, find a path visiting all points, starting at a given point p_0, such that the total *latency* of the points is *maximized*. If in a given path P the length of the ith edge traversed is e_i, then the latency of the jth point visited ($j > 0$) is $L_j = \sum_{i=1}^{j} e_i$ and the total latency $L(P)$ is

$$L(P) = \sum_{j=1}^{n} L_j = \sum_{i=1}^{n} (n - i + 1)e_i.$$

We would like to find a path P for which $L(P)$ is maximized. We can show that this problem is NP-hard by reducing from the maximum Hamiltonian path (MaxHP) problem. A related problem, *minimum* latency TSP, has been addressed in [7, 22] where constant-factor approximations are obtained.

We show that the greedy strategy of at any stage, visiting the farthest unvisited point from the current point achieves a total latency at least half that of the maximum latency path. (In fact, it can also be shown from our proof that the greedy path has *length* at least half that of the MaxHP from the starting point p_0. This result was obtained previously by Fisher, Nemhauser, and Wolsey [17].) The key observation is the following lemma.

LEMMA 3.1. *Let* G_i *be the length of the first* i *edges in the greedy path starting from* p_0*. Let* P_i *be the length of the maximum* i*-path, i.e., the longest path that visits* i *vertices from* p_0*. Then for* $i \leq n$*,* $G_i \geq P_i/2$*.*

Proof. For brevity, paths/edges and their lengths are denoted by the same symbols. Consider the maximum matching M in the maximum i-path P_i, i.e., the maximum-length collection M of independent edges from P_i. Let the (lengths of) edges of M be $m_1 \geq m_2 \geq \cdots \geq m_k$. Note that $P_i \leq 2(m_1 + m_2 + \cdots + m_k)$. We will argue that $G_i \geq (m_1 + m_2 + \cdots + m_k)$.

Consider the edges in G_i as being *directed* in the direction of travel, starting from p_0. Call an edge m of M an *anchor* if there is an edge of G_i that *starts* at an endpoint of m; the *earliest* such edge g of G_i is said to be *anchored* at m.

CLAIM A. *If* g *is anchored at* m*, then* $g \geq m$*, since* G_i *is greedy and the other endpoint of* m *has not yet been visited at the time* g *was traversed by the greedy path.*

CLAIM B. *If* $m \in M$ *is not an anchor, then* every *edge of* G_i *has length at least* $m/2$*. To see this, note that neither endpoint of* m *is visited by* G_i *(except possibly at the end). By the triangle inequality, from any point* p*, at least one of the endpoints of* m *is at distance at least* $m/2$*. Since* G_i *is greedy, its edges have length at least* $m/2$*.*

Now suppose $m_1, m_2, \ldots, m_u$ are anchors and m_{u+1} is not; i.e., m_{u+1} is the heaviest edge of M that is *not* an anchor. (Note that u could be 0.) Let $g_1, g_2, \ldots, g_u$ be the corresponding anchored edges of G_i. Then Claims A and B imply that $g_1 + g_2 + \cdots + g_u \geq m_1 + m_2 + \cdots + m_u$ and that every edge of G_i has length at least $m_{u+1}/2$. There are now two cases to be considered. Recall that $k = |M|$.

Case 3.1 ($u = 0$ and $i = 2k - 1$). If $u = 0$, then m_1 is not an anchor, so all edges of G_i have length at least $m_1/2$. Also $i = 2k - 1$ implies that the starting point p_0

must occur in the matching M. Thus, the first edge of G_i must be anchored at an edge m' of M, and the total length of G_i is at least

$$m' + \frac{(i-1)m_1}{2} \;\geq\; m' + \frac{(2k-2)m_1}{2} \;=\; m' + (k-1)m_1 \geq m_1 + m_2 + \cdots + m_k).$$

Case 3.2 ($u \geq 1$ or $i \geq 2k$). Note that the total length L of unanchored edges of G_i is at least $(i-u)m_{u+1}/2$. If $u \geq 1$,

$$L \geq \frac{(2k-1-u-(u-1))m_{u+1}}{2} = (k-u)m_{u+1},$$

and if $i \geq 2k$,

$$L \geq \frac{(2k-u-u)m_{u+1}}{2} = (k-u)m_{u+1}.$$

Thus, G_i is at least $(m_1+m_2+\cdots+m_u)+(k-u)m_{u+1} \geq m_1+m_2+\cdots+m_k$. □

Clearly the latency of the ith point in the maximum latency path is at most P_i, so the lemma implies that the total latency of the greedy path is at least half that of the maximum latency path. Since P_n is the MaxHP, it also follows that the greedy path is a $\frac{1}{2}$-approximation of the MaxHP. Lemma 3.1 implies our result.

THEOREM 3.2. *The greedy strategy of always visiting the farthest unvisited point achieves a total latency at least half that of the maximum latency path.*

We can also show that the greedy heuristic works well for a different "latency measure" that arises implicitly in dynamic k-collect TSP.

THEOREM 3.3. *Let $\alpha < 1$ be a positive constant. For a Hamiltonian path P starting at p_0, define the cost $L_\alpha(P) = \sum_{i=1}^{n-1}(1-\alpha^{n-i})e_i$. The greedy heuristic produces a path whose L_α cost is at least $(1-\alpha)/2$ that of the maximum-L_α path.*

Proof. Let G denote the greedy path and H^* denote the MaxHP. Let P^*_α denote the path that maximizes $L_\alpha(.)$ and P^* be the path that maximizes $L(.)$. Since $(1-\alpha^{n-i}) < 1$ for $i = 1, 2, \ldots, n-1$, we have

$$L_\alpha(P^*_\alpha) \leq length(P^*_\alpha) \leq length(H^*),$$

and since $(1-\alpha^{n-i}) \geq 1-\alpha$ for $i = 1, 2, \ldots, n-1$, we have

$$L_\alpha(G) \geq (1-\alpha)length(G).$$

The desired result then follows from the fact that $length(G) \geq length(H^*)/2$ (Theorem 3.2). □

4. TSP for moving-points: Dynamic *k*-collect TSP. The dynamic k-collect TSP (defined in the introduction) is inspired by the following application in industrial robotics. After manufacture, parts are dumped onto a conveyor belt in arbitrary positions and orientations. Prior to packaging (or assembly), the parts must be collected by a robot arm of capacity k and delivered to an empty pallet at a fixed location (not on the belt) that can be treated as the origin of the coordinate space. Once filled, the pallet is moved away by another conveyor belt, and a new empty pallet appears at the origin. The general scenario of designing configurations and algorithms for robots working in their cells to handle parts as they come down a conveyor belt has been termed *rapid deployment automation* [4, 10, 30].

For convenience, we switch to the L_1 metric, and assume that the robot translates only parallel to the x- and y-axes. While there are situations where this applies

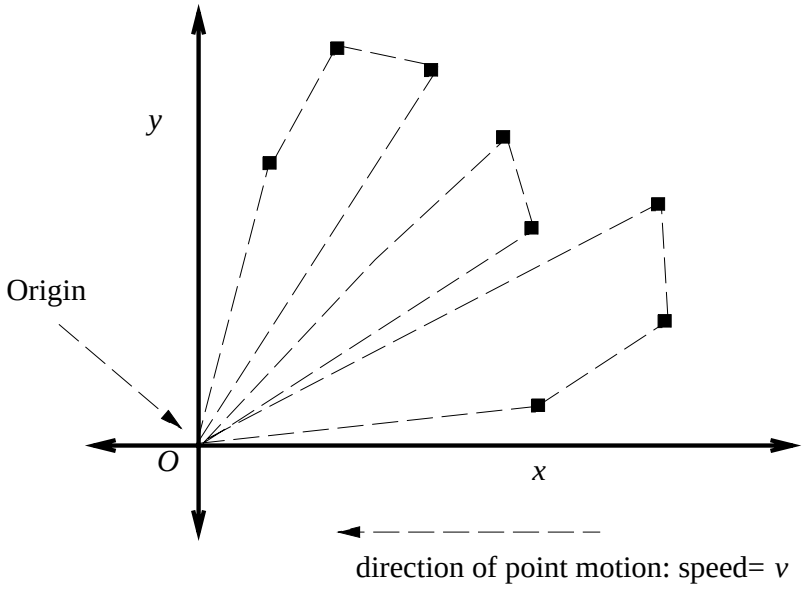


FIG. 2. *Illustrating the moving-points model. A* 3*-collect tour is shown.*

directly, it is also easy to see that this causes an error of only factor $\sqrt{2}$ with respect to the L_2 metric. We assume that the points $p_1, p_2, \ldots, p_n$ are always within the positive quadrant of the coordinate frame centered at the origin $p_0 = (0, 0)$ (see Figure 2). The robot moves with speed 1, and the belt (and each point p_i) moves with a velocity v directed in the negative x-direction. The y-axis represents the end of the conveyor belt, and to obtain meaningful results we must assume at the very least that v is suitably bounded below 1, since a slow robot may be unable to catch up with some points. In fact, to prove our approximation results it is sufficient to assume that $v \leq \frac{k}{2n}$, which is necessary to ensure that no p_i crosses the y-axis while the robot is in the process of executing the tour. *We will assume this upper bound on* v *throughout the rest of this paper.* Also, we define $\alpha = \frac{1-v}{1+v}$.

Some remarks are in order about our restriction to the case where the moving points' velocities are sufficiently smaller than that of the robot arm. As further explicated in section 4.6, this is essential to ensure that the robot arm is able to retrieve all objects and is certainly a valid assumption in the motivating application. An interesting variant, which we do not explore here, concerns the model where the velocities are unrestricted and the goal is to maximize the number of points visited, with possible restrictions on the total time available. Clearly, this would involve generalizing the *prize-collecting TSP* [5] to the case of moving points.

For clarity, we refer to a fixed point in space (such as the origin) as a *space-point*, to distinguish it from a *moving-point*, p_i. Define (x_i, y_i) as the coordinates of point p_i at time 0, and let $d_i = x_i + y_i$ denote the L_1 distance of p_i from the origin at time 0. Clearly the x-coordinate of p_i at time t is $x_i - vt$ and the y-coordinate doesn't change. The distance of p_i from the origin at time t is therefore $x_i + y_i - vt = d_i - vt$.

4.1. Shortest paths and triangle inequality. We establish several basic properties of travel times that are not immediate. Henceforth, when we say that the robot moves from a point A to a space-point B or meets a moving-point p_i, we will assume

that the robot takes the shortest-time path. What is the quickest way for a robot to meet a moving-point p? The following lemma characterizes such paths. A robot path is said to be *monotone* if no two points on the path have the same x-coordinate or the same y-coordinate.

LEMMA 4.1 (shortest paths). *Suppose the robot is at a space-point A and meets moving-point p at the earliest possible time, say, at space-point B. Then the robot's path from A to B is necessarily a shortest path between those points, and the robot never stops at any time before it meets p.*

Proof. At any time the robot may "sit and wait," move parallel to the x-axis, or move parallel to the y-axis. Suppose the robot meets point p after time T. If the robot's path to B is not the shortest path to B (from A) or the robot stopped at some time, this means that the robot could have arrived at B at an earlier time $T' < T$ (by either using a shorter path to B or not waiting along the way). At this time T' the moving-point p must be to the right of B a distance $(T - T')v$ away on the same y-coordinate, and the robot can meet p in time $(T - T')\frac{v}{1+v}$ by moving toward it. Thus, the total time to meet p could be $T' + \frac{v}{1+v}(T - T')$, and the time saved would be $(T - T') - \frac{v}{1+v}(T - T') = \frac{1}{1+v}(T - T')$, which is positive. This contradicts our assumption that the robot met p at the earliest possible time. □

We have the following important corollaries of the above lemma.

COROLLARY 4.2 (monotone path). *Suppose the robot moves from space-point A to meet a moving-point p. Then the robot's path must be monotone. In particular a quickest way for the robot to meet p is to first move to the y-coordinate of p and then move toward p.*

Proof. The following fact will be useful in this and other proofs about shortest paths.

FACT 4.3. *The shortest (and therefore least travel-time) path for the robot to move from a given space-point A to a given space-point B is a monotone path; in particular the monotone path that first moves to the y-coordinate of B and then to the x-coordinate of B is shortest.*

The monotonicity follows from Lemma 4.1 and Fact 4.3. Any monotone path that meets p can be replaced by a path where all the x-motion is done after the y-motion, without changing the rendezvous time or coordinates. Clearly the y-motion consists simply of moving to the y-coordinate of p. At this time p may either be left or right of the robot. In case p is left of the robot, by monotonicity the quickest way to meet p is to move toward p. In case p is to the right of the robot, if the robot moves to the left, then either the path becomes nonmonotone before it meets p or it must stop and wait for p to catch up with it, both of which are not possible in a shortest path, by Lemma 4.1. Thus, in this case also the robot must move toward p. □

COROLLARY 4.4 (triangle inequality 1). *If the moving-points p_i and p_j are in the positive quadrant and the robot is at p_i, then the time t_{ij} to travel directly to p_j is bounded by the time to travel to p_j via the origin.*

Proof. If the composition of the robot's path from p_i to the origin and the path from the origin to p_j is not monotone, then by Corollary 4.2 it cannot be shorter than the shortest p_i–p_j path; if it is monotone, then it cannot be shorter than the particular shortest p_i–p_j path described in Lemma 4.2. □

COROLLARY 4.5 (triangle inequality 2). *Let t_{ij} denote the shortest time of travel from moving-point p_i to moving-point p_j. Then for any three points p_i, p_j, p_ℓ, $t_{ij} + t_{j\ell} \geq t_{i\ell}$*

4.2. Useful time expressions. The following time expressions are easily verified; we will use them in the proofs in this paper. In all of the following keep in mind that d_i denotes the distance of p_i from the origin at time 0.

- *Origin to moving-point.* The robot is at the origin at time τ and meets moving-point p_i. The earliest rendezvous time is

$$\tau + \frac{d_i - v\tau}{1+v} = \frac{\tau + d_i}{1+v}. \tag{1}$$

- *Moving-point to origin.* At time τ the robot is at a moving-point p_i, and it then returns to the origin. The time of return to the origin is

$$\tau + d_i - v\tau = d_i + (1-v)\tau. \tag{2}$$

- *Moving-point to origin to moving-point.* At time τ the robot is at a moving-point p_i; it then returns to the origin and goes to moving-point p_j. From (2), the time of arrival at the origin is $d_i + (1-v)\tau$, and during this time point p_j moves closer to the origin by a distance $v(d_i + \tau(1-v))$. Thus the time of arrival at p_j is

$$d_i + \tau(1-v) + \frac{d_j - v[d_i + \tau(1-v)]}{1+v} = \frac{d_i + d_j}{1+v} + \tau\frac{1-v}{1+v}. \tag{3}$$

4.3. Optimal rendezvous. We can use the above characterization to derive some expressions for the shortest time (or distance) needed to meet a moving-point. Suppose at time t_0 the robot is at space-point $A = (x_1, y_1)$, and moving-point p is at (x_2, y_2). Let $x = |x_1 - x_2|$ and $y = |y_1 - y_2|$. If the robot moves and meets moving-point p at the earliest possible time $t_0 + t$, then t equals one of the following. In view of Lemma 4.2 we may assume that the robot first moves to the y-coordinate of p and then meets p by moving parallel to the x-axis.

- If p is to the right of A at time t_0 and $vy \leq x$, then p remains to the right of the robot when it reaches the y-coordinate of p, so

$$t = y + \frac{x - vy}{1+v} = \frac{x+y}{1+v}. \tag{4}$$

- If p is to the right of A at time t_0 and $vy > x$, then moving-point p will be to the left of the robot by the time the robot reaches the y-coordinate of p, so

$$t = y + \frac{vy - x}{1-v} = \frac{y-x}{1-v}. \tag{5}$$

- If p is to the left of A at time t_0, then

$$t = y + \frac{vy + x}{1-v} = \frac{x+y}{1-v}. \tag{6}$$

4.4. Robot arms with capacity 1. In the remainder of the paper we write C_k to denote the (length of the) optimal dynamic k-collect tour.

In dynamic 1-collect TSP, the robot must visit the points one at a time, returning to the origin after visiting each point. We prove the following theorem.

THEOREM 4.6. *For dynamic* 1*-collect TSP, the minimum-time tour (under the* L_1 *metric) visits moving-points in decreasing order of their distance from the origin at time* 0.

Proof. Suppose that the robot visits points $p_1, \ldots, p_n$ in that order. We derive an expression for the total time $T^{(1)}$ taken by a 1-collect tour. Let T_m denote the time taken by the tour after it has visited m points and returned to the origin. We show by induction on m that

$$T_m = \frac{2}{1+v}\left(\sum_{i=1}^{m} \alpha^{m-i} d_i\right),$$

where we recall that $\alpha = \frac{1-v}{1+v}$. The base case, $m = 1$, follows from an application of (1), using $\tau = 0$ and $i = 1$; note that the factor of 2 comes from the requirement that the robot returns to the origin.

For the induction step, we assume that

$$T_{m-1} = \frac{2}{1+v} \sum_{j=1}^{m-1} \alpha^{m-(i+1)} d_i.$$

Suppose that at time T_{m-1} the robot has just returned to the origin after visiting the point p_{m-1} and that now it is ready to go visit point p_m. The time taken to reach p_m is given by an application of (1) with $\tau = T_{m-1}$ and $i = m$, which equals

$$\frac{d_m}{1+v} - \frac{vT_{m-1}}{1+v}.$$

The total time required is given by

$$\begin{aligned} T_m &= T_{m-1} + 2\left(\frac{d_m}{1+v} - \frac{vT_{m-1}}{1+v}\right) \\ &= \left(1 - \frac{2v}{1+v}\right) T_{m-1} + \left(\frac{2}{1+v} d_m\right) \;=\; \alpha T_{m-1} + \frac{2}{1+v} d_m \\ &= \frac{2}{1+v}\left(\sum_{i=1}^{m-1} \alpha^{m-(i+1)+1} d_i\right) + \frac{2}{1+v} d_m \;=\; \frac{2}{1+v}\left(\sum_{i=1}^{m} \alpha^{m-i} d_i\right). \end{aligned}$$

So the total time $T^{(1)}$ taken by a 1-collect tour is given by

$$T^{(1)} = \frac{2}{1+v} \sum_{i=1}^{n} \alpha^{n-i} d_i. \tag{7}$$

The theorem then follows. □

Thus the optimal dynamic 1-collect tour has some aspects of a maximum latency tour. In fact the maximum latency problem is implicit in the dynamic k-collect problem for arbitrary finite k. We make this formal in section 4.7 where we introduce the notion of geometric latency of a tour.

4.5. Robot arms with infinite capacity. In dynamic k-collect TSP with $k = \infty$, the robot must visit all the n moving-points before returning to the origin. For the optimal tour, suppose that the last moving-point visited is p and say that p is at

distance d from the origin at time 0. If the tour reaches p at time T_p, then by (2) the total time taken is $C_\infty = d + (1-v)T_p$.

Our strategy for approximating C_∞ is to "guess" the last point p visited by the optimal tour (there are only n possibilities for p) and approximate the minimum-length path T_p^* from the origin to p that visits every other moving-point before visiting p (i.e., the minimum Hamiltonian path from the origin to p).

The problem of approximating T_p^* can be set up as an asymmetric TSP instance on a graph with $n+1$ vertices $v_0, v_1, \ldots, v_n$, where v_0 represents the origin and v_i the moving-point p_i. The directed distance $d(v_i, v_j)$ is defined to be the appropriate distance among (4), (5), and (6). Note that the ratio of the two directed distances between a given pair of vertices is bounded by either $(1+v)/(1-v)$ or $(y+x)/(y-x)$, where in the second case $vy > x$, i.e., $x/y < v < 1$. Thus, the ratio never exceeds $1/\alpha$. The best-known approximation algorithm [28] for the minimum Hamiltonian path between two specified vertices for a *symmetric* distance matrix has a ratio $\frac{5}{3}$. Therefore, using this algorithm, we can approximate T_p^* to within a factor $\frac{5}{3\alpha}$ and thereby approximate C_∞ to within the same ratio.

THEOREM 4.7. *There is a $\frac{5}{3\alpha}$-approximation algorithm for dynamic k-collect TSP with $k = \infty$.*

4.6. Robot arms with finite capacity k. We now consider dynamic k-collect TSP. We derive an expression for the time $T^{(k)}$ taken by a k-collect tour. Let $m = n/k$ denote the number of returns to the origin. We note that when the arm capacity is k, *at least* n/k returns to the origin are required, and it may be necessary to return to the origin more often to minimize the tour length. However, it is easy to show that the assumption of exactly n/k returns affects our results by a factor of at most 2. Also, in our motivating application, the robot may be required to pick exactly k objects on each excursion from the origin, which would justify our assumption of exactly n/k returns.

The sequence of edges traversed can be viewed as

$$(p_0, p_{v(0)})\ \ L_1\ \ (p_{u(1)}, p_0), (p_0, p_{v(1)})\ \ L_2\ \ (p_{u(2)}, p_0), (p_0, p_{v(2)})\ \ L_3, \ldots, L_m\ \ (p_{u(m)}, p_0),$$

where each L_i denotes the sequence of edges involving nonorigin points in the ith excursion from the origin. We abuse notation and denote by L_i the time spent traversing the corresponding edges. Let $D_0 = d_{v(0)}$, $D_m = d_{u(m)}$, and, for $2 \le i \le m-1$, $D_i = d_{u(i)} + d_{v(i)}$.

Notice first that the time till the end of L_1 is $T_1 = D_0/(1+v) + L_1$ and that the time to the end of L_2 (from (1)) is

$$T_2 = \alpha\left(\frac{D_0}{1+v} + L_1\right) + \frac{D_1}{1+v} + L_2.$$

If we were to return to the origin after L_2 (and d denotes the time-0 distance of the end of L_2 to the origin) the total time would be

$$T' = d + (1-v)T_2 = D_0\alpha^2 + D_1\alpha + d + (1-v)[L_1\alpha + L_2].$$

Generalizing this gives the following expression for $T^{(k)}$:

$$T^{(k)} = D_0\alpha^m + D_1\alpha^{m-1} + \cdots + D_{m-1}\alpha + D_m + (1-v)[L_1\alpha^{m-1} + L_2\alpha^{m-2} + \cdots + L_m]. \tag{8}$$

It might appear from (8) that $T^{(k)} \to 0$ as $\alpha \to 0$. However, (8) is valid only if at all times, all unvisited moving points remain to the right of the origin (i.e., in the positive quadrant). In the moving belt scenario, the y-axis represents the end of the belt. A small α corresponds to a v close to 1, whereas the problem is meaningful only if v is small enough that the points do not cross the y-axis when the robot is in the process of grasping them. A reasonable restriction on v is that the time for an optimal k-collect TSP tour should not suffice for any point to cross the origin. This implies that $C_1/k \leq d_{av}/v$, where d_{av} denotes the average of the distances d_i and that C_1/k is a lower bound on the optimal in the static problem (see for instance [1]). Thus, we will assume that $v(2d_{av}n)/k \leq d_{av}$, i.e., $v \leq \frac{k}{2n}$. This bound on v, although sufficient to enable us to prove our approximation results, may not be sufficient to ensure that the points do not cross the y-axis while the robot arm is visiting them. For instance, if the points are very close to the y-axis or very far from the x-axis, a tighter bound on v is needed. Nevertheless, we point out that in practical applications, the bound is close to the correct one since (a) the conveyor belt is of a fixed width, so no point is too far from the x-axis, and (b) the parts initially appear at some reasonable distance from the y-axis.

Given that $v \leq \frac{k}{2n}$, in the expression (8) for $T^{(k)}$, the smallest coefficient on any term is

$$\alpha^m = \alpha^{n/k} \geq \frac{(1 - \frac{k}{2n})^{n/k}}{(1 + \frac{k}{2n})^{n/k}} \geq \frac{1}{e}.$$

This implies that $T^{(k)} \geq \frac{1}{e}(D_0 + L_1 + D_1 + L_2 + D_2 + \cdots + L_m + D_m)$.

Let us denote the sum of terms in the parentheses by T'. Minimizing T' is a static k-collect TSP problem on $n+1$ vertices $v_0, v_1, \ldots, v_n$, where the directed distances are defined as in the C_∞ approximation, except that for all $i > 0$, $d(v_0, v_i) = d_i = d(v_i, v_0)$. As we showed before, the ratio of the two directed distances between a pair of vertices never exceeds $1/\alpha$, so the k-collect TSP approximation algorithm of [1] can be used to approximate T' to within a factor $2.5/\alpha$ of optimal. Thus, we can approximate the optimal k-collect tour time C_k to within a factor $2.5e/\alpha$.

THEOREM 4.8. *For $v \leq k/2n$, there is a $(2.5e/\alpha)$-approximation algorithm for dynamic k-collect TSP.*

The following lower bound, although not used in this paper, may lead to a different approximation algorithm for C_k than the one we presented above.

THEOREM 4.9. *For any finite k, $C_\infty \leq C_k$.*

Proof. Consider the optimal tour C_k for capacity k. (We will abuse notation and use the same symbol for a tour as for its length.) Let d be the time-0 distance from the origin of the final moving-point p visited by C_k before returning to the origin, and let T_p be the time taken by T up to this final point p. From (2), the total time taken is $C_k = T_p(1 - v) + d$. Note that before reaching the final point p, T may do the following several times: go from a moving-point p_i to the origin and then to moving-point p_j. However by the Triangle Inequality I (Corollary 4.4), a direct path from p_i to p_j is no longer than the path via the origin, so we can short-circuit each such indirect path by a direct path and gain time. By applying these short-circuits to every origin-return except the last one, we can modify the optimal tour C_k to an infinite-capacity tour that reaches p at an earlier time $T'_p \leq T_p$, and the total time of this tour would be $(1 - v)T'_p + d$ which is no larger than C_k. □

4.7. Geometric latency. The somewhat unwieldy expression (8) for $T^{(k)}$ can be lower-bounded by a more pleasant cost expression, which we call the *geometric*

latency of a tour. We describe this below and show how a variant of the *maximum* latency problem arises implicitly in minimizing the geometric latency.

Consider the asymmetric k-collect TSP problem with directed distances as defined in section 4.6. Now consider a capacity-k tour on this graph and let $e_1, e_2, \ldots, e_u$ be the sequence of (weights of) edges traversed. Fix some positive $\beta < 1$, and define the *geometric latency* G_β of this tour to be $G_\beta = \sum_{i=1}^{u} \beta^{u-i} e_i$. Assuming for convenience that n is a multiple of k, $u = m + n$.

LEMMA 4.10. *Let $\beta = \alpha^{1/k}$ and fix a capacity-k tour of length $T^{(k)}$ for the moving-points problem. Let G_β be the geometric latency of the corresponding (i.e., same order of visiting points) tour in the corresponding asymmetric k-collect TSP instance as described above. Then $T^{(k)} \geq G_\beta$.*

Proof. Consider expression (8) for $T^{(k)}$. Since $(1 - v) \geq \alpha$,

$$T^{(k)} \geq (D_0 + L_1)\alpha^m \ + \ (D_1 + L_2)\alpha^{m-1} \ + \ \cdots \ + \ (D_{m-1} + L_m)\alpha \ + \ D_m.$$

The lemma follows by comparing weights of corresponding terms in this expression and the one for G_β. ∎

Thus, if we are able to find a constant-factor approximation to a capacity-k tour with minimum geometric latency G_β, we would have a constant factor approximation for the dynamic k-collect TSP problem. We know of no such algorithm that runs in polynomial time. It is worth noting that G_β may be rewritten as

$$G_\beta = \sum_{i=1}^{u} e_i - \sum_{i=1}^{u-1} (1 - \beta^{u-i} e_i),$$

so minimizing G_β roughly involves simultaneously minimizing the total tour length and maximizing the second term, which is a variant of the linear latency. As we mentioned in section 3, we can approximately maximize the second term, but we do not know how to simultaneously bound the length of the tour.

4.8. Extensions. In conclusion, we briefly mention some easy extensions of the model and results for the moving-point scenario. We omit the details.

- The first extension is to the case where all points are moving *away* from the origin at velocity v, as would be the case in midair refueling of a formation of planes. The results and analysis are similar to that presented above.
- The second extension is to the case where the conveyor belt is circular and is rotating around the origin. We can obtain a constant-factor approximation for the case where the rotation speed is bounded below the robot's speed.

Acknowledgments. We are grateful to Alan Frieze for suggesting the bipartite TSP and for his encouragement. We are deeply indebted to Ken Goldberg and Anil S. Rao for describing the robotics application and the dynamic TSP problem, and for their valuable feedback and constant encouragement during the course of this work. Some of the results reported in this paper are based on joint work with Anil Rao [11].

REFERENCES

[1] K. ALTINKEMER AND B. GAVISH, *Heuristics for delivery problems with constant error guarantees*, Transportation Science, 24 (1990), pp. 294–297.
[2] S. ANILY AND R. HASSIN, *The swapping problem*, Networks, 22 (1992), pp. 419–433.
[3] S. ANILY AND G. MOSHEIOV, *The traveling salesman problem with delivery and backhauls*, Oper. Res. Lett., 16 (1994), pp. 11–18.

[4] M. J. Atallah and S. R. Kosaraju, *Efficient solutions to some transportation problems with applications to minimizing robot arm travel*, SIAM J. Comput., 17 (1988), pp. 849–869.
[5] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala, *Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen*, in Proceedings of the 27th Annual ACM Symposium on the Theory of Computing, Las Vegas, NV, 1995, pp. 277–283.
[6] L. Bianco, A. Mingozzi, S. Riccardelli, and M. Spadoni, *Exact and heuristic procedures for the traveling salesman problem with precedence constraints, based on dynamic programming*, INFOR, 32 (1994), pp. 19–32.
[7] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, *The minimum latency problem*, in Proceedings of the 26th Annual ACM Symposium on the Theory of Computing, Montreal, Quebec, Canada, 1994, pp. 163–171.
[8] A. Blum, P. Chalasani, and S. Vempala, *A constant-factor approximation for the k-MST problem in the plane*, in Proceedings of the 27th Annual ACM Symposium on the Theory of Computing, Las Vegas, NV, 1995, pp. 294–302.
[9] C. Brezovec, G. Cornuejols, and F. Glover, *A matroid algorithm and its application to the efficient solution of two optimization problems on graphs*, Math. Programming, 42 (1988), pp. 471–487.
[10] B. Carlisle and K. Y. Goldberg, *Report on TARDA*, in Symposium on Theoretical Aspects of Rapid Deployment Automation, Adept Technology, San Jose, CA, 1994.
[11] P. Chalasani, R. Motwani, and A. Rao, *Approximation algorithms for robot grasp and delivery*, in Proceedings of the 2nd International Workshop on Algorithmic Foundations of Robotics, Toulouse, France, 1996, pp. 347–362.
[12] M. Charikar, S. Khuller, and B. Raghavachari, *Algorithms for capacitated vehicle routing*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 349–358.
[13] N. Christofides, *Worst-case analysis for a new heuristic for the traveling salesman problem*, in Symposium on New Directions and Recent Results in Algorithms and Complexity, J. F. Traub, ed., Academic Press, New York, 1976.
[14] N. Christofides, *Vehicle routing*, in The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, eds., John Wiley, New York, 1985, pp. 431–448.
[15] J. Edmonds, *Submodular functions, matroids and certain polyhedra*, in Combinatorial Structures and Their Applications, Proceedings of Calgary International Conference, Calgary, AB, Canada, 1970, pp. 69–87.
[16] J. Edmonds, *Matroid intersection*, Ann. Discrete Math., 4 (1979), pp. 39–49.
[17] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, *An analysis of approximations for finding a maximum weight Hamiltonian circuit*, Oper. Res., 27 (1979), pp. 799–809.
[18] K. R. Fox, B. Gavish, and S. C. Graves, *An n-constraint formulation of the (time-dependent) traveling salesman problem*, Oper. Res., 28 (1980), pp. 1018–1021.
[19] G. N. Frederickson, M. Hecht, and C. Kim, *Approximation algorithms for some routing problems*, SIAM J. Comput., 7 (1978), pp. 178–193.
[20] A. M. Frieze, *An Extension of Christofides' Heuristic to the k-person traveling salesman problem*, Discrete Appl. Math., 6 (1983), pp. 79–83.
[21] N. Garg and D. S. Hochbaum, *An $O(\log k)$ approximation algorithm for the k minimum spanning tree problem in the plane*, in Proceedings of the 26th Annual ACM Symposium on the Theory of Computing, Montreal, Quebec, Canada, 1994, pp. 432–438.
[22] M. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, in Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1996.
[23] B. L. Golden and A. A. Assad, eds., *Vehicle Routing: Methods and Studies*, North-Holland, Amsterdam, 1988.
[24] P. Hall, *On representatives of subsets*, J. London Math. Soc., 10 (1935), pp. 26–30.
[25] K. Jansen, *Bounds for the general capacitated routing problem*, Networks, 23 (1993), pp. 165–173.
[26] D. Konig, *Über graphen und ihre andwendung auf determinantentheorie und mengenlehre*, Math. Ann., 77 (1916), pp. 453–465.
[27] M. Kubo and H. Kagusai, *Heuristic algorithms for the single-vehicle dial-a-ride problem*, J. Oper. Res. Soc. Japan, 30 (1990), pp. 354–365.
[28] J. A. Hoogeveen, *Analysis of Christofides' heuristic: Some paths are more difficult than cycles*, Oper. Res. Lett., 10 (1991), pp. 291–295.
[29] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Reinhart and Winston, New York, 1976.

[30] T-Y. Li and J-C. Latombe, *On-line manipulation planning for two robot arms in a dynamic environment*, in Proceedings of the 12th Annual IEEE International Conference on Robotics and Automation, 1995, pp. 1048–1055.

[31] L. Lovász and M. Plummer, *Mathing Theory*, North-Holland Math. Stud. 29, North-Holland, Amsterdam, 1986.

[32] A. Lucena, *Time-dependent traveling salesman problem—the deliveryman case*, Networks, 20 (1990), pp. 753–763.

[33] J. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-MST problem*, in Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1996.

[34] H. Psaraftis, *Scheduling large-scale advance-request dial-a-ride systems*, Amer. J. Math. Management Sci., 6 (1986), pp. 327–367.

[35] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi, *Spanning trees short or small*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA, 1994, pp. 546–555.

[36] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, *An analysis of several heuristics for the traveling salesman problem*, SIAM J. Comput., 6 (1977), pp. 563–581.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2150–2167

ON FLOOR-PLAN OF PLANE GRAPHS*

XIN HE†

Abstract. A floor-plan is a rectangle partitioned into a set of disjoint rectilinear polygonal regions (called *modules*). A floor-plan F represents a plane graph G as follows: Each vertex of G corresponds to a module of F and two vertices are adjacent in G iff their corresponding modules share a common boundary. Floor-plans find applications in VLSI chip design.

If a module M is a union of k disjoint rectangles, M is called a k-rectangle module. It was shown in [K.-H. Yeap and M. Sarrafzadeh, *SIAM J. Comput.*, 22 (1993), pp. 500–526] that every triangulated plane graph G has a floor-plan using 1-, 2-, and 3-rectangle modules. In this paper, we present a simple linear time algorithm that constructs a floor-plan for G using only 1- and 2-rectangle modules.

Key words. algorithm, plane graph, rectangular dual, floor-plan

AMS subject classifications. 05C10, 05C75, 05C85, 68Q25, 68Q35, 68R10

PII. S0097539796308874

1. Introduction.

Floor-planning is an early step in VLSI chip design where one decides the relative location of functional entities on a chip (see [25] and the references cited within). The most immediate representation of a floor-plan is a partition of a rectangular chip area into modules (usually rectilinear polygons) where each module represents a functional entity. In the floor-planning process, we are given a graph $G = (V, E)$. Each vertex of G represents a functional entity. The edges of G represent the adjacency requirements between the functional entities. We want to partition a rectangular chip area into a set of rectilinear polygonal modules and map each vertex of G to a module such that two vertices are adjacent in G iff their corresponding modules share a common boundary. If G has such a floor-plan, clearly G must be a plane graph.

For simplicity, most floor-planning systems are restricted to using rectangular modules. (In [25], a rectangular module is called a 0-concave rectilinear module or 0-CRM since it has no concave corners). In the literature, a floor-plan of G using only rectangular modules is called a *rectangular dual* of G. The problem of constructing rectangular duals has been extensively studied. The necessary and sufficient conditions for G to have a rectangular dual were given in [14, 15, 16]. An algorithm for solving this problem was developed in [1, 2]. Although it runs in linear time, this algorithm is fairly complicated. A simple linear time algorithm for solving the rectangular dual problem was developed in [7, 12, 13]. The coordinates of the rectangular duals constructed by this algorithm are integers and carry clear combinatorial meaning. A parallel implementation of this algorithm, working in $O(\log^2 n)$ time with $O(n)$ processors, was given in [8]. A necessary and sufficient condition for a plane graph to have a *rectangular drawing* was given in [22]. Based on this characterization, a linear time algorithm to obtain a rectangular drawing of a plane graph was given in [18]. A rectangular dual of a plane graph G can be obtained by applying this algorithm

*Received by the editors September 9, 1996; accepted for publication (in revised form) September 15, 1998; published electronically July 7, 1999. A preliminary version of this paper appeared in the Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, 1997, pp. 426–435.
http://www.siam.org/journals/sicomp/28-6/30887.html

†Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY 14260 (xinhe@cs.buffalo.edu). This work was partially supported by National Science Foundation grant CCR-9205982.

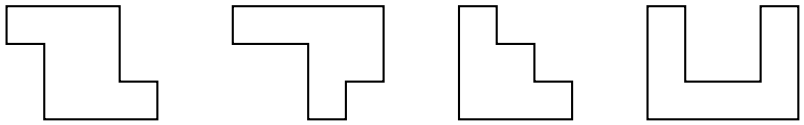


FIG. 1. *Rectilinear modules with* 2 *concave corners.*

to the dual graph of G. The applications of rectangular duals were also discussed in [5, 9, 17].

However, not every plane graph G has a rectangular dual. If G has a triangle (a cycle with three edges) that encloses some vertices in its interior (such triangles will be called *separating triangles* and were called *complex triangles* in [25]), then G cannot have a rectangular dual because at least four rectangles are needed in order to enclose a nonempty area on the plane.

The problem of constructing floor-plans for general plane graphs (which may have separating triangles) was investigated in [19, 25]. If G has separating triangles, any floor-plan of G must use nonrectangular modules. In the applications of floor-plans, it is desirable to use modules whose shapes are as simple as possible [25]. This raises a natural question: How complex must these modules be? In [19, 25], the number of concave corners of a module M is taken as its complexity measure. If M has k concave corners, it is called a k-CRM in [25]. Thus a 0-CRM is just a rectangle. A 1-CRM is an L-shaped module. The possible shapes of 2-CRMs are shown in Figure 1. The necessary and sufficient conditions for G to have a floor-plan consisting of only 0- and 1-CRMs were given in [19]. It was shown in [25] that, in order to design a floor-plan for an arbitrary plane graph, it is both necessary and sufficient to use 0-, 1-, and 2-CRMs.

A more natural measure on the complexity of a module M is the minimum number of disjoint rectangles whose union forms M. If M is the union of k disjoint rectangles, we call M a *k-rectangle module*. In particular, a 1-rectangle module is a rectangle. Some 2-CRMs shown in Figure 1 are 3-rectangle modules. Thus the results in [25] state that we can always construct a floor-plan of an arbitrary plane graph G using only 1-, 2-, and 3-rectangle modules. As mentioned above, the floor-plans of plane graphs with separating triangles must use k-rectangle modules for $k > 1$. The remaining question is: Can we always find a floor-plan for an arbitrary plane graph G using only 1- and 2-rectangle modules? In this paper, we answer this question affirmatively. We present a linear time algorithm that constructs a floor-plan for G using only 1- and 2-rectangle modules.

As mentioned above, the existence of separating triangles is the reason for using nonrectangular modules. The main problem in designing a floor-plan for G using low complexity modules is to "eliminate" all separating triangles of G in some way. The weighted separating triangle elimination problem has been shown to be NP-complete [20].

The floor-plan algorithm in [25] consists of two steps. The first step is to "assign" all separating triangles of G to the vertices of G so that each vertex is assigned at most twice. The second step constructs a floor-plan by using the separating triangle-vertex assignment computed in the first step. The algorithms in [25] for both steps use recursion and are fairly complex. The floor-plan algorithm presented here uses the same two steps as in [25]. However, our approaches are quite different from that in [25]. In the first step, we use the *canonical ordering* concept introduced in [6]. The

resulting algorithm is extremely simple. Our algorithm for the second step uses the *vertex expansion* operation to eliminate all separating triangles and then constructs a floor-plan of G by calling the rectangular dual algorithm in [7, 12, 13]. Although both our algorithm and the algorithm in [25] run in linear time, our algorithm is much simpler. In addition, the floor-plan constructed by our algorithm uses 1- and 2-rectangle modules and integer coordinates. In contrast, the floor-plans produced by the algorithm in [25] need 3-rectangle modules.

Instead of using the vertex expansion operation, there is another way to eliminate a separating triangle T from the input graph: Break an edge e of T by adding a new vertex at the middle of e. A heuristic algorithm based on this approach was given in [23]. Unfortunately, this approach does not work well. We will present a plane graph G and show that if the "break edge" approach is used to construct a floor-plan for G, we must use 3-rectangle modules. The construction of G requires sophisticated graph-theoretic arguments.

The present paper is organized as follows. Section 2 presents necessary definitions and background. In section 3, we present our simple floor-plan algorithm. In section 4, we describe a plane graph G whose floor-plan requires using 3-rectangle modules if the "break-edge" approach is taken.

2. Definitions and background. Most graph-theoretic definitions in this paper are standard [3]. Let $G = (V, E)$ be a planar graph with n vertices. In this paper, we always assume G is equipped with a fixed plane embedding. Namely G is a *plane* graph. The embedding of G divides the plane into a number of connected regions. Each region is called a *face*. The unbounded face of G is called its *exterior face*. Other faces are *interior faces*. The vertices and the edges on the boundary of the exterior face are called *exterior vertices* and *exterior edges*. Other vertices and edges are *interior vertices* and *interior edges*. $N(v)$ denotes the set of neighbors of a vertex v. Let $deg(v) = |N(v)|$.

In this paper, the terms *path* and *cycle* always mean *simple path* and *simple cycle* (i.e., the vertices of path and cycle are distinct). A *triangle* is a cycle consisting of three edges. A cycle C of G divides the plane into its interior and exterior regions. If x is a vertex of a cycle C, we say C *contains* x. If a vertex y is in the interior of C, we say C *encloses* y. If a triangle C encloses at least one vertex in its interior, C is called a *separating cycle*. (This definition differs from its standard meaning which requires that there exists at least one vertex in the interior and at least one vertex in the exterior of C.) An *internally triangulated* plane graph is a plane graph all of whose interior faces are triangles. A *triangulated* plane graph is a plane graph all of whose faces (including the exterior face) are triangles.

We assume the embedding information of G is given by the following data structure. For each $v \in V$, there is a doubly linked circular list containing all vertices of $N(v)$ in counterclockwise order. The two copies of an edge (u, v) are cross-linked to each other. This representation can be constructed as a by-product by using a planarity testing algorithm in linear time (e.g., [10]).

A *floor-plan* F is a rectangle partitioned into a set of disjoint rectilinear modules. If all modules of F are rectangles, F is called a *rectangular dual*. A floor-plan F represents a plane graph G as follows: Each module of F corresponds to a vertex of G. Two vertices of G are adjacent in G iff their corresponding modules in F share at least one line seqment as their common boundary. A floor-plan F of a plane graph G is shown in Figure 2. The vertices e, i, g form a separating triangle and the module for g is a nonrectangular module.

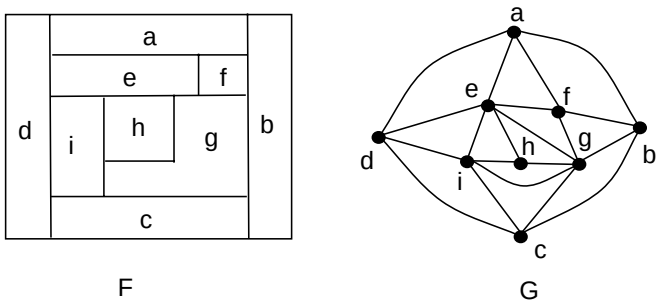


FIG. 2. *A floor-plan F of a plane graph G.*

Given a floor-plan F, the locations where at least two line segments meet are called the *points* of F. Since all modules of F are rectilinear polygons, at most four line segments can meet at the same point. We will assume the degree of any point of F is at most 3. This convention is used in most literature (see, for example, [25]). This convention considerably simplifies discussions and does not really limit the generality of the results. (The reason can be seen as follows. Each interior point of F corresponds to an internal face of G. Thus each internal face of G consists of either three or four vertices. If G has a face f consisting of four vertices u, v, w, x in counterclockwise order, we add a dummy edge (u, w) to triangulate the face f. In the floor-plan for the resulting graph, the modules for the vertices u and w share a common boundary. In the VLSI layout problem, we can simply ignore this common boundary. Alternatively, if four line segments meet at a point p of F, we can replace p by two degree-3 points connected by a line segment of zero length). Thus we assume all interior faces of G are triangles. When discussing rectangular duals, we assume that the exterior face of G has four vertices. (The conversion from the case of more than four exterior vertices to the case of four exterior vertices were discussed in [1, 7, 14]). When discussing floor-plans (where nonrectangular modules are allowed), we assume that G has either three or four exterior vertices. So under these conventions, we can restrict our discussion to internally triangulated plane graphs with either three or four exterior vertices. We will call such a graph a triangulated plane graph (TPG). Our floor-plan algorithm is based on the following theorem [7, 12, 13]:

THEOREM 2.1. *A TPG G has a rectangular dual iff it has four exterior vertices and has no separating triangles. Moreover, a rectangular dual of G with integer coordinates can be constructed in linear time.*

3. A simple floor-plan algorithm. Let $G = (V, E)$ be a TPG. We want to construct a floor-plan for G. The basic idea of our algorithm is as follows. First, we identify a subset $V' \subseteq V$ such that every separating triangle T of G contains at least one vertex $x \in V'$. (In this case, we say T is *assigned to* x). Then we convert G to a new TPG G_1 which has no separating triangles. G_1 is obtained from G by performing the *vertex expansion* operation on all vertices in V'. When this operation is performed on a vertex $x \in V'$, x is "split" into two vertices x_1 and x_2. The purpose of this operation is to "destroy" the separating triangles of G assigned to x. Next we run the rectangular dual algorithm in Theorem 2.1 on G_1 to obtain a rectangular dual F_1 of G_1. Finally, a floor-plan F of G is constructed from F_1 as follows: For each vertex $x \in V'$ that was expanded into two vertices x_1 and x_2 in G_1, the module of F representing x is the union of the two rectangles of F_1 corresponding to x_1 and x_2.

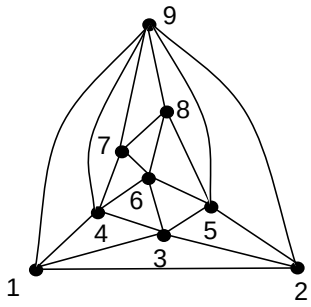


FIG. 3. *A canonical ordering of G.*

The rest of this section describes the details of our algorithm.

Let $\mathcal{T}(G)$ denote the set of all separating triangles of G. If a separating triangle $T_1 \in \mathcal{T}(G)$ is in the interior of another separating triangle $T_2 \in \mathcal{T}(G)$, T_1 is a *descendent* of T_2. If there is no $T_3 \in \mathcal{T}(G)$ such that T_1 is a descendent of T_3 and T_3 is a descendent of T_2, T_1 is a *child* of T_2. The child relation defines a forest structure on $\mathcal{T}(G)$.

Consider a separating triangle $T \in \mathcal{T}(G)$ with three vertices x, y, z. In any floor-plan of G, in order to satisfy the adjacency requirements of T, at least one of the three modules for x, y, z must be a nonrectangular module. We will *assign* T to one of x, y, z. If a vertex x is assigned a separating triangle, x will be represented by a nonrectangular module. Consider two separating triangles $T_1, T_2 \in \mathcal{T}(G)$ where T_1 is a descendent of T_2 and both T_1 and T_2 contain x. If we use a nonrectangular module for x, it is possible that the adjacency requirements for both T_1 and T_2 are satisfied. So we can assign both T_1 and T_2 to x. This observation was made in [25]. It leads to the following definition.

DEFINITION 3.1. *A* nesting sequence assigned to a vertex $x \in V$ *is a sequence* $T_1, T_2, \ldots, T_k$ *of separating triangles in* $\mathcal{T}(G)$ *such that each* T_i $(1 \le i \le k)$ *contains* x; *and* T_i *is a descendent of* T_{i+1} *for* $1 \le i \le k-1$.

In order to construct a floor-plan for G, we need to assign all separating triangles in $\mathcal{T}(G)$ to the vertices of G satisfying the following *valid assignment conditions*.

DEFINITION 3.2. *A* valid assignment *of G is a mapping of all separating triangles in* $\mathcal{T}(G)$ *to a set of nesting sequences such that:*

1. *Each* $T \in \mathcal{T}(G)$ *belongs to at least one nesting sequence.*
2. *Each vertex of G is assigned at most two nesting sequences.*

The first step of our floor-plan algorithm is to find a valid assignment of G. Our algorithm uses the following *canonical ordering* concept introduced in [6].

DEFINITION 3.3. *Let* $G = (V, E)$ *be a TPG with three exterior vertices* u, v, w. *A* canonical ordering *of G is an ordering of V by* $v_1, v_2, \ldots, v_n$ *such that* $v_1 = u$, $v_2 = v$, $v_n = w$ *and the following requirements hold for every* $4 \le k \le n$:

1. *The subgraph* G_{k-1} *of G induced by* $v_1, v_2, \ldots, v_{k-1}$ *is biconnected and the boundary of its exterior face is a cycle* C_{k-1} *containing the edge* (v_1, v_2).
2. v_k *is in the exterior face of* G_{k-1}, *and its neighbors in* G_{k-1} *form a subpath of the path* $C_{k-1} - \{(v_1, v_2)\}$ *consisting of at least two vertices. If* $k \le n-1$, v_k *has at least one neighbor in* $G - G_{k-1}$.

An example of a canonical ordering is shown in Figure 3.

The exterior face C_k of G_k is called the *contour* of G_k. The canonical ordering

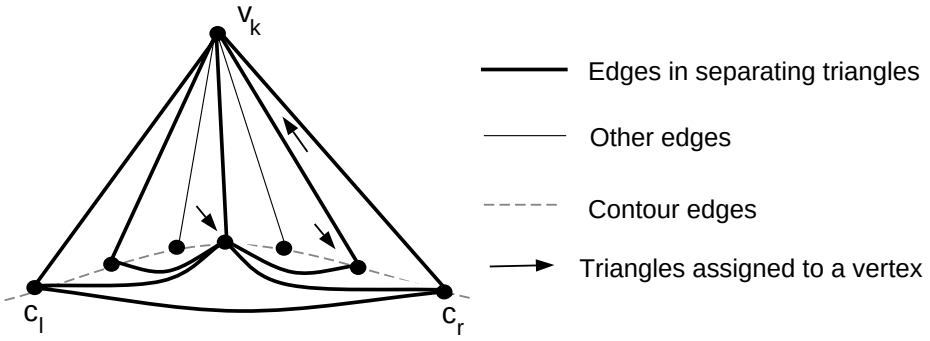


FIG. 4. *Assigning separating triangles to vertices.*

of G can be viewed as an ordering of adding the vertices one-by-one into the graph starting from the edge (v_1, v_2). Let $c_l, c_{l+1}, \ldots, c_r$ be the neighbors of v_k in G_{k-1} ordered from left to right. When v_k is added, it becomes a new contour vertex while the vertices $c_{l+1}, \ldots, c_{r-1}$ cease to be contour vertices. If v_k and two vertices c_i, c_j $(l \leq i < j \leq r)$ form a separating triangle T, (v_k, c_i) is called the *left edge* of T and (v_k, c_j) the *right edge* of T. The following algorithm computes a valid assignment of G.

ALGORITHM 1.
Input: A TPG G.
Output: A valid assignment of G.

1. Compute a canonical ordering $v_1, v_2, \ldots, v_n$ of G by calling the algorithm in [11]. (If G has four exterior vertices, a slight modification of the algorithm in [11] finds a canonical ordering of G. Let u, v, w, x be the four exterior vertices of G in counterclockwise order. If (v, x) is not an edge of G, u, v, x will be numbered as v_1, v_2, v_n. If (v, x) is an edge of G, u, v, w will be numbered as v_1, v_2, v_n).
2. For $k := 4$ to n Do:
   (a) Let $c_l, c_{l+1}, \ldots, c_r$ be the neighbors of v_k in G_{k-1} ordered from left to right. For each i $(l < i < r)$, the separating triangles of G (if any) that are introduced when v_k is added and have (v_k, c_i) as their right edge define a nesting sequence. Assign this sequence to c_i. (See Figure 4.)
   (b) The separating triangles of G (if any) that are introduced when v_k is added and have (v_k, c_r) as their right edge define a nesting sequence. Assign this sequence to v_k. (See Figure 4.)

End Algorithm 1.

LEMMA 3.1. *Algorithm* 1 *computes a valid assignment of* G *and runs in linear time.*

Proof. Consider an arbitrary separating triangle T of G. Let v_k be the highest numbered vertex among the three vertices of T. At the kth stage of the For loop, T belongs to a nesting sequence that is assigned to either a contour vertex c_i or to the vertex v_k. Since T is arbitrary, each separating triangle of G belongs to at least one nesting sequence.

Any vertex v_k of G can be assigned at most two nesting sequences: (i) when it first becomes a contour vertex (at step 2 (b)); and (ii) when it ceases to be a contour

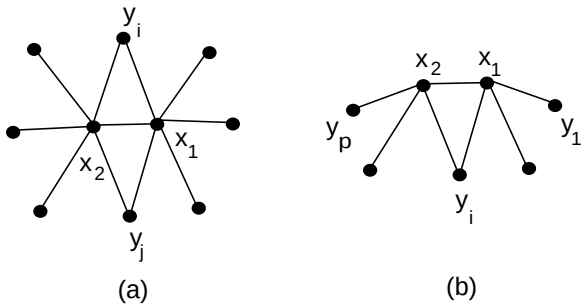


FIG. 5. *Vertex expansion operation.*

vertex (at step 2 (a)). Thus Algorithm 1 computes a valid assignment of G.

Next, we show Algorithm 1 can be implemented in linear time. A canonical ordering of G can be computed in $O(n)$ time [11]. By using the algorithm in [4], we can enumerate all triangles of G in linear time. For each triangle T of G identified, we can check if T is a separating triangle of G in $O(1)$ time by using the plane embedding data structure of G. Thus the set $\mathcal{T}(G)$ of all separating triangles of G can be identified in $O(n)$ time. Sort the separating triangles T in $\mathcal{T}(G)$ according to the highest numbered vertex in T. This can be done by using bucket sort in $O(n)$ time. Let $\mathcal{T}(v_k)$ be the separating triangles in $\mathcal{T}(G)$ with v_k as the highest numbered vertex.

The kth iteration of the For loop processes the separating triangles in $\mathcal{T}(v_k)$. Clearly, the number of separating triangles in $\mathcal{T}(v_k)$ is at most $O(deg(v_k))$. We sort the separating triangles T in $\mathcal{T}(v_k)$ according to the right vertex of T on the contour $c_l, \ldots, c_r$. This can be done in $O(deg(v_k))$ time by using bucket sort. For each c_i $(l < i < r)$, all separating triangles in $\mathcal{T}(v_k)$ with (c_i, v_k) as the right edge are assigned to c_i. All separating triangles with (c_r, v_k) as the right edge are assigned to v_k. Clearly, the kth iteration can be performed in $O(deg(v_k))$ time. So the total time to execute step 2 is $O(\sum_{k=4}^{n} deg(v_k)) = O(n)$. □

Remark 1. The vertices v_1 and v_2 are not assigned any sequence. If G has three exterior vertices, its exterior face is a separating triangle and the vertex v_n is assigned exactly one sequence. If G has four exterior vertices $u = v_1, v = v_2, w, x$, the vertices w and x are assigned at most one sequence each.

Next we describe the second step of our algorithm: Construct a floor-plan of G using a valid assignment of G. First we define the *vertex expansion* operation.

DEFINITION 3.4. *Let x be an interior vertex of G with neighbors $y_1, \ldots, y_p$ in clockwise order. The operation of* vertex expansion on x with respect to y_i, y_j $(1 \leq i < j \leq p)$ *is (see Figure* 5 (a)*):*

1. *Delete x. Create two new vertices x_1 and x_2. Add a new edge (x_1, x_2).*
2. *For $t = i+1, \ldots, j-1$, replace the edge (x, y_t) by (x_1, y_t). For $t = j+1, j+2, \ldots, p, 1, \ldots, i-1$, replace the edge (x, y_t) by (x_2, y_t).*
3. *Replace the two edges (x, y_i) and (x, y_j) by four new edges: (x_1, y_i), (x_2, y_i) and (x_1, y_j), (x_2, y_j).*

DEFINITION 3.5. *Let x be an exterior vertex of G with neighbors $y_1, \ldots, y_p$ in clockwise order where y_1 and y_p are exterior vertices. The operation of* vertex expansion on x with respect to y_i $(1 \leq i \leq p)$ *is as follows (see Figure* 5 (b)*).*

1. *Delete x. Create two new vertices x_1 and x_2. Add a new edge (x_1, x_2).*
2. *For $t = 1, \ldots, i-1$, replace the edge (x, y_t) by (x_1, y_t). For $t = i+1, \ldots, p$,*

replace the edge (x, y_t) *by* (x_2, y_t).

3. *Replace the edge* (x, y_i) *by two new edges:* (x_1, y_i), (x_2, y_i).

Note that after x is expanded, we always have $N(x) = N(x_1) \cup N(x_2)$. In a valid assignment of G, each vertex of G is assigned at most two nesting sequences. For each vertex x that is assigned at least one sequence, we perform the vertex expansion operation on x. There are three cases.

Case 1. x is an interior vertex and is assigned two sequences $\{T_1, T_2, \ldots, T_p\}$ and $\{S_1, S_2, \ldots, S_q\}$ (where T_1 is the innermost triangle in the first sequence and S_1 is the innermost triangle in the second sequence). Let y_i be a neighbor of x in the interior of T_1 (since T_1 is a separating triangle, y_i must exist) and y_j be a neighbor of x in the interior of S_1. Perform the vertex expansion operation on x with respect to y_i and y_j.

Case 2. x is an interior vertex and is assigned one sequence $\{T_1, T_2, \ldots, T_p\}$ (where T_1 is the innermost triangle in the sequence). Let y_i be a neighbor of x in the interior of T_1 and y_j be a neighbor of x in the *exterior* of T_1. (It is easy to see that y_j must exist). Perform the vertex expansion operation on x with respect to y_i and y_j.

Case 3. x is an exterior vertex. In this case, x is assigned one sequence $\{T_1, T_2, \ldots, T_p\}$ (where T_1 is the innermost triangle in the sequence). Let y_i be a neighbor of x in the interior of T_1. Perform the vertex expansion operation on x with respect to y_i.

After performing the vertex expansion operation on all vertices that are assigned at least one nesting sequence, the resulting graph G_1 is a TPG with no separating triangles. (This is because every separating triangle of G belongs to at least one nesting sequence assigned to a vertex v, and the expansion of v destroys such a separating triangle. On the other hand, no new separating triangles can be created by vertex expansion operation). Below we describe how to obtain a floor-plan for G.

First, assume G has three exterior vertices. By Remark 1, exactly one exterior vertex is assigned a sequence and is expanded. Thus G_1 has four exterior vertices. We use the algorithm in Theorem 2.1 to find a rectangular dual F_1 for G_1. A floor-plan F of G can be obtained from F_1 as follows. Consider any vertex x of G. If the vertex expansion operation is not performed on x, we take the rectangle in F_1 for x to be the module in F representing x. If the vertex expansion operation is performed on x, there are two vertices x_1 and x_2 in G_1 corresponding to x. The module in F representing x is the union of the two rectangles in F_1 for x_1 and x_2. Clearly, F is a floor-plan of G consisting of only 1- and 2-rectangle modules.

Next assume G has four exterior vertices. By Remark 1, either 0, or 1, or 2 exterior vertices of G are assigned a sequence and expanded. Thus G_1 has either 4, or 5, or 6 exterior vertices. In the first case, we can use the same construction described in the last paragraph. In the other two cases, in order to use Theorem 2.1, we have to modify the graph G_1. We consider the case where G_1 has 6 exterior vertices. (The case of 5 exterior vertices is similar). Let u, v, w, x be the four exterior vertices of G in counterclockwise order. Suppose that w is expanded to w_1 and w_2; and x is expanded to x_1 and x_2. The exterior face of G_1 is u, v, w_1, w_2, x_1, x_2 in counterclockwise order (Figure 6 (a).) It is easily seen that G_1 cannot have both edges (u, w_2) and (v, x_1). We modify G_1 to a new TPG G_2 as follows: Add two new vertices y, z and a new edge (y, z) in the exterior face of G_1. Connect y and z to the exterior vertices of G_1 according to the following conditions:

Case A. (u, w_2) is not an edge of G_1. Add edges $(z, u), (z, x_2), (z, x_1), (z, w_2)$ and the edges $(y, w_2), (y, w_1), (y, v)$. (See Figure 6 (a).) Let G_2 be the resulting graph. It is easy to check that G_2 has no separating triangles. So we can construct a rectangular dual F_2 of G_2 by using the algorithm in Theorem 2.1. By removing the two rectangles of F_2 corresponding to y and z (and modifying the rectangles for u and v if necessary),

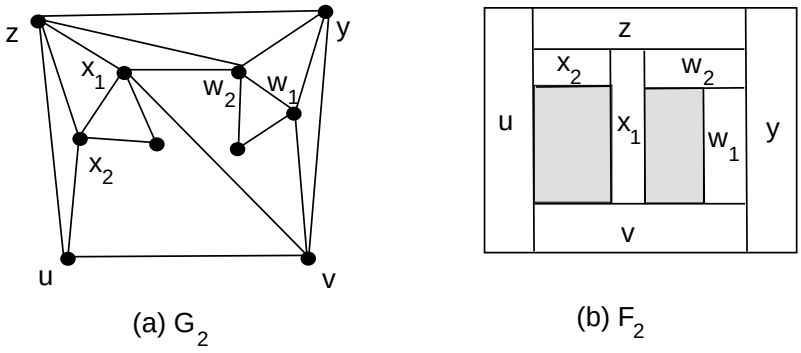


FIG. 6. *Convert G_1 to a TPG with four exterior vertices.*

we get a rectangular dual F_1 for G_1. (See Figure 6 (b).) By using the method described above, F_1 can be converted to a floor-plan F of G.

Case B. (v, x_1) is not an edge of G_1. Add edges: $(z, u), (z, x_2), (z, x_1), (y, x_1)$, $(y, w_2), (y, w_1), (y, v)$. Then the construction is the same as in Case A.

THEOREM 3.2. *Given an arbitrary TPG G, a floor-plan of G using* 1- *and* 2-*rectangle modules and integer coordinates exists and can be computed in linear time.*

Proof. The existence of such a floor-plan is proved in the above discussion. We only need to show it can be constructed in linear time. A valid assignment of G can be computed in linear time (Lemma 3.1). The vertex expansion operation on x can be performed in $O(deg(x))$ time by using the planar embedding data structure. Thus the total time for this step is $O(\sum_{x \in V} deg(x)) = O(n)$. After G_1 is constructed, its rectangular dual F_1 can be computed in $O(n)$ time by using the algorithms in [7, 12, 13]. The conversion from F_1 to the floor-plan F of G clearly takes linear time. □

4. A TPG requiring 3-rectangle models using break-edge method. A heuristic algorithm was proposed in [23] to construct a floor-plan for a triangulated plane graph G using low complexity modules. Instead of using vertex expansion, this algorithm uses *breaking edge* operation to eliminate the separating triangles. In this section, we show that this heuristic algorithm does not work well. We will describe a plane graph G and show that if the algorithm in [23] is applied to G, a 3-rectangle module must be used.

First we describe the "break edge" operation used in [23]. Consider a separating triangle T of G with three vertices x, y, z. In order to "eliminate" T, an edge of T, say (x, y) is selected. Let T' and T'' be the two interior triangular faces of G having (x, y) on their boundary. Let u and v be the third vertex of T' and T'', respectively (Figure 7 (a)). Now add a new vertex w at the middle of (x, y) and connect w to u and v. (If (x, y) is an exterior edge, there is only one interior face T' containing (x, y). In this case, we only connect w to u.) After this "break edge" operation, the resulting graph is still a TPG and T is no longer a separating triangle. We say the edge (x, y) *covers* T. The edge (x, y) is *assigned* to either x or y.

In order to find a floor-plan F of G, the algorithm in [23] performs this operation on all separating triangles of G. More precisely, for each separating triangle T, an edge $e = (x, y)$ of T is selected to cover T and e is assigned to either x or y. Let E_1 be the set of the selected edges. Then the "break edge" operation is performed on

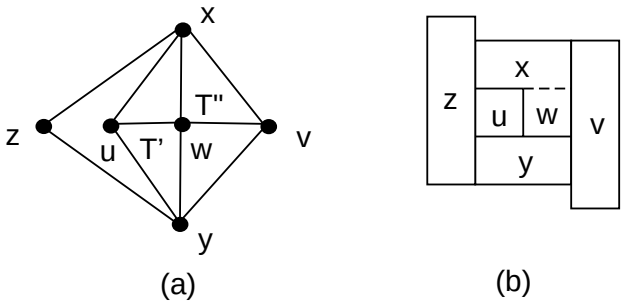


FIG. 7. *Break an edge.*

every edge in E_1. The resulting graph G_1 is a TPG with no separating triangles. A rectangular dual F_1 of G_1 is constructed. F_1 can be converted to a floor-plan F of G as follows: For each vertex x of G, the module in F representing x is the union of the rectangles in F_1 for x and for the added vertices on the edges that are assigned to x. (See Figure 7 (b). In this figure, the edge (x, y) is assigned to x. So the module for x is the union of the two rectangles labeled by x and w.) F is clearly a floor-plan of G. Note that F uses only 1- and 2-rectangle modules iff each vertex x of G is assigned at most one edge.

DEFINITION 4.1. *A* valid cover-assignment *of $G = (V, E)$ is a subset $E_1 \subseteq E$ and an assignment of each edge $e \in E_1$ to one of its two end vertices such that the following hold:*

1. *Each separating triangle T of G has at least one edge in E_1.*
2. *Each vertex of G is assigned at most one edge.*

Thus if G has a floor-plan consisting of only 1- and 2-rectangle modules by using this "break edge" approach, then G must have a valid cover-assignment. In this section, we describe a TPG $\bar{G}$ and show that $\bar{G}$ has no valid cover-assignment. Hence if the "break edge" approach is used to eliminate the separating triangles of $\bar{G}$, there must be a vertex v with at least two edges assigned to v. Therefore, in the floor-plan for $\bar{G}$ constructed by using this approach, the module for v is the union of at least three rectangles. In other words, we are forced to use 3-rectangle modules.

Consider a triangulated plane graph G with n vertices. Let $\bar{G}$ be the TPG obtained from G by adding a new vertex v_f in each internal face f of G and connecting v_f to the three vertices of f. We would like to find a floor-plan for $\bar{G}$. Note that all faces of G are separating triangles of $\bar{G}$. Let G^* be the dual graph of G. To avoid confusion, the vertices of G^* will be referred to as *nodes*. Each edge e^* of G^* corresponds to an edge e in G. e^* will be called the dual edge of e. Clearly, G^* is a 3-regular and 3-connected plane graph.

Our proof plan is as follows: First, we assume that $\bar{G}$ has a valid cover-assignment and derive the conditions that G and G^* must satisfy. Then we present a graph G and show that these conditions do not hold for G and G^*.

Define a *pseudotree* to be a connected graph with at most one cycle. In other words, a pseudotree is either a tree, or a tree T plus an additional edge connecting two vertices of T.

Let E_1 be a valid cover-assignment of $\bar{G}$. Since only the edges in G can be used to cover the separating triangles of $\bar{G}$, E_1 is actually a subset of the edges in G. Let $G[E_1]$ denote the subgraph of G spanned by E_1.

LEMMA 4.1. *Let E_1 be a valid cover-assignment of $\bar{G}$. Let $G[E_1]$ denote the subgraph of G spanned by E_1. Then any connected component of $G[E_1]$ is a pseudotree.*

Proof. Since each edge in E_1 is assigned to a unique vertex of G, we have $|E_1| \leq n$. Consider a connected component C of $G[E_1]$ with k vertices. Being connected, C has at least $k-1$ edges. On the other hand, since every edge of C is assigned to a unique vertex of C, C contains at most k edges. So C is a pseudotree. □

Consider the dual graph G^* of G. Each node of G^* corresponds to a face of G. Since G is a triangulated plane graph, G^* has $2n-4$ nodes. Let E_1^* be the set of dual edges of E_1. A face f of G is covered by an edge $e \in E_1$ iff the dual edge e^* of e is incident to the node in G^* corresponding to f. Since all of the $2n-4$ faces of G must be covered and each edge of E_1 can cover only two faces, we have $|E_1^*| = |E_1| \geq n-2$. Hence $|E_1^*| = n-2$, or $n-1$, or n. Consider the subgraph $G^* - E_1^*$ (which is obtained from G^* by removing the edges in E_1^*). Since G^* is 3-regular, we have the following three cases.

Case 1. $|E_1^*| = n-2$. Then each node in G^* is incident to exactly one edge in E_1^*. (Namely, E_1^* is a perfect matching of G^*). Thus, each node of G^* has degree 2 in $G^* - E_1^*$. Hence $G^* - E_1^*$ is a collection of disjoint cycles.

Case 2. $|E_1^*| = n-1$. Either one node in G^* is incident to three edges in E_1^* and all other nodes are each incident to one edge in E_1^*; or two nodes in G^* are each incident to two edges in E_1^* and all other nodes are each incident to one edge in E_1^*. In either case, $G^* - E_1^*$ is the union of a path and a collection of cycles. (A single node is considered as a path of length 0).

Case 3. $|E_1^*| = n$. Similar to Case 2, it can be shown that $G^* - E_1^*$ is the union of two paths and a collection of cycles.

This motivates the following definition.

DEFINITION 4.2. *A* Hamiltonian-like decomposition (HLD) *of a 3-regular plane graph G^* is a partition of the node set of G^* into subsets $P_1, P_2, \ldots, P_s$ such that each P_i $(1 \leq i \leq s)$ forms either a cycle or a path of G^*, and at most two P_i's are paths.*

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_s\}$ be a HLD of G^*. Consider a cycle $P_k \in \mathcal{P}$. If $P_i \in \mathcal{P}$ $(i \neq k)$ is in the interior of P_k, we say P_i is a *descendent* of P_k. If there is no $P_j \in \mathcal{P}$ such that P_i is a descendent of P_j and P_j is a descendent of P_k, then P_i is a *child* of P_k. The child relation defines a forest structure on $\mathcal{P}$. Let $K(G^*, \mathcal{P})$ be the following graph: Its vertex set is $\mathcal{P}$. $P_i, P_j \in \mathcal{P}$ are adjacent in $K(G^*, \mathcal{P})$ iff P_i is a child of P_j.

DEFINITION 4.3. *A HLD $\mathcal{P}$ of G^* is called a* linear HLD *if the forest $K(G^*, \mathcal{P})$ has at most two trees and each of them is a linear path (i.e., each vertex has at most one child).*

THEOREM 4.2. *If $\bar{G}$ has a valid cover-assignment E_1, then $G^* - E_1^*$ is a linear HLD of G^*.*

Proof. In the above discussion, we have shown that if $\bar{G}$ has a valid cover-assignment E_1, then $\mathcal{P} = G^* - E_1^* = \{P_1, P_2, \ldots, P_q\}$ is a HLD of G^*. We will show $\mathcal{P}$ is linear. Suppose not. Then there exist $P_i, P_j, P_k \in \mathcal{P}$ such that one of the following two situations occurs:

(1) P_k is a cycle; P_i and P_j are children of P_k;

(2) P_i, P_j, P_k are not in the interior of any cycle in $\mathcal{P}$.

The theorem follows if we can prove both (1) and (2) are impossible. We first prove the situation (1) is impossible. Let $P_{l_1} = P_i, P_{l_2} = P_j, P_{l_3}, \ldots, P_{l_r}$ be the members in $\mathcal{P}$ that are children of P_k. Let $E_2^* \subset E_1^*$ be the subset of the edges in E_1^* that are in the interior of P_k but in the exterior of $P_{l_1}, P_{l_2}, \ldots, P_{l_r}$. Let E_2 be the subset of the

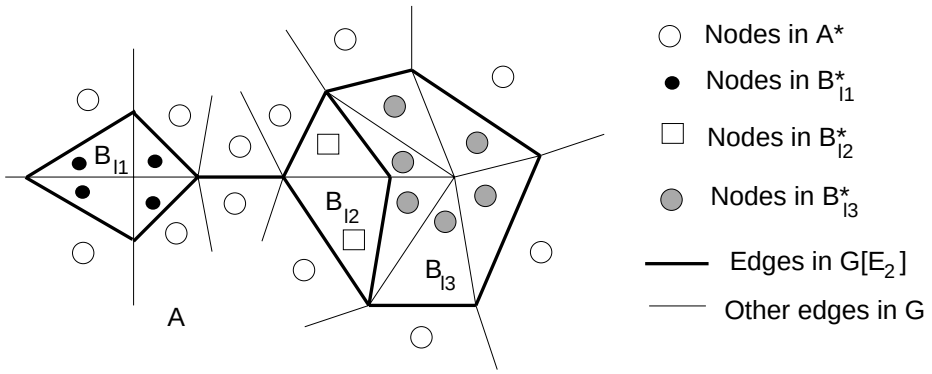


FIG. 8. *The proof of the claim.*

edges in G corresponding to the edges in E_2^*.

Claim. The subgraph of $G[E_2]$ of G is connected and is not a pseudotree.

Note: The claim implies that $G[E_2]$ is contained in a connected component of $G[E_1]$ and is not a pseudotree. Hence E_1 cannot be a valid cover-assignment of $\bar{G}$ by Lemma 4.1.

Let A^* be the set of the nodes of G^* that are on P_k or in the exterior of P_k. For each $1 \leq s \leq r$, let $B_{l_s}^*$ be the set of the nodes of G^* that are on P_{l_s} or in the interior of P_{l_s} (if P_{l_s} is a cycle).

Let A be the region on the plane that is the union of the faces of G corresponding to the nodes in A^*. Let B_{l_s} $(1 \leq s \leq r)$ be the region on the plane that is the union of the faces of G corresponding to the nodes in $B_{l_s}^*$.

It can be seen that $G[E_2]$ is exactly the common boundary of the regions A,$B_{l_1}, \ldots,$ B_{l_r}. (See Figure 8 where $r = 3$). It is easy to show that the common boundary of these regions are connected. The boundary of B_{l_1} is a cycle (not necessarily simple). The boundary of B_{l_2} is also a cycle. Thus, $G[E_2]$ is connected and has at least two cycles. Hence it is not a pseudotree. This proves the claim. So the situation (1) cannot occur.

To show the situation (2) cannot occur, let P_k be the member of $\mathcal{P}$ that contains the node of G^* corresponding to the exterior face of G and repeat the above argument. □

In the rest of this section, we present a 3-regular and 3-connected plane graph G^* that does not have a linear HLD. Then the corresponding graph $\bar{G}$ has no valid cover-assignment by Theorem 4.2. The construction of G^* is fairly complex. This is to be expected considering the following historical perspective (see [3], pp. 160–162). While attempting to prove the 4-color conjecture, Tait "proved" the following proposition in 1880 [21].

PROPOSITION. *Every* 3*-regular and* 3*-connected planar graph has a Hamiltonian cycle.*

The 4-color theorem follows easily from this proposition (see [3, pp. 158–159]). Over sixty years later, however, Tutte showed in [24] that this proposition is false by giving a counter-example shown in Figure 9 (a) (ignore the vertices a, b, c). This graph is commonly referred to as the Tutte's graph. Since the conditions of having a linear HLD is weaker than that of having a Hamiltonian cycle, it is harder to find a 3-regular and 3-connected planar graph without a linear HLD.

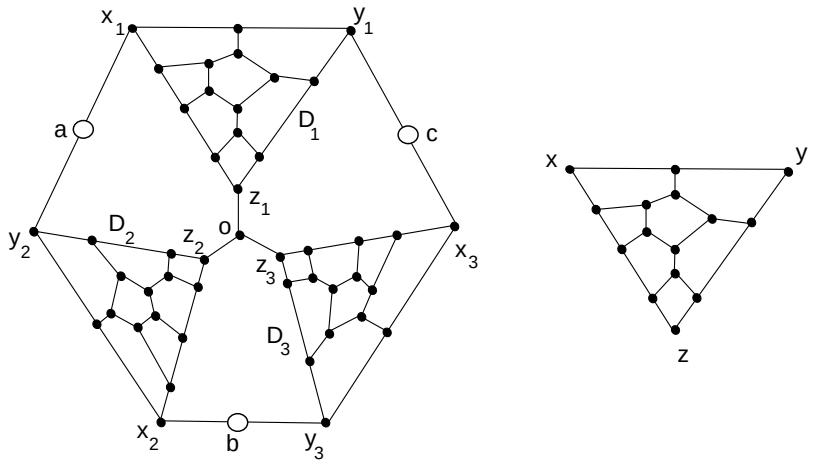


FIG. 9. *Tutte's graph.*

We use the Tutte's graph as a building block in constructing our graph G^*. Let S denote the graph shown in Figure 9 (a). It is obtained from the Tutte's graph by adding three new nodes a, b and c at the middle of three edges. In the following, HP stands for Hamiltonian path.

LEMMA 4.3. *In the graph S, there is no HP between a and b; no HP between b and c, no HP between c and a.*

Proof. Tutte's graph T consists of three copies of the graph D shown in Figure 9 (b). The following fact was proved in [24].

Fact 1. In the graph D, there is no HP between x and y. (However, D does have a HP between y and z and a HP between x and z.)

Toward a contradiction, suppose that S has a HP Q starting at a and ending at c.

Case 1. The second node of Q is x_1 and the second last node of Q is y_1. Then $Q - \{(a, x_1), (y_1, c)\}$ is a HP in the graph $S - \{(y_2, a), (a, x_1), (y_1, c), (c, x_3)\}$ between x_1 and y_1. This is impossible.

Case 2. The second node of Q is x_1 and the second last node of Q is x_3. Then $Q' = Q - \{(a, x_1), (x_3, c)\}$ is a HP in the graph $S - \{(y_2, a), (a, x_1), (y_1, c), (c, x_3)\}$ between x_1 and x_3. It is easy to see that Q' must have the following form:

A HP of D_1 from x_1 to z_1; the edges (z_1, o) and (o, z_2); a HP of D_2 from z_2 to x_2, the edges (x_2, b) and (b, y_3); a HP of D_3 from y_3 to x_3.

However, the last HP of D_3 does not exist by Fact 1. So Case 2 is impossible.

Case 3. The second node of Q is y_2 and the second last node of Q is y_1. This case is symmetric to Case 2. Similar to Case 2, this case is impossible.

Case 4. The second node of Q is y_2 and the second last node of Q is x_3. Then $Q - \{(a, y_2), (x_3, c)\}$ is a HP in the graph $S - \{(y_2, a), (a, x_1), (y_1, c), (c, x_3)\}$ between y_2 and x_3. Clearly, this is impossible.

Since all four cases are impossible, the assumed HP between a and c does not exist. Similarly, we can show S has no HP between a and b and no HP between c and b. □

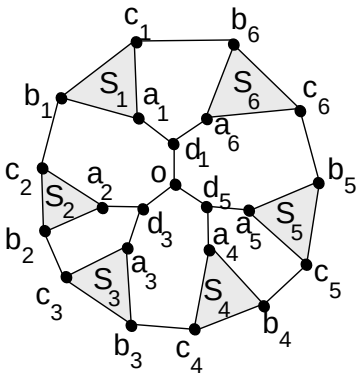


FIG. 10. *The graph G^*.*

Let G^* be the graph shown in Figure 10. G^* consists of six copies of S. Clearly, G^* is a 3-regular and 3-connected plane graph. We will show that G^* does not have a linear HLD. First, we prove a technical lemma.

LEMMA 4.4. *The node set of G^* cannot be partitioned into two paths.*

Proof. Towards a contradiction, suppose that the node set of G^* can be partitioned into two paths P_1 and P_2. At least one of S_i $(1 \leq i \leq 6)$, say S_1, does not contain any end nodes of P_1 and P_2. Since S_1 is connected to other parts of G^* by only three edges, all nodes of S_1 must be entirely contained in either P_1 or P_2. This can happen only if there is a HP in S_1 between a_1 and b_1, (or between b_1 and c_1, or between c_1 and a_1). This is impossible by Lemma 4.3. □

THEOREM 4.5. *G^* does not have a linear HLD.*

Proof. Consider any HLD $\mathcal{P} = \{P_1, P_2, \ldots, P_s\}$ of G^*. We show $\mathcal{P}$ cannot be a linear HLD. If no $P_i \in \mathcal{P}$ is a cycle, then we must have $s \leq 2$. This is impossible by Lemma 4.4. So at least one member of $\mathcal{P}$, say P_1, is a cycle. We will show that one of the following three cases hold for the graph $G^* - P_1$:

(a) $G^* - P_1$ has at least 3 connected components; or

(b) $G^* - P_1$ has two connected components and one of them cannot be enclosed in the interior of a cycle, nor can it have an HP; or

(c) $G^* - P_1$ has one connected component C and C cannot be enclosed in the interiors of two disjoint cycles; nor can C be partitioned into two paths; nor can some nodes of C be enclosed in a cycle and other nodes of C form a path.

If any of these three cases occurs, it is easily seen that $\mathcal{P}$ cannot be a linear HLD and we are done.

Case 1. The node o is on the cycle P_1. Without loss of generality, we assume the edges (o, d_1) and (o, d_3) are in P_1.

Case 1.1. The edges (d_1, a_6) and (d_3, a_3) are in P_1. Clearly, P_1 must contain either (i) the edge (b_4, c_5); (ii) the edge (b_1, c_2); or (iii) the node d_5.

(i) P_1 contains (b_4, c_5). Then P_1 passes through S_i $(i = 3, 4, 5, 6)$. (See Figure 11 (a). The solid lines indicate P_1.) By Lemma 4.3, P_1 cannot contain all nodes of S_i (for each $i = 3, 4, 5, 6$). So there must exist at least one *left-over* node u_i in S_i not belonging to P_1. Depending on whether u_i's $(i = 3, 4, 5, 6)$ are in the interior or the exterior of P_1, there are several subcases. If $G^* - P_1$ has at least three connected components, then by the remark at the beginning of the proof, $\mathcal{P}$ cannot be a linear HLD and we are done. The only case where $G^* - P_1$ has only two connected components is: u_4 and

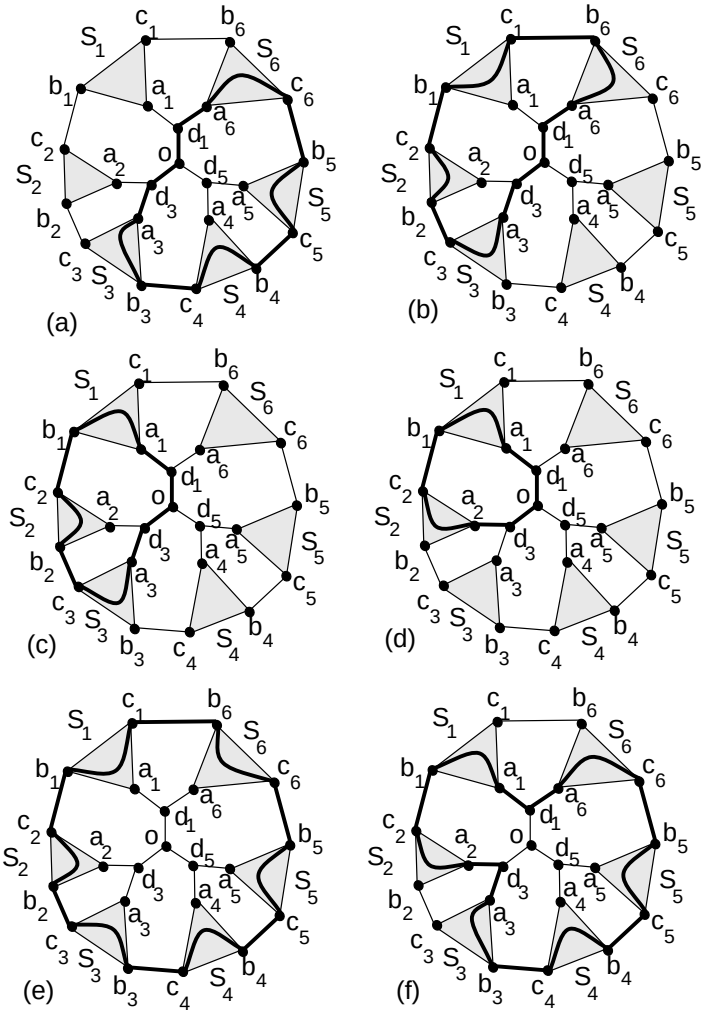


FIG. 11. *The proof of Theorem* 4.5.

u_5 are in the interior of P_1 and form a connected component C_1 (of $G^* - P_1$) through the node d_5; u_3 and u_6 are in the exterior of P_1 and form a connected component C_2 (of $G^* - P_1$) with the nodes in S_2 and S_1. However, C_2 does not have a HP (by Lemma 4.3); nor can its nodes be enclosed in a simple cycle (since the exterior face of C_2 is not a cycle). Thus, $\mathcal{P}$ cannot be linear.

(ii) P_1 contains the edge (b_1, c_2). Then P_1 passes through S_i $(i = 1, 2, 3, 6)$. (Figure 11 (b).) P_1 cannot contain all nodes in S_i. Thus, there must exist at least one left-over node u_i in S_i $(i = 1, 2, 3, 6)$ not belonging to P_1. Note that P_1 passes through the edges $(b_6, c_1), (b_1, c_2), (b_2, c_3)$ and the nodes o, d_1, d_3. Thus, $G^* - P_1$ has at least three connected components: one contains u_1, one contains u_2, and another contains the nodes in S_4 and S_5. Thus, $\mathcal{P}$ cannot be linear.

(iii) P_1 contains the node d_5. Then P_1 passes through the edges (b_3, c_4) and (b_5, c_6), but not the edge (b_4, c_5). Similar to the subcase (i), we can show $\mathcal{P}$ is not

linear.

Case 1.2. P_1 contains the edges (d_1, a_1) and (d_3, a_3). Then P_1 passes through either (i) the edge (b_4, c_5), (ii) the edge (b_1, c_2), or (iii) the node d_5.

(i) P_1 passes through (b_4, c_5). Then P_1 passes through S_i $(i = 1, 3, 4, 5, 6)$. Let u_i be the left-over node in S_i not belonging to P_1. Then $G^* - P_1$ has at least three connected components: one contains u_4 (and possibly u_5), one contains u_6, and another contains S_2. Thus, $\mathcal{P}$ cannot be linear.

(ii) P_1 passes through (b_1, c_2). Then P_1 passes through S_i $(i = 1, 2, 3)$. (See Figure 11 (c).) Let u_i $(i = 1, 2, 3)$ be the left-over node in S_i not belonging to P_1. Depending on whether u_1, u_2, u_3 are in the interior or the exterior of P_1, there are several subcases. If $G^* - P_1$ has at least three connected components, then $\mathcal{P}$ cannot be linear. The only case where $G^* - P_1$ has only two connected components C_1 and C_2 is that: C_1 contains u_2; and C_2 contains u_1, u_3 and the nodes in S_j $(j = 4, 5, 6)$. C_2 does not have a HP (by Lemma 4.3), nor can its nodes be enclosed in the interior of a cycle (since the exterior face of C_2 is not a cycle). Thus, $\mathcal{P}$ cannot be linear.

(iii) P_1 contains the node d_5. Similar to the subcase (i), we can show $\mathcal{P}$ is not linear.

Case 1.3. P_1 contains the edges (d_1, a_6) and (d_3, a_2). This case is symmetric to Case 1.2.

Case 1.4. P_1 contains the edges (d_1, a_1) and (d_3, a_2). Then P_1 must pass through either (i) the edge (b_4, c_5); or (ii) the edge (b_1, c_2); or the node d_5.

(i) P_1 passes through (b_4, c_5). Then P_1 passes through S_i $(i = 1, 2, 3, 4, 5, 6)$. Let u_i be the left-over node in S_i not belonging to P_1. Then $G^* - P_1$ contains at least four connected components: one contains u_3, one contains u_4 (and possibly u_5), one contains u_6, and another contains u_1 (and possibly u_2). Thus, $\mathcal{P}$ cannot be linear.

(ii) P_1 passes through (b_1, c_2). Let u_i $(i = 1, 2)$ be the left-over node in S_i not belonging to P_1. (See Figure 11 (d).) Depending on whether u_1 and u_2 are in the interior or the exterior of P_1, there are several subcases. One possible (the worst) subcase is that: both u_1 and u_2 are in the exterior of P_1 and are connected through S_i $(i = 3, 4, 5, 6)$. Thus, $G^* - P_1$ has only one connected component C containing u_1, u_2 and all nodes in S_i $(i = 3, 4, 5, 6)$. Note that C has at least six cut nodes: $c_3, b_3, c_4, b_5, c_6, b_6$. It is easily seen that C satisfies the condition (c) at the beginning of the proof. Thus, $\mathcal{P}$ cannot be linear.

(iii) P_1 contains the node d_5. Similar to the subcase (i), we can show $\mathcal{P}$ is not linear.

Case 2. P_1 does not contain the node o.

Case 2.1. P_1 does not contain the nodes d_1, d_3, d_5. Then P_1 must pass through the edges (b_i, c_{i+1}) (for $1 \leq i \leq 5$) and (b_6, c_1). (See Figure 11 (e).) Let u_i $(1 \leq i \leq 6)$ be the left-over node in S_i not belonging to P_1. Depending on whether u_i's are in the interior or the exterior of P_1, there are a number of subcases. One possible (the worst) case is that: All u_i are in the interior of P_1 and connected through the nodes d_1, d_3, d_5, o. Thus, $G^* - P_1$ has only one connected component C containing $o, d_1, d_3, d_5, u_1, \ldots, u_6$. Note that C contains a tree with at least six leaves. It is easily seen that C satisfies the condition (c) at the beginning of the proof. Thus, $\mathcal{P}$ cannot be linear.

Case 2.2. P_1 contains exactly one node from d_1, d_3, d_5, say d_1. Then P_1 passes through either (i) the edge (b_6, c_1), or (ii) the edge (b_3, c_4).

(i) P_1 passes through (b_6, c_1). Then P_1 passes through S_1 and S_6. Let u_i $(i = 1, 6)$ be the left-over node in S_i not belonging P_1. Depending on whether u_1 and u_6 are in

the interior or the exterior of P_1, there are a number of subcases. One possible (the worst) case is that: Both u_1 and u_6 are in the exterior of P_1 and are connected through the nodes in S_i $(i = 2, 3, 4, 6)$. Thus, $G^* - P_1$ has only one connected component C. It is easily seen that C satisfies the condition (c) at the beginning of the proof. Thus $\mathcal{P}$ cannot be linear.

(ii) P_1 passes through the edge (b_3, c_4). Then P_1 passes through S_i $(i = 1, 2, 3, 4, 5, 6)$. Let u_i $(i = 1, \ldots, 6)$ be the left-over node in S_i not belonging to P_1. Depending on whether u_i's are in the interior or the exterior of P_1, there are a number of subcases. If $G^* - P_1$ has at least three connected components, then $\mathcal{P}$ cannot be linear. The only case where $G^* - P_1$ has only two connected components is where u_1 and u_6 are in the exterior of P_1 and they form a connected component C_1 through the edge (b_6, c_1) and the nodes u_2, u_3, u_4, u_5 are in the interior of P_1 and they form a connected component C_2 through the nodes d_3, d_5, o. It is easily seen that C_2 does not have a HP, nor can its nodes be enclosed in a cycle. Thus, $\mathcal{P}$ cannot be linear.

Case 2.3. P_1 contains exactly two nodes from d_1, d_3, d_5, say P_1 contains d_1 and d_3. Since P_1 does not contain the node o, P_1 must be as shown in Figure 11 (f). Let u_i $(i = 1, 2, \ldots, 6)$ be the left-over node in S_i not belonging to P_1. In the best case, $G^* - P_1$ contains at least three connected components: one contains u_1 and u_6 connected through the edge (b_6, c_1); one contains u_2 and u_3 connected through the edge (b_2, c_3); and one contains u_4 and u_5 connected through the node d_5. Thus, $\mathcal{P}$ cannot be linear.

Case 2.4. P_1 contains the nodes d_1, d_3, d_5. Since P_1 does not contain the node o, it must pass through the edges $(b_1, c_2), (b_3, c_4), (b_5, c_6)$, but not the edges $(b_2, c_3), (b_4, c_5)$, (b_6, c_1). Let u_i $(1 \leq i \leq 6)$ be the left-over node in S_i not belonging to P_1. In the best case, $G^* - P_1$ has at least 4 connected components: one contains o alone; one contains u_1 and u_6; one contains u_2 and u_3; one contains u_4 and u_5. Thus, $\mathcal{P}$ cannot be linear.

Since $\mathcal{P}$ cannot be a linear HLD in all cases, the theorem is proved. □

Acknowledgments. The author thanks the anonymous referees for their helpful comments.

REFERENCES

[1] J. Bhasker and S. Sahni, *A linear algorithm to check for the existence of a rectangular dual of a planar triangulated graph*, Networks, 17 (1987), pp. 307–317.

[2] J. Bhasker and S. Sahni, *A linear algorithm to find a rectangular dual of a planar triangulated graph*, Algorithmica, 3 (1988), pp. 247–278.

[3] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North Holland, Amsterdam, 1979.

[4] N. Chiba and N. Takao, *Arboricity and subgraph listing algorithms*, SIAM J. Comput., 14 (1985), pp. 210–223.

[5] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Algorithms for drawing graphs: an annotated bibliography*, Comput. Geom., 4 (1994), pp. 235–282.

[6] H. de Fraysseix, J. Pach, and R. Pollack, *How to draw a planar graph on a grid*, Combinatorica, 10 (1990), pp. 41–51.

[7] X. He, *On finding the rectangular duals of planar triangular graphs*, SIAM J. Comput., 22 (1993), pp. 1218–1226.

[8] X. He, *An efficient parallel algorithm for finding rectangular duals of planar triangular graphs*, Algorithmica, 13 (1995), pp. 553–572.

[9] W. R. Heller, G. Sorkin, and K. Mailing, *The planar package planner for system designers*, in Proceedings of the 19th Annual IEEE Design Automation Conference, New York, 1982, pp. 253–260.

[10] J. Hopcroft and R. E. Tarjan, *Efficient planarity testing*, J. ACM, 21 (1974), pp. 549–568.

[11] G. Kant, *Drawing planar graphs using the lmc-ordering*, in Proceedings of the 33rd Annual IEEE Symposium on the Foundations of Computer Science, Pittsburgh, 1992, pp. 101–110.
[12] G. Kant and X. He, *Two algorithms for finding rectangular duals of planar graphs*, in Proceedings of the 19th Workshop on Graph-Theoretic Concepts in Computer Science, 1993. LNCS 790, pp. 396–410.
[13] G. Kant and X. He, *Regular edge labeling of* 4*-connected plane graph and its applications in graph drawing algorithms*, Theoret. Comput. Sci., 172 (1997), pp. 175–193.
[14] K. Koźmiński and E. Kinnen, *Rectangular dual of planar graphs*, Networks, 15 (1985), pp. 145–157.
[15] K. Koźmiński and E. Kinnen, *Rectangular dualization and rectangular dissection*, IEEE Trans. Circuits and Systems, 35 (1988), pp. 1401–1416.
[16] Y.-T. Lai and S. M. Leinwand, *A theory of rectangular dual graphs*, Algorithmica, 5 (1990), pp. 467–483.
[17] K. Mailing, S. H. Mueller, and W. R. Heller, *On finding most optimal rectangular package plans*, in Proceedings of the 19th Annual IEEE Design Automation Conference, New York, 1982, pp. 263–270.
[18] M. S. Rahman, S. Nakano, and T. Nishizeki, *Rectangular grid drawings of plane graphs*, Comput. Geom., 10 (1998), pp. 203–220.
[19] Y. Sun and M. Sarrafzadeh, *Floorplanning by graph dualization: L-shaped models*, in Proceedings of the IEEE International Symposium on Circuits and Systems, 1990, pp. 2845–2848.
[20] Y. Sun and K-H. Yeap, *Edge covering of complex triangles in rectangular dual floorplanning*, J. Circuits Systems Comput., 3 (1993), pp. 721–731.
[21] P. G. Tait, *Remarks on coloring of maps*, in Proc. Roy. Soc. Edinburgh Sect. A, 10 (1880), p. 729.
[22] C. Thomassen, *Plane representations of graphs*, in Progress in Graph Theory, J. A. Bondy, and U. S. R. Murty, eds., Academic Press, Canada, 1984, pp. 43–69.
[23] S. Tsukiyama, K. Koike, and I. Shirakawa, *An Algorithm to Eliminate All Complex Triangles in a Maximal Planar Graph for Use in VLSI Floorplan*, in Proceedings of the IEEE Int. Symp. on Circuits and Systems, 1986, pp. 321–324.
[24] W. T. Tutte, *On Hamiltonian circuits*, J. London Math. Soc., 21 (1946), pp. 98–101.
[25] K.-H. Yeap and M. Sarrafzadeh, *Floor-planning by graph dualization:* 2*-concave rectilinear modules*, SIAM J. Comput., 22 (1993), pp. 500–526.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2168–2186

HORN EXTENSIONS OF A PARTIALLY DEFINED BOOLEAN FUNCTION*

KAZUHISA MAKINO†, KEN-ICHI HATANAKA‡, AND TOSHIHIDE IBARAKI§

Abstract. Given a partially defined Boolean function (pdBf) (T, F), we investigate in this paper how to find a Horn extension $f : \{0,1\}^n \mapsto \{0,1\}$, which is consistent with (T, F), where $T \subseteq \{0,1\}^n$ denotes a set of true Boolean vectors (or positive examples) and $F \subseteq \{0,1\}^n$ denotes a set of false Boolean vectors (or negative examples). Given a pdBf (T, F), it is known that the existence of a Horn extension can be checked in polynomial time. As there are many Horn extensions, however, we consider those extensions f which have maximal and minimal sets $T(f)$ of the true vectors of f, respectively. For a pdBf (T, F), there always exists the unique maximal (i.e., maximum) Horn extension, but there are in general many minimal Horn extensions. We first show that a polynomial time membership oracle can be constructed for the maximum extension, even if its disjunctive normal form (DNF) can be very long. Our main contribution is to show that checking if a given Horn DNF represents a minimal extension and generating a Horn DNF of a minimal Horn extension can both be done in polynomial time. We also can check in polynomial time if a pdBf (T, F) has the unique minimal Horn extension. However, the problems of finding a Horn extension f with the smallest $|T(f)|$ and of obtaining a Horn DNF, whose number of literals is smallest, are both NP-hard.

Key words. partially defined Boolean function, extension, Horn function, knowledge acquisition

AMS subject classifications. 68Q25, 68R05, 68T01

PII. S0097539796297954

1. Introduction.

Knowledge acquisition in the form of Boolean logic has been intensively studied (e.g., [2, 4, 6, 18, 20, 24]): Given a set of data, represented as a set $T \subseteq \{0,1\}^n$ of binary "true n-vectors" (or "positive examples") and a set $F \subseteq \{0,1\}^n$ of "false n-vectors" (or "negative examples"), establish a (fully defined) Boolean function (i.e., *extension*) $f : \{0,1\}^n \mapsto \{0,1\}$ in a specified class $\mathcal{C}$, such that $T \subseteq T(f)$ and $F \subseteq F(f)$, where $T(f)$ (resp., $F(f)$) denotes the set of true (resp., false) vectors of f. A pair of sets (T, F) is called a *partially defined Boolean function* (*pdBf*) throughout this paper.

For instance, a vector x may represent the symptoms used to diagnose a disease; e.g., x_1 denotes whether temperature is high ($x_1 = 1$) or not ($x_1 = 0$), and x_2 denotes whether blood presure is high ($x_2 = 1$) or not ($x_2 = 0$), etc. Each vector x in T corresponds to a case of symptoms that caused the disease, while a vector in F describes a case with which the disease did not appear. Establishing an extension f, which is consistent with the given data, amounts to finding a logical diagnostic explanation of the given data.

In this paper, we consider the case in which f is a Horn function [15]. The class of Horn functions is at the heart of knowledge-based systems [1, 12, 5] and motivates increasing research, e.g., minimum representations [13, 14], their learning

*Received by the editors January 29, 1996; accepted for publication (in revised form) September 3, 1998; published electronically July 7, 1999. This work was partially supported by a grant from the Ministry of Education, Sports, Science, and Culture of Japan.
http://www.siam.org/journals/sicomp/28-6/29795.html

†Department of Systems and Human Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka, 560, Japan (makino@sys.es.osaka-u.ac.jp).
‡Sumitomo Electric Industries, Shimaya 1-1-3, Konohana-ku, Osaka, 554, Japan (khata@sei.co.jp).
§Department of Applied Mathematics and Physics, Graduate School of Engineering, Kyoto University, Kyoto, 606, Japan (ibaraki@kuamp.kyoto-u.ac.jp).

and identification [1, 8], and constructing Horn approximations [17, 25]. One of the main reasons for this attention is that the *satisfiability problem (SAT)* of a Horn conjunctive normal form (CNF) (H-SAT in short) can be solved in polynomial time [7], whereas the SAT of a general CNF is NP-complete [11]. As problem SAT of CNF is fundamental, many problems related to Horn functions can be solved efficiently. In terms of sets $T(f)$ and $F(f)$, a Horn function has an elegant characterization: f is Horn if and only if $F(f)$ is closed under intersection of vectors (i.e., $v, w \in F(f)$ implies $v \bigwedge w \in F(f)$, where $\bigwedge$ denotes the componentwise AND operation).

Because there are in general many Horn extensions f for a given pdBf (T, F), we shall mainly consider those extensions that are *maximal* and *minimal* in the sense of set $T(f)$, respectively. We note here that most of the papers written on the representation by Horn theory (e.g., [4, 8, 17, 18]) are based on model theory, in which finding a Horn representation f of a given model $(T(g), F(g))$, where g is a Boolean function and sets $T(g)$ and/or $F(g)$ of vectors are explicitly given, is a primary target. For example, the problem of finding the best Horn approximation of a model $(T(g), F(g))$, i.e., finding the Horn function with the minimum $|F(f)|$ under the constraint $F(f) \supseteq F(g)$, has received some attention [18], and it is known [18, 19] that obtaining an irredundant disjunctive normal form (DNF) of such an f is at least as difficult as computing the DNF of the dual h^d of a positive (i.e., monotone) Boolean function h. The latter problem is a well-known open problem [3, 9, 16], for which the recent result of Fredman and Khachiyan [10] shows that there is an $O(m^{o(\log m)})$ time algorithm, where m is the total length of DNFs for both h and h^d. We emphasize that our problem setting is different from model theory in that the input (T, F) is only partially defined. However, the above problem of best Horn approximation is very close to the problem of finding a maximal Horn extension. We also note that, although the problem of finding a best approximation in terms of $T(g)$ is a bit artificial (since $T(g)$ is not closed under intersection), finding a minimal Horn extension of a pdBf (T, F) is quite a natural problem in our framework.

It is known [4] that the existence of at least one Horn extension of a given pdBf (T, F) can be checked in polynomial time. After preparing necessary notation and definitions in section 2 and introducing canonical Horn DNFs in section 3, we proceed to maximal and minimal Horn extensions. In section 4, by using an argument similar to the one used in model theory, we show that there exists the unique maximal Horn extension $f_{\max}$ (i.e., maximum) and we provide a polynomial time membership oracle for $f_{\max}$. In section 5, we investigate minimal Horn extensions. Contrary to the case of maximum Horn extension, there are in general many minimal Horn extensions. Our main contribution is to show that the minimality of f_φ, which denotes the function represented by a Horn DNF φ, can be checked in polynomial time. Based on this, a minimal Horn extension of a pdBf (T, F) can be generated in polynomial time and the uniqueness of a minimal extension can also be checked in polynomial time.

To derive the above results, we first show that any minimal Horn extension can be represented by a *canonical* Horn DNF, although the converse is not true. The nontriviality of finding a canonical DNF representing a minimal Horn extension may be exemplified by the existence of a canonical DNF that satisfies *local minimality* but does not represent a minimal Horn extension. To overcome this, we reduce the nonminimality condition to the condition that some CNF Φ_v, $v \in T$, is satisfiable, where Φ_v is a CNF derived from canonical DNF φ, $v \in T$, and (T, F). Although this does not immediately give a polynomial time algorithm, since Φ_v is not Horn, we then derive a series of lemmas, with which (non-Horn) CNFs Φ_v can be eventually

transformed into Horn CNFs Φ_v^*. Therefore, the minimality condition can be checked in polynomial time.

Finally, we show in sections 5 and 6 that the problems of computing a Horn extension f with the minimum $|T(f)|$ and of finding the shortest Horn DNF (i.e., having the smallest number of literals) that represents a Horn extension are both NP-hard. It is still not known whether there exists a polynomial total time algorithm to generate all minimal Horn extensions of a given pdBf (T, F).

2. Preliminaries. A *Boolean function* (or a *function*) is a mapping $f : \{0,1\}^n \mapsto \{0,1\}$, where $x \in \{0,1\}^n$ is called a *Boolean vector* (or a *vector*). If $f(x) = 1$ (resp., 0), then x is called a *true* (resp., *false*) vector of f. The set of all true vectors (resp., false vectors) is denoted by $T(f)$ (resp., $F(f)$). Denote, for a vector $v \in \{0,1\}^n$, $ON(v) = \{j \mid v_j = 1,\ j = 1,2,\ldots,n\}$ and $OFF(v) = \{j \mid v_j = 0,\ j = 1,2,\ldots,n\}$. For vectors $v, w \in \{0,1\}^n$, we write $v \le w$ (resp., $v \ge w$) if $v_i \le w_i$ (resp., $v_i \ge w_i$) holds for all $i = 1,2,\ldots,n$. Two special functions with $T(f) = \emptyset$ and $F(f) = \emptyset$ are, resp., denoted by $f = \bot$ and $f = \top$. For two functions f and g on the same set of variables, we write $f \le g$ if $f(x) = 1$ implies $g(x) = 1$ for any $x \in \{0,1\}^n$ and $f < g$ if $f \le g$ and $f \ne g$.

Boolean variables $x_1,\ldots,x_n$ and their negations $\bar{x}_1,\ldots,\bar{x}_n$ are called *literals*, where we call literals $x_1,\ldots,x_n$ *positive* and literals $\bar{x}_1,\ldots,\bar{x}_n$ *negative*. A *term* t is a conjunction of literals such that at most one of x_i and $\bar{x}_i$ appears for each i. The constant 1 (viewed as the conjunction of an empty set of literals) is also considered a term. We say that a term t *subsumes* a term t' if $t \ge t'$, where terms t and t' are considered the functions they represent. For example, a term $x\bar{y}$ subsumes a term $x\bar{y}z$. A term t is called an *implicant* of a function f if $t \le f$. An implicant t of a function is called *prime* if there is no implicant $t' > t$.

A DNF φ is a disjunction of terms. It is well known that a DNF φ defines a function, which we denote by f_φ, and any function can be represented by a DNF (however, such a representation may not be unique). In this paper, we do not always distinguish a DNF φ from the function f_φ it represents. For example, a term t is also considered as the function f_t. The number of literals in a DNF φ is denoted by $|\varphi|$. In this paper, we shall deal exclusively with DNF expressions, although some of the literature on Horn functions is based on CNFs. By complementing the involved concepts, all the results in this paper can be translated into the results for CNFs.

A term is called *positive* if it contains only positive literals and is called *Horn* if it contains at most one negative literal. A DNF is called *positive* if it contains only positive terms and is called *Horn* if it contains only Horn terms. For example, a DNF $\varphi = 123 \vee 245 \vee 156$ is positive and $\psi = 15\bar{7} \vee 24 \vee \bar{2}67$ is Horn. (Here, for simplicity, a positive literal x_i is denoted as i and a negative literal $\bar{x}_i$ as $\bar{i}$.) It is easy to see that, by complementing Horn DNFs, we obtain Horn CNFs, where a CNF $\Phi = \bigwedge_i C_i$ is Horn if each clause C_i contains at most one positive literal, e.g., $(1 \vee \bar{2} \vee \bar{3})(\bar{1} \vee \bar{3})(\bar{1} \vee \bar{3} \vee 4)$ is Horn, while $(1 \vee \bar{2} \vee \bar{3})(\bar{1} \vee \bar{3})(1 \vee \bar{3} \vee 4)$ is not Horn. A Boolean function is called *positive* (or *monotone*) if it can be represented by a positive DNF and *Horn* if it can be represented by a Horn DNF. It is known [14] that if f is a Horn function, then all prime implicants of f are Horn. It is important to know that the following two variants of SAT for a Horn CNF Φ can be solved in time linear in $|\Phi|$ [7, 22]:

Problem H-SAT
Input: A Horn CNF Φ of n variables.
Question: Is there a vector $u \in \{0,1\}^n$ satisfying $\Phi(u) = 1$?

Problem UNIQUE-H-SAT

Input: A Horn CNF Φ of n variables.

Question: Is there a unique vector $u \in \{0,1\}^n$ satisfying $\Phi(u) = 1$?

If the answer to these problems is "yes," the vector u satisfying $\Phi(u) = 1$ can also be output in linear time. If the answer to UNIQUE-H-SAT is "no," two vectors u and v satisfying $\Phi(u) = \Phi(v) = 1$ can also be output in linear time. Based on these, various problems associated with Horn functions can be solved in polynomial time. For example, given two Horn DNFs φ and ψ, the conditions such as $f_\varphi = f_\psi$ and $f_\varphi < f_\psi$ can be checked in $O(|\varphi||\psi|)$ time [14]. Also, for a term t (not necessarily Horn), condition $t \leq f_\varphi$ can be checked in $O(|\varphi|)$ time [14].

A pdBf is defined by a pair of sets (T, F) satisfying $T \cap F = \emptyset$, where $T, F \subseteq \{0,1\}^n$. A function f is an *extension* (or theory) of the pdBf (T, F) if $T \subseteq T(f)$ and $F \subseteq F(f)$, and it is a *Horn extension* if f is in addition Horn. We sometimes refer to a Horn DNF representing a Horn extension of a pdBf (T, F) as a Horn DNF of (T, F). A Horn extension f of a pdBf (T, F) is called *minimal* (resp., *maximal*) if there is no Horn extension f' satisfying $f' < f$ (resp., $f' > f$), that is, set $T(f)$ is minimal (resp., maximal). Furthermore, a Horn extension f of a pdBf (T, F) is *minimum* (resp., *maximum*) if there is no Horn extension f' such that $|T(f')| < |T(f)|$ (resp., $|T(f')| > |T(f)|$). Obviously, a minimum (resp., maximum) Horn extension is one of the minimal (resp., maximal) Horn extensions.

Example 2.1. Let (T, F) be a pdBf defined by $T = \{1110, 0011, 0101\}$ and $F = \{0010, 1100, 0110\}$. Then, by generating all extensions of (T, F), we can see that the unique maximum Horn extension of (T, F) is represented by the DNF

$$\varphi = 13 \vee 4 \vee 1\bar{2}$$

while there are two minimal Horn extensions:

$$\psi^{(1)} = 123\bar{4} \vee \bar{1}34 \vee \bar{1}24,$$
$$\psi^{(2)} = 123\bar{4} \vee \bar{2}34 \vee 2\bar{3}4.$$

Furthermore, $\psi^{(1)}$ represents a minimum Horn extension of (T, F).

3. Canonical Horn DNF. In this section, we first review the following fundamental problem, which was originally discussed in [4]:

Problem H-EXTENSION

Input: A pdBf (T, F).

Question: Is there a Horn extension f of (T, F)?

We point out that the following well-known characterization of a Horn function provides a polynomial time algorithm to solve H-EXTENSION. Call the componentwise AND operation $\bigwedge$ of vectors v and w the *intersection* of v and w. For example, if $v = (0101)$ and $w = (1001)$, then $v \bigwedge w = (0001)$. For a set $X \subseteq \{0,1\}^n$, the set of vectors $C(X)$ is called the *intersection closure* if it is a minimal set that contains X and is closed under intersection. Clearly, intersection closure is unique (i.e., "minimal" can be replaced by "minimum").

PROPOSITION 3.1 (see [21, 8]). *A function f is Horn if and only if $F(f) = C(F(f))$ (i.e., $F(f)$ is closed under intersection).*

The next definition provides a means to generate Horn DNFs from (T, F).

DEFINITION 3.1. *For a pdBf (T, F) and a vector $v \in T$, the set of terms $R(v)$ is*

defined by

$$R(v) = \begin{cases} \{\bigwedge_{j\in ON(v)} x_j\} & \text{if } OFF(v) = \emptyset, \\ \{(\bigwedge_{j\in ON(v)} x_j)\,\bar{x}_l \mid l \in I(v)\} & \text{if } OFF(v) \neq \emptyset \text{ and } I(v) \neq \emptyset, \\ \emptyset & \text{if } OFF(v) \neq \emptyset \text{ and } I(v) = \emptyset, \end{cases}$$

where

$$F_{\geq v} = \{w \in F \mid w \geq v\},$$
$$I(v) = (\cap_{w\in F_{\geq v}} ON(w)) \cap OFF(v).$$

By convention, we define $I(v) = OFF(v)$ *if* $F_{\geq v} = \emptyset$. *A DNF* φ *is called a* canonical *Horn DNF of* (T, F) *if* φ *is given by*

$$\varphi = \bigvee_{v\in T} t_v, \text{ where } t_v \in R(v), \tag{3.1}$$

i.e., by selecting one term from each $R(v)$, $v \in T$. *Note that the canonical Horn DNF is not defined if* $R(v) = \emptyset$ *holds for some* $v \in T$.

For a $v \in T$, there are many Horn terms t such that $t(v) = 1$. However, in order to satisfy $t(w) = 0$ for all $w \in F$, we can restrict the negative literal $\bar{x}_l$, which appears in t. The above definition says that $I(v)$ represents the set of such indices l and $R(v)$ represents the particular subset of terms t such that $t(v) = 1$ and $t(w) = 0$ for all $w \in F$. Construction of Horn DNFs in this manner can be found in the literature of learning theory [1], model theory [17], and Horn approximation [25]. Precisely speaking, however, the above canonical DNF is different from those used in the literature in that both T and F are explicitly taken into account.

Example 3.1. Let us define $T, F \subseteq \{0,1\}^9$ by

$$T = \left\{ \begin{array}{lcl} v^{(1)} & = & (111100100) \\ v^{(2)} & = & (111010100) \\ v^{(3)} & = & (111001010) \\ v^{(4)} & = & (001000100) \\ v^{(5)} & = & (100000100) \\ v^{(6)} & = & (011000001) \\ v^{(7)} & = & (110000001) \\ v^{(8)} & = & (111111000) \end{array} \right\}, \quad F = \left\{ \begin{array}{lcl} w^{(1)} & = & (111100110) \\ w^{(2)} & = & (111010111) \\ w^{(3)} & = & (111001110) \\ w^{(4)} & = & (111000101) \end{array} \right\}.$$

Then $F_{\geq v^{(1)}} = \{w^{(1)}\}$, $F_{\geq v^{(2)}} = \{w^{(2)}\}$, $F_{\geq v^{(3)}} = \{w^{(3)}\}$, $F_{\geq v^{(4)}} = F_{\geq v^{(5)}} = \{w^{(1)}, w^{(2)}, w^{(3)}, w^{(4)}\}$, $F_{\geq v^{(6)}} = F_{\geq v^{(7)}} = \{w^{(2)}, w^{(4)}\}$, and $F_{\geq v^{(8)}} = \emptyset$.

$$\begin{array}{ll} I(v^{(1)}) = \{8\}, & R(v^{(1)}) = \{12347\bar{8}\}, \\ I(v^{(2)}) = \{8,9\}, & R(v^{(2)}) = \{12357\bar{8}, 12357\bar{9}\}, \\ I(v^{(3)}) = \{7\}, & R(v^{(3)}) = \{1236\bar{7}8\}, \\ I(v^{(4)}) = \{1,2\}, & R(v^{(4)}) = \{\bar{1}37, \bar{2}37\}, \\ I(v^{(5)}) = \{2,3\}, & R(v^{(5)}) = \{1\bar{2}7, 1\bar{3}7\}, \\ I(v^{(6)}) = \{1,7\}, & R(v^{(6)}) = \{\bar{1}239, 23\bar{7}9\}, \\ I(v^{(7)}) = \{3,7\}, & R(v^{(7)}) = \{12\bar{3}9, 12\bar{7}9\}, \\ I(v^{(8)}) = \{7,8,9\}, & R(v^{(8)}) = \{1234567, 1234568, 1234569\}. \end{array}$$

There are $1 \times 2 \times 1 \times 2 \times 2 \times 2 \times 2 \times 3 = 96$ canonical Horn DNFs, among which we list the following two:

$$\begin{array}{rcl} \varphi^{(1)} & = & 12347\bar{8} \vee 12357\bar{9} \vee 1236\bar{7}8 \vee \bar{1}37 \vee 1\bar{3}7 \vee 23\bar{7}9 \vee 12\bar{7}9 \vee 123456\bar{7}, \\ \varphi^{(2)} & = & 12347\bar{8} \vee 12357\bar{9} \vee 1236\bar{7}8 \vee \bar{1}37 \vee 1\bar{3}7 \vee \bar{1}239 \vee 12\bar{3}9 \vee 123456\bar{9}. \end{array} \tag{3.2}$$

LEMMA 3.1 (see [4]). *Any canonical Horn DNF φ of a given pdBf (T, F) represents a Horn extension of (T, F), and (T, F) has no Horn extension if there is no canonical Horn DNF.*

Proof. Let $\varphi = \bigvee_{v \in T} t_v$ be a canonical Horn DNF of a pdBf (T, F). It is clear that, for each $v \in T$, we have $t_v(v) = 1$ and $t_v(w) = 0$ for all $w \in F$. This implies that φ represents a Horn extension. Conversely, if there is no canonical Horn DNF, then $R(v) = \emptyset$ holds for some $v \in T$, i.e., $OFF(v) \neq \emptyset$ and $I(v) = \emptyset$ holds for some $v \in T$. This means $(\cap_{w \in F_{\geq v}} ON(w)) \cap OFF(v) = \emptyset$; i.e., $\bigwedge_{w \in F_{\geq v}} w = v$. Therefore $F(f)$ of no Horn extension f of (T, F) is closed under intersection, and there is no Horn extension by Proposition 3.1. □

Therefore, we have the following results.

THEOREM 3.1 (see [4]). *Problem H-EXTENSION can be solved in $O(n|T||F|)$ time, and if a pdBf (T, F) has a Horn extension, one of its canonical Horn DNFs can be obtained in $O(n|T||F|)$ time.*

Proof. The proof is immediate from the above discussion and the fact that a canonical Horn DNF of (T, F) can be constructed in $O(n|T||F|)$ time. □

4. Maximum Horn extension. In this section, we first show the uniqueness of a maximal Horn extension.

THEOREM 4.1. *If a given pdBf (T, F) has a Horn extension, its maximal Horn extension is unique.*

Proof. By Proposition 3.1, $F(f)$ of any Horn extension f of (T, F) is closed under intersection. Let us define $f_{\max}$ by $F(f_{\max}) = C(F)$, that is,

$$f_{\max}(v) = \begin{cases} 0 & \text{if } v \in C(F), \\ 1 & \text{otherwise.} \end{cases} \tag{4.1}$$

Since $C(F)$ is the unique minimal set that contains F and is closed under intersection, this $f_{\max}$ is the unique maximal Horn extension (that is, $T(f)$ is maximal) of (T, F). □

Unfortunately, it is known [17] that there is a pdBf (T, F) for which the size of any DNF φ of $f_{\max}$ is exponential in n, $|T|$, and $|F|$. In other words, there may not be any compact DNF representation of $f_{\max}$. However, we can do better if we do not stick to the DNF representation. Note that $f_{\max}$ of (4.1) is defined by $C(F)$, for which $v \in C(F)$ holds if and only if

$$\bigwedge_{w \in F_{\geq v}} w = v.$$

As this condition can be checked in polynomial time in n and $|F|$ for a given v, we can build an oracle that answers membership queries for $f_{\max}$ in polynomial time.

A vector $x \in X \subseteq \{0, 1\}^n$ is called *extreme* [8] with respect to a set X if $x \notin C(X \setminus \{x\})$. The set of all extremal vectors of X is called the *characteristic set* of X [17, 19] (or its *base* [8]) and is denoted by $C^*(X)$. Note that every set $X \subseteq \{0, 1\}^n$ has the unique characteristic set $C^*(X)$ and that $C^*(X) \subseteq X$ is the minimum set satisfying $C(C^*(X)) = C(X)$. It is known [8] that $C^*(X)$ can be constructed from X in polynomial time in n and $|X|$; therefore $C^*(F) = C^*(C(F))$ can be computed from F of (T, F) in polynomial time. There are a number of papers on the relationship between $C^*(F(f))$ of a Horn function f and its Horn DNF expression φ [9, 17, 19]. For example, there is a *polynomial total time* algorithm (i.e., polynomial algorithm in the length of input and output) for computing from $C^*(F(f))$ all prime

implicants of Horn DNF φ that represents f if and only if there is a polynomial total time algorithm for dualizing a positive function h (i.e., computing all prime implicants of h^d from all prime implicants of h, where $h^d(x) = \bar{h}(\bar{x})$); if there is a polynomial total time algorithm for computing from $C^*(F(f))$ an *irredundant* Horn DNF φ that represents f (i.e., no term in φ can be dropped), then there is a polynomial total time algorithm for dualizing a positive function. From the viewpoint of $f_{\max}$ (whose $C^*(F(f_{\max})) = C^*(F)$ can be computed in polynomial time), this shows that computing an irredundant Horn DNF φ of $f_{\max}$ is at least as hard as dualizing a positive function. It is not known yet [3, 9, 16] whether or not the problem of dualizing a positive function has a polynomial total time algorithm. However, the recent result by Fredman and Khachiyan [10] shows that dualizing a positive function can be done in $O(m^{o(\log m)})$ time, where m denotes the number of prime implicants of f and f^d, and hence it is unlikely for the problem to be NP-hard.

5. Minimal Horn extensions. There are in general many minimal Horn extensions of a given pdBf (T, F). However, these minimal Horn extensions can all have canonical Horn DNFs of Definition 3.1.

LEMMA 5.1. *A minimal Horn extension f of a given pdBf (T, F) can always be represented by a canonical Horn DNF.*

Proof. Assume that there exists a minimal Horn extension f, which cannot be represented by a canonical DNF. Since f is a Horn extension, for every $v \in T$, there is a Horn implicant $t_v = \bigwedge_{i \in P} x_i \bigwedge_{i \in N} \bar{x}_i$ of f such that $t_v(v) = 1$ (i.e., $P \subseteq ON(v)$ and $N \subseteq OFF(v)$) and $|N| \leq 1$. Then by the definition of $R(v)$, there is a term $t'_v \in R(v)$ such that $t'_v \leq t_v$. Define a canonical Horn DNF $\varphi = \bigvee_{v \in T} t'_v$. This φ satisfies $f_\varphi < f$ since $f_\varphi \leq f$ holds and f cannot be represented by a canonical Horn DNF φ, contradicting the minimality of f. □

The converse, however, is not true (i.e., some canonical Horn DNFs do not represent minimal Horn extensions), as will be shown in Example 5.1 in the next subsection. Recall that a Horn DNF φ representing a Horn extension of (T, F) is called a Horn DNF of a pdBf (T, F). Furthermore, we say that a Horn DNF φ of (T, F) is *minimal* if f_φ is a minimal Horn extension of (T, F). It is interesting to know whether the following problem can be solved in polynomial time, where we assume that a Horn DNF φ (not necessarily canonical) is given as an input:

Problem MINIMAL-H-EXTENSION

Input: A pdBf (T, F) and a Horn DNF φ.

Question: Is φ a minimal Horn DNF of (T, F)?

In passing, we note an interesting implication of Lemma 5.1: all minimal Horn extensions of (T, F) have "short" DNFs in the sense that all canonical Horn DNFs have only $|T|$ terms, respectively. This contrasts with the fact that the DNFs of some maximum Horn extensions have exponentially many terms, as noted after Theorem 4.1.

5.1. Checking the minimality of a Horn DNF. We show via a series of lemmas in this subsection that MINIMAL-H-EXTENSION can be solved in polynomial time. In the following, we assume without loss of generality that $(1, 1, \ldots, 1) \notin T$ holds, because it can be shown that, for a pdBf (T, F) with $(1, 1, \ldots, 1) \in T$, $\varphi \vee (\wedge_{j=1}^n x_j)$ is a minimal Horn DNF of (T, F) if and only if φ is a minimal Horn DNF of $(T \setminus \{(1, 1, \ldots, 1)\}, F)$. For a pdBf (T, F), a vector $v \in T$, and a Horn DNF

φ of (T, F), define

$$I(v;\varphi) = \{l \in I(v) \mid (\textstyle\bigwedge_{j\in ON(v)} x_j)\,\bar{x}_l \le f_\varphi\},$$
$$R(v;\varphi) = \{t \in R(v) \mid t \le f_\varphi\}\ (= \{(\textstyle\bigwedge_{j\in ON(v)} x_j)\,\bar{x}_l \mid l \in I(v;\varphi)\}).$$

Note that Lemma 5.1 implies $I(v;\varphi) \neq \emptyset$ and $R(v;\varphi) \neq \emptyset$ for all $v \in T$. By Lemma 5.1, if a Horn DNF φ of (T, F) is not minimal, then there exists a canonical Horn DNF ψ of (T, F) such that $f_\psi < f_\varphi$, where ψ can be written as

$$\psi = \bigvee_{v\in T} t_v; \quad t_v \in R(v;\varphi).$$

It is known that the candidate set of terms $R(v;\varphi)$ can be computed in polynomial time. More precisely, given a canonical Horn DNF φ and a vector $v \in T$, set $I(v;\varphi)$ can be constructed in time linear in $|\varphi|$ by using the following forward chaining procedure [13].

Algorithm F-CHAINING

Input: A Horn DNF φ and a vector $v \in T$.

Output: Set $I(v;\varphi)$.

Step 1: $S := ON(v)$ and $I(v;\varphi) := \emptyset$.

Step 2: If there exists a term $t = (\wedge_{i\in S'} x_i)\bar{x}_l$ in φ such that $S' \subseteq S$ and $l \notin S$, let $S := S \cup \{l\}$. Repeat Step 2 until no term in φ satisfies the condition.

Step 3: $I(v;\varphi) := S \setminus ON(v)$.

The essential part of this algorithm comes from the *consensus* procedure [23]. Given a DNF φ, the consensus procedure generates a new implicant $\bigwedge_{j\in P_1\cup P_2} x_j \bigwedge_{j\in N_1\cup N_2} \bar{x}_j$ from two implicants $x_i\,(\bigwedge_{j\in P_1} x_j \bigwedge_{j\in N_1} \bar{x}_j)$ and $\bar{x}_i\,(\bigwedge_{j\in P_2} x_j \bigwedge_{j\in N_2} \bar{x}_j)$ such that $i \notin P_k \cup N_k$ for $k = 1, 2$, and $P_1 \cap N_2 = N_1 \cap P_2 = \emptyset$. For example, $2\bar{3}4\bar{5}6$ is generated from $1\bar{3}4\bar{5}$ and $\bar{1}2\bar{3}46$. It is known [23] that all prime implicants of f_φ eventually can be generated by the consensus procedure starting from the terms of φ. Since every prime implicant of a Horn function is Horn, Algorithm F-CHAINING works correctly.

Example 5.1. Consider the pdBf (T, F) given in Example 3.1 and choose two canonical Horn DNFs $\varphi^{(1)}$ and $\varphi^{(2)}$ of (3.2). Then

$$I(v^{(1)};\varphi^{(1)}) = \{8\},\quad I(v^{(1)};\varphi^{(2)}) = \{8\},$$
$$I(v^{(2)};\varphi^{(1)}) = \{9\},\quad I(v^{(2)};\varphi^{(2)}) = \{9\},$$
$$I(v^{(3)};\varphi^{(1)}) = \{7\},\quad I(v^{(3)};\varphi^{(2)}) = \{7\},$$
$$I(v^{(4)};\varphi^{(1)}) = \{1\},\quad I(v^{(4)};\varphi^{(2)}) = \{1\},$$
$$I(v^{(5)};\varphi^{(1)}) = \{3\},\quad I(v^{(5)};\varphi^{(2)}) = \{3\},$$
$$I(v^{(6)};\varphi^{(1)}) = \{1, 7\},\quad I(v^{(6)};\varphi^{(2)}) = \{1\},$$
$$I(v^{(7)};\varphi^{(1)}) = \{3, 7\},\quad I(v^{(7)};\varphi^{(2)}) = \{3\},$$
$$I(v^{(8)};\varphi^{(1)}) = \{7, 8, 9\},\quad I(v^{(8)};\varphi^{(2)}) = \{9\},$$

and

$$R(v^{(1)};\varphi^{(1)}) = \{1234\bar{7}\bar{8}\},\quad R(v^{(1)};\varphi^{(2)}) = \{1234\bar{7}\bar{8}\},$$
$$R(v^{(2)};\varphi^{(1)}) = \{123\bar{5}\bar{7}\bar{9}\},\quad R(v^{(2)};\varphi^{(2)}) = \{123\bar{5}\bar{7}\bar{9}\},$$
$$R(v^{(3)};\varphi^{(1)}) = \{12\bar{3}6\bar{7}\bar{8}\},\quad R(v^{(3)};\varphi^{(2)}) = \{12\bar{3}6\bar{7}\bar{8}\},$$
$$R(v^{(4)};\varphi^{(1)}) = \{\bar{1}37\},\quad R(v^{(4)};\varphi^{(2)}) = \{\bar{1}37\},$$
$$R(v^{(5)};\varphi^{(1)}) = \{1\bar{3}7\},\quad R(v^{(5)};\varphi^{(2)}) = \{1\bar{3}7\},$$
$$R(v^{(6)};\varphi^{(1)}) = \{\bar{1}239, 23\bar{7}9\},\quad R(v^{(6)};\varphi^{(2)}) = \{\bar{1}239\},$$
$$R(v^{(7)};\varphi^{(1)}) = \{12\bar{3}9, 12\bar{7}9\},\quad R(v^{(7)};\varphi^{(2)}) = \{12\bar{3}9\},$$
$$R(v^{(8)};\varphi^{(1)}) = \{1234567, 1234568, 1234569\},\quad R(v^{(8)};\varphi^{(2)}) = \{1234569\}.$$

The two functions $f_{\varphi^{(1)}}$ and $f_{\varphi^{(2)}}$ satisfy $f_{\varphi^{(1)}} \geq f_{\varphi^{(2)}}$, since both are canonical and $R(v^{(l)};\varphi^{(1)}) \supseteq R(v^{(l)};\varphi^{(2)})$ holds for all l. Furthermore, $23\bar{7}9 \leq f_{\varphi^{(1)}}$ and $23\bar{7}9 \not\leq f_{\varphi^{(2)}}$ imply $f_{\varphi^{(1)}} > f_{\varphi^{(2)}}$. Therefore, $\varphi^{(1)}$ is not minimal. However, this $\varphi^{(1)}$ satisfies local minimality in the sense that, after replacing one of its terms t_v by $t'_v \in R(v^{(l)};\varphi^{(1)}) \setminus \{t_v\}$ for any l with $|R(v^{(l)};\varphi^{(1)})| > 1$, the resulting DNF also represents $f_{\varphi^{(1)}}$. For example, the following two DNFs also represent the same $f_{\varphi^{(1)}}$:

$$\begin{aligned} \varphi^{(3)} &= 1234\bar{7}\bar{8} \vee 1235\bar{7}\bar{9} \vee 1236\bar{7}\bar{8} \vee \bar{1}\bar{3}7 \vee 1\bar{3}7 \vee 23\bar{7}9 \vee 12\bar{7}9 \vee 1234\bar{5}\bar{6}, \\ \varphi^{(4)} &= 1234\bar{7}\bar{8} \vee 1235\bar{7}\bar{9} \vee 1236\bar{7}\bar{8} \vee \bar{1}\bar{3}7 \vee 1\bar{3}7 \vee \bar{1}239 \vee 12\bar{7}9 \vee 1234\bar{5}\bar{7}. \end{aligned}$$

This result shows that local minimality of φ (in the above sense) does not always imply its minimality. Therefore, some other proof is necessary to ensure the minimality. Of course, if we replace more than one term in $\varphi^{(1)}$, the resulting DNF may represent a different function; $\varphi^{(2)}$ is such an example.

On the other hand, $\varphi^{(2)}$ is minimal since $|R(v^{(l)};\varphi^{(2)})| = 1$ for all $l = 1, 2, \ldots, 8$. However, this is not always the case, since there can be a minimal φ with $|R(v^{(l)};\varphi)| > 1$ for some l. For example, consider a pdBf (T, F) defined by $T = \{v^{(1)} = (1100), v^{(2)} = (1010), v^{(3)} = (0101)\}$ and $F = \emptyset$, and a canonical DNF $\varphi = 12\bar{3} \vee 13\bar{4} \vee 2\bar{3}4$. Then $I(v^{(1)};\varphi) = \{3, 4\}$, $I(v^{(2)};\varphi) = \{4\}$, and $I(v^{(3)};\varphi) = \{3\}$; that is, $R(v^{(1)};\varphi) = \{12\bar{3}, 12\bar{4}\}$, $R(v^{(2)};\varphi) = \{13\bar{4}\}$, and $R(v^{(3)};\varphi) = \{2\bar{3}4\}$. However, since it is easy to see that $\varphi' = 12\bar{4} \vee 13\bar{4} \vee 2\bar{3}4$ satisfies $f_{\varphi'} = f_\varphi$, there is no canonical DNF ψ such that $\psi < \varphi$; hence φ is minimal.

Example 5.1 may suggest that Problem MINIMAL-H-EXTENSION is not trivial. Let us now examine the condition when φ is *not* minimal.

Let f be a Horn extension of (T, F). Then f is not minimal if and only if there is a nonempty subset of $T(f)\backslash T$, whose removal from $T(f)$ results in a new Horn extension. This means that there exists a vector $u \in T(f)$ such that $C(F(f) \cup \{u\}) \cap T = \emptyset$ holds, where $C(X)$ denotes the intersection closure of X. In other words, $u \wedge \bigwedge_{w \in S} w$ is different from any vector in T for all $S \subseteq F(f)$. Since $u \wedge \bigwedge_{w \in S} w = a$ holds for some $a \in T$ and $S \subseteq F(f)$ if and only if it holds for $S = F(f)_{\geq a}$, where $F(f)_{\geq a} = \{w \in F(f) \,|\, w \geq a\}$, this argument leads to the following lemma.

LEMMA 5.2. *Let f be a Horn extension of a pdBf (T, F). Then f is not minimal if and only if there exists a vector $u \in T(f)$ such that*

$$u \wedge \bigwedge_{w \in F(f)_{\geq a}} w \neq a \quad \text{for all } a \in T. \tag{5.1}$$

Assume that f can be represented by a canonical Horn DNF φ of a pdBf (T, F) (i.e., $f = f_\varphi$). Then $y = \bigwedge_{w \in F(f)_{\geq a}} w$ for an $a \in T$ is given by $ON(y) = ON(a) \cup I(a;\varphi)$, because, by the definition of $I(a;\varphi)$, all vectors w such that $w \geq a$ and $w_l = 0$ for some $l \in I(a;\varphi)$ satisfy $\varphi(w) = 1$ (i.e., $w \notin F(f)$), and, for every $l \in OFF(a) \setminus I(a;\varphi)$, there is a vector $w \in F(f)_{\geq a}$ such that $w_l = 0$ (since otherwise l must be included in $I(a;\varphi)$). In other words, $u \wedge \bigwedge_{w \in F(f)_{\geq a}} w = a$ holds for an $a \in T$ if and only if u satisfies $ON(u) \supseteq ON(a)$ and $OFF(u) \supseteq I(a;\varphi)$. Thus, the condition (5.1) in Lemma 5.2 can be rewritten as follows.

LEMMA 5.3. *Let $\varphi = \bigvee_{v \in T} t_v$ be a canonical Horn DNF of a pdBf (T, F). Then φ is not minimal if and only if at least one of the following CNFs is satisfiable:*

$$\Phi_v = t_v \wedge \bigwedge_{a \in T} C_a, \quad v \in T, \tag{5.2}$$

where

$$C_a = \left(\bigvee_{j \in ON(a)} \bar{x}_j \vee \bigvee_{j \in I(a;\varphi)} x_j \right). \tag{5.3}$$

That is, there is a vector $u \in \{0,1\}^n$ such that $\Phi_v(u) = 1$ for some $v \in T$.

Proof. By the above discussion, φ is not minimal if and only if the formula

$$\varphi(x) \wedge \bigwedge_{a \in T} \left(\bigvee_{j \in ON(a)} \bar{x}_j \vee \bigvee_{j \in I(a;\varphi)} x_j \right)$$

is satisfiable, which is equivalent to (5.2). □

Example 5.2. Consider the pdBf (T, F) of Example 3.1. Recall that we have the following canonical DNF:

$$\varphi = \varphi^{(1)} = 12347\bar{8} \vee 12357\bar{9} \vee 1236\bar{7}8 \vee \bar{1}37 \vee 1\bar{3}7 \vee 23\bar{7}9 \vee 12\bar{7}9 \vee 1234\bar{5}67.$$

Using $I(v^{(l)}; \varphi^{(1)})$, $l = 1, 2, \ldots, 8$, listed in Example 5.1, (5.2) can be written as

$$\begin{aligned}
\Phi_{v^{(1)}} &= 12347\bar{8}(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{7} \vee 8)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{5} \vee \bar{7} \vee 9)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{6} \vee 7 \vee \bar{8})(1 \vee \bar{3} \vee \bar{7}) \\
&\quad (\bar{1} \vee 3 \vee \bar{7})(1 \vee \bar{2} \vee \bar{3} \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee 3 \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{5} \vee \bar{6} \vee 7 \vee 8 \vee 9), \\
\Phi_{v^{(2)}} &= 12357\bar{9}(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{7} \vee 8)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{5} \vee \bar{7} \vee 9)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{6} \vee 7 \vee \bar{8})(1 \vee \bar{3} \vee \bar{7}) \\
&\quad (\bar{1} \vee 3 \vee \bar{7})(1 \vee \bar{2} \vee \bar{3} \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee 3 \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{5} \vee \bar{6} \vee 7 \vee 8 \vee 9), \\
\Phi_{v^{(3)}} &= 1236\bar{7}8(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{7} \vee 8)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{5} \vee \bar{7} \vee 9)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{6} \vee 7 \vee \bar{8})(1 \vee \bar{3} \vee \bar{7}) \\
&\quad (\bar{1} \vee 3 \vee \bar{7})(1 \vee \bar{2} \vee \bar{3} \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee 3 \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{5} \vee \bar{6} \vee 7 \vee 8 \vee 9), \\
\Phi_{v^{(4)}} &= \bar{1}37(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{7} \vee 8)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{5} \vee \bar{7} \vee 9)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{6} \vee 7 \vee \bar{8})(1 \vee \bar{3} \vee \bar{7}) \\
&\quad (\bar{1} \vee 3 \vee \bar{7})(1 \vee \bar{2} \vee \bar{3} \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee 3 \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{5} \vee \bar{6} \vee 7 \vee 8 \vee 9), \\
\Phi_{v^{(5)}} &= 1\bar{3}7(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{7} \vee 8)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{5} \vee \bar{7} \vee 9)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{6} \vee 7 \vee \bar{8})(1 \vee \bar{3} \vee \bar{7}) \\
&\quad (\bar{1} \vee 3 \vee \bar{7})(1 \vee \bar{2} \vee \bar{3} \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee 3 \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{5} \vee \bar{6} \vee 7 \vee 8 \vee 9), \\
\Phi_{v^{(6)}} &= 23\bar{7}9(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{7} \vee 8)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{5} \vee \bar{7} \vee 9)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{6} \vee 7 \vee \bar{8})(1 \vee \bar{3} \vee \bar{7}) \\
&\quad (\bar{1} \vee 3 \vee \bar{7})(1 \vee \bar{2} \vee \bar{3} \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee 3 \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{5} \vee \bar{6} \vee 7 \vee 8 \vee 9), \\
\Phi_{v^{(7)}} &= 12\bar{7}9(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{7} \vee 8)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{5} \vee \bar{7} \vee 9)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{6} \vee 7 \vee \bar{8})(1 \vee \bar{3} \vee \bar{7}) \\
&\quad (\bar{1} \vee 3 \vee \bar{7})(1 \vee \bar{2} \vee \bar{3} \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee 3 \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{5} \vee \bar{6} \vee 7 \vee 8 \vee 9), \\
\Phi_{v^{(8)}} &= 1234\bar{5}67(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{7} \vee 8)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{5} \vee \bar{7} \vee 9)(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{6} \vee 7 \vee \bar{8})(1 \vee \bar{3} \vee \bar{7}) \\
&\quad (\bar{1} \vee 3 \vee \bar{7})(1 \vee \bar{2} \vee \bar{3} \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee 3 \vee 7 \vee \bar{9})(\bar{1} \vee \bar{2} \vee \bar{3} \vee \bar{4} \vee \bar{5} \vee \bar{6} \vee 7 \vee 8 \vee 9).
\end{aligned}$$

Now take a vector $u = (111000001)$. This u satisfies $\Phi_{v^{(6)}}(u) = 1$, showing that f_φ is not a minimal Horn extension of (T, F).

Note that CNFs Φ_v of (5.2) are not Horn, in general, and therefore their SATs in Lemma 5.3 may not be easy. However, we prove in the rest of this subsection that this can be done in polynomial time.

Let $\varphi = \bigvee_{v \in T} t_v$ be a canonical Horn DNF of (T, F). Given a $v \in T$, define

$$\hat{I}(v; \varphi) = I(v; \varphi) \setminus \{l_v\}, \tag{5.4}$$

where $\bar{x}_{l_v}$ is the negative literal in t_v. Now assume that Φ_v defined by (5.2) is satisfiable, i.e., there is a vector $u \in \{0,1\}^n$ such that $\Phi_v(u) = 1$. This means $t_v(u) = 1$, and hence

$$ON(u) \supseteq ON(v) \text{ and } l_v \in OFF(u). \tag{5.5}$$

Therefore, we can fix $u_j = 1$ for all $j \in ON(v)$ and $u_{l_v} = 0$. We also have

$$ON(u) \cap \hat{I}(v; \varphi) \neq \emptyset \tag{5.6}$$

in order to satisfy the clause $(\bigvee_{j\in ON(v)} \bar{x}_j \vee \bigvee_{j\in I(v;\varphi)} x_j)$ associated with $a = v$. Furthermore, we shall show below that we can fix

$$u_j = 0 \quad \text{for all } j \in OFF(v) \setminus I(v;\varphi) \tag{5.7}$$

without loss of generality. As a result of these observations, we denote by

$$\Phi'_v = \bigwedge_{a\in T} C'_a \tag{5.8}$$

the CNF obtained from Φ_v by fixing variables x_j to 1 for $j \in ON(v)$ and 0 for $j \in OFF(v) \setminus \hat{I}(v;\varphi)$, in which C'_a denotes the clause obtained from C_a in the same way. Then Φ_v is satisfiable if and only if Φ'_v is satisfiable. Note that only those x_j satisfying $j \in \hat{I}(v;\varphi)$ remain as variables in Φ'_v. In Example 5.2, it can be seen that $\Phi'_{v^{(i)}} = \bot$ for $i = 1,2,3,4,5$, $\Phi'_{v^{(6)}} = 1$, $\Phi'_{v^{(7)}} = 3$, and $\Phi'_{v^{(8)}} = \bar{8}(8 \vee 9)$. This may indicate that Φ'_v is much simpler than Φ_v to consider.

Now we prove the above claim (5.7) after showing the next lemma.

LEMMA 5.4. *Let $\varphi = \bigvee_{v\in T} t_v$ be a canonical Horn DNF of a pdBf (T,F). Let $\Phi_v = t_v \wedge \bigwedge_{a\in T} C_a$ and $\Phi'_v = \bigwedge_{a\in T} C'_a$ be defined as above for a $v \in T$. Then $C'_a \neq \top$ holds for an $a \in T$ if and only if the following two conditions hold:*

(i) $ON(a) \subseteq ON(v) \cup \hat{I}(v;\varphi)$,

(ii) $I(a;\varphi) \subseteq I(v;\varphi)$.

Proof. It is easy to see that if a satisfies (i) and (ii), then $C'_a \neq \top$ holds because C'_a is obtained from $C_a = (\bigvee_{j\in ON(a)} \bar{x}_j \vee \bigvee_{j\in I(a;\varphi)} x_j)$ by fixing $x_j = 1$ for $j \in ON(v)$ and 0 for $j \in OFF(v) \setminus \hat{I}(v;\varphi)$.

On the other hand, let us assume that $C'_a \neq \top$ holds. If there exists an $l \in ON(a) \setminus (ON(v) \cup \hat{I}(v;\varphi))\,(= ON(a) \cap (OFF(v) \setminus \hat{I}(v;\varphi)))$, then $C'_a = \top$ holds, because the x_l is fixed to 0, which is a contradiction to the assumption. This proves property (i).

Next, to prove (ii), assume that there exists an index $l \in I(a;\varphi) \setminus I(v;\varphi)$. The following two cases are possible:

(a) $l \in ON(v) \cap I(a;\varphi)$. Then, x_l is fixed to 1 and $C'_a = \top$ holds, which is a contradiction.

(b) $l \in OFF(v) \cap I(a;\varphi)$. Clearly, $l \in (OFF(v) \setminus I(v;\varphi)) \cap I(a;\varphi)$. Then $l \in I(a;\varphi)$ implies $(\bigwedge_{j\in ON(a)} x_j)\,\bar{x}_l \leq f_\varphi$, and therefore, by property (i),

$$\left(\bigwedge_{j\in ON(v)\cup I(v;\varphi)} x_j\right) \bar{x}_l \leq f_\varphi. \tag{5.9}$$

Now, $l \in OFF(v) \setminus I(v;\varphi)$ implies that $(\bigwedge_{j\in ON(v)} x_j)\,\bar{x}_l \not\leq f_\varphi$. However, since $(\bigwedge_{j\in ON(v)} x_j)\,\bar{x}_h \leq f_\varphi$ for all $h \in I(v;\varphi)$ and

$$\begin{aligned} &T\left(\left(\bigwedge_{j\in ON(v)\cup I(v;\varphi)} x_j\right)\bar{x}_l\right) \\ &\quad = T\left(\left(\bigwedge_{j\in ON(v)} x_j\right)\bar{x}_l\right) \setminus T\left(\left(\bigwedge_{j\in ON(v)} x_j\right)\left(\bigvee_{h\in I(v;\varphi)} \bar{x}_h\right)\bar{x}_l\right), \end{aligned}$$

we have $(\bigwedge_{j\in ON(v)\cup I(v;\varphi)} x_j)\,\bar{x}_l \not\leq f_\varphi$, which is a contradiction to (5.9). ∎

Now, we prove our claim.

LEMMA 5.5. *Let $\varphi = \bigvee_{v\in T} t_v$ be a canonical Horn DNF of a pdBf (T, F). If a vector u satisfies $\Phi_v(u) = 1$ for a vector $v \in T$, then the vector u' also satisfies $\Phi_v(u') = 1$, where u' is defined by*

$$u'_j = \begin{cases} u_j & \text{if } j \in ON(v) \cup I(v;\varphi), \\ 0 & \text{otherwise.} \end{cases} \tag{5.10}$$

Proof. Let $\Phi'_v = \bigwedge_{a\in T} C'_a$ and consider an $a \in T$ such that $C'_a \neq \top$. By $\Phi_v(u) = 1$, $C_a(u) = (\bigvee_{j\in ON(a)} \bar{u}_j \vee \bigvee_{j\in I(a;\varphi)} u_j) = 1$ holds. However, considering the condition $ON(a) \cup I(a;\varphi) \subseteq ON(v) \cup I(v;\varphi)$ (which follows from Lemma 5.4), we have $C_a(u') = C_a(u) = 1$. Furthermore, $C_b(u') = 1$ holds for all other clauses C_b with $C'_b = \top$, and also $t_v(u') = 1$ holds. These prove $\Phi_v(u') = 1$. □

Now, we summarize the above result as the following lemma.

LEMMA 5.6. *Let $\varphi = \bigvee_{v\in T} t_v$ be a canonical Horn DNF of a pdBf (T, F). Then φ is not minimal if and only if at least one of the CNFs Φ'_v, $v \in T$, is satisfiable, where Φ'_v is defined by* (5.8).

In order to find a vector u such that $\Phi'_v(u) = 1$, we can remove from Φ'_v all the clauses $C'_a = \top$. Furthermore, by (5.6), if a vector u satisfies $C'_v(u) = 1$, then all other clauses C'_a such that $I(a;\varphi) = I(v;\varphi)$ satisfy $C'_a(u) = 1$. Therefore, we can also remove from Φ'_v all the clauses C'_a satisfying $a \neq v$ and $I(a;\varphi) = I(v;\varphi)$. In the following, we write the resulting CNF also as Φ'_v. In other words, denoting by T_v the set of vectors $a \in T$ such that $C'_a \neq \top$ and $I(a;\varphi) \subset I(v;\varphi)$, where $\subset$ denotes proper inclusion, Φ'_v can be written as

$$\Phi'_v \;=\; C'_v \wedge \bigwedge_{a\in T_v} C'_a. \tag{5.11}$$

Note that $v \notin T_v$ holds by definition, and that a vector u satisfies $C'_v(u) = 1$ if and only if $ON(u) \cap \hat{I}(v;\varphi) \neq \emptyset$ holds (since $C'_v = \bigvee_{j\in\hat{I}(v;\varphi)} x_j$). Since C'_v is a special clause in Φ'_v (see the subsequent discussion), we check the conditions $C'_v(u) = 1$ and $\Phi''_v(u) = 1$ separately, where $\Phi''_v = \bigwedge_{a\in T_v} C'_a$. We emphasize here that these CNFs Φ''_v (and hence Φ'_v) may still be non-Horn. However, the following lemma shows that they can be transformed into Horn CNFs

$$\Phi^*_v \;=\; \bigwedge_{a\in T_v} C^*_a, \tag{5.12}$$

where C^*_a denotes the clause obtained from C'_a by removing all literals x_j, $j \in \hat{I}(a;\varphi)$. For example, if $C'_a = (\bar{1} \vee \bar{2} \vee 3 \vee 4 \vee 5)$ and $\hat{I}(a;\varphi) = \{4, 5, 6\}$, then $C^*_a = (\bar{1} \vee \bar{2} \vee 3)$ holds; in this case $\bar{x}_3$ is the negative literal in t_a. We can easily see that Φ^*_v is in fact Horn, because each clause C^*_a has at most one positive literal x_{l_a}, which appears negated in t_a.

LEMMA 5.7. *Let $\varphi = \bigvee_{v\in T} t_v$ be a canonical Horn DNF of a pdBf (T, F). Then φ is not minimal if and only if at least one of the Horn CNFs Φ^*_v, $v \in T$, has a vector $u \in \{0, 1\}^n$ such that $\Phi^*_v(u) = 1$ and $ON(u) \cap \hat{I}(v;\varphi) \neq \emptyset$.*

Proof. Let us first assume that some Φ^*_v has a vector u such that $\Phi^*_v(u) = 1$ and $ON(u) \cap \hat{I}(v;\varphi) \neq \emptyset$. Since $\Phi^*_v(u) = 1$ and $ON(u) \cap \hat{I}(v;\varphi) \neq \emptyset$, resp., imply $C'_a(u) = 1$ for all $a \in T_v$, and $C'_v(u) = 1$ (note that $C'_v = \bigvee_{j\in\hat{I}(v;\varphi)} x_j$), we have $\Phi'_v(u) = 1$. Thus Lemma 5.6 shows the if-part.

To prove the only-if part, let us assume by Lemma 5.6 that Φ'_v has a vector u such that $\Phi'_v(u) = 1$ and has the minimum $|I(v;\varphi)|$; i.e., no Φ'_w satisfies $|I(w;\varphi)| < |I(v;\varphi)|$. By (5.6), this u must satisfy $ON(u) \cap \hat{I}(v;\varphi) \neq \emptyset$. To show $\Phi^*_v(u) = 1$, let us assume the contrary, i.e., $C^*_b(u) = 0$ holds for some $b \in T_v$. Without loss of generality, we assume that the condition

$$u_j = 1 \text{ for all } j \in ON(v) \text{ and } 0 \text{ for all } j \in OFF(v) \setminus \hat{I}(v;\varphi) \tag{5.13}$$

holds, since Φ'_v consists of only those variables x_j satisfying $j \in \hat{I}(v;\varphi)$. Then $C^*_b(u) = 0$ implies $\overline{t_b}(u) = (\bigvee_{j \in ON(b)} \bar{u}_j \vee u_{l_b}) = 0$, where $\bar{x}_{l_b}$ is the negative literal in t_b. This is because C^*_b is obtained from the clause $(\bigvee_{j \in ON(b)} \bar{x}_j \vee x_{l_b})$ by fixing $x_j = 1$ for $j \in ON(v)$ and 0 for $j \in OFF(v) \setminus \hat{I}(v;\varphi)$ (which u also satisfies by (5.13)). Thus we have $t_b(u) = 1$. Since $\Phi_v = t_v \wedge \bigwedge_{a \in T} C_a$ and $\Phi_b = t_b \wedge \bigwedge_{a \in T} C_a$, $t_b(u) = 1$ and $\Phi_v(u) = 1$ imply $\Phi_b(u) = 1$. However, by Lemmas 5.3 and 5.6, this means that Φ'_b is satisfiable, which is a contradiction to our assumption that $|I(v;\varphi)|$ is the minimum (since $I(b;\varphi) \subset I(v;\varphi)$ holds by (ii) of Lemma 5.4 and the discussion following Lemma 5.6). □

Based on Lemma 5.7, we can propose an algorithm to solve problem MINIMAL-H-EXTENSION.

Algorithm CHECK-MINIMAL

Input: A pdBf (T, F) and a Horn DNF φ.

Question: Is φ a minimal Horn DNF of (T, F)?

Step 1: Check if f_φ is a Horn extension of (T, F). If not, output "no" and halt.

Step 2: Construct a canonical Horn DNF $\psi = \bigvee_{v \in T} t_v$ such that $f_\psi \leq f_\varphi$. If $f_\psi < f_\varphi$, then output "no" and halt; otherwise (i.e., $f_\psi = f_\varphi$), rewrite ψ as φ.

Step 3: For each $v \in T$, check if Φ^*_v has a vector u such that $ON(u) \cap \hat{I}(v;\varphi) \neq \emptyset$ and $\Phi^*_v(u) = 1$. Output "no" if some Φ^*_v has such a vector; otherwise, "yes." Halt.

THEOREM 5.1. *Given a pdBf (T, F) and a Horn DNF φ, Problem MINIMAL-H-EXTENSION can be solved in $O(|F||\varphi|+n|T||\varphi|+n|T|^2)$ time by Algorithm CHECK-MINIMAL, where $T, F \subseteq \{0,1\}^n$ and $|\varphi|$ denotes the number of literals in φ.*

Proof. The correctness of Algorithm CHECK-MINIMAL follows from Lemma 5.7. We therefore consider its time complexity. Step 1 can be executed in $O((|T|+|F|)|\varphi|)$ time, since we can check if $\varphi(v) = 1$ or 0 for each $v \in T \cup F$ in $O(|\varphi|)$ time. In Step 2, we first compute $I(v;\varphi)$ for all $v \in T$, which can be done in $O(|T||\varphi|)$ time by applying Algorithm F-CHAINING in subsection 5.1 to all $v \in T$. Choose an arbitrary term $t_v \in R(v;\varphi)$ for each $v \in T$, and then construct a canonical Horn DNF $\psi = \bigvee_{v \in T} t_v$, which satisfies $f_\psi \leq f_\varphi$ and $|\psi| \leq n|T|$. This can be done in $O(n|T|)$ time. Checking if two Horn DNFs ψ and φ satisfy $f_\psi < f_\varphi$ can then be done in $O(|\psi||\varphi|) = O(n|T||\varphi|)$ time [14]. Totally, Step 2 requires $O(n|T||\varphi|)$ time. In Step 3, for each $v \in T$, we can construct Φ^*_v in $O(n|T|)$ time, because $I(v;\varphi)$ was already obtained in Step 2.

Let us now consider how to check if there is a vector u such that $ON(u) \cap \hat{I}(v;\varphi) \neq \emptyset$ and $\Phi^*_v(u) = 1$. Since the variable set of Φ^*_v is $\hat{I}(v;\varphi)$, we regard Φ^*_v as a CNF of $|\hat{I}(v;\varphi)|$ variables in this proof. For notational convenience, given a $u \in \{0,1\}^n$, we write $\Phi^*_v(u|_{\hat{I}(v;\varphi)}) = 1$ instead of $\Phi^*_v(u) = 1$, where $u|_S$ is the projection of u to $S \subseteq \{1, 2, \ldots, n\}$. To check if there is a vector $u|_{\hat{I}(v;\varphi)} \neq (0, 0, \ldots, 0)$ such that $\Phi^*_v(u|_{\hat{I}(v;\varphi)}) = 1$, we consider two cases, $\Phi^*_v(0, 0, \ldots, 0) = 0$ and $\Phi^*_v(0, 0, \ldots, 0) = 1$. If $\Phi^*_v(0, 0, \ldots, 0) = 0$, then solve H-SAT for Φ^*_v (see section 2); if the output of H-SAT is "yes" (resp., "no"), there is a desired vector u (resp., no desired vector u).

On the other hand, if $\Phi_v^*(0,0,\ldots,0)=1$, then solve UNIQUE-H-SAT (see section 2) to see if $(0,0,\ldots,0)$ is the unique vector such that $\Phi_v^*(0,0,\ldots,0)=1$. There is no desired vector u if and only if $(0,0,\ldots,0)$ is the unique such vector. Since H-SAT and UNIQUE-H-SAT can be solved in time linear in the number of literals [7, 22], this can be done in $O(|\varphi|)=O(n|T|)$ time for each $v\in T$. Therefore, Step 3 can be executed in $O(n|T|)\times|T|=O(n|T|^2)$ time.

Summing up the time of all steps, we conclude that Algorithm CHECK-MINIMAL requires $O(|F||\varphi|+n|T||\varphi|+n|T|^2)$ time. □

If a given DNF φ is already a canonical Horn DNF, we can skip Steps 1 and 2 of CHECK-MINIMAL, leading to the following corollary.

COROLLARY 5.1. *Given a pdBf (T,F) and a canonical Horn DNF φ of (T,F), Problem MINIMAL-H-EXTENSION can be solved in $O(n|T|^2)$ time.*

Example 5.3. Consider the pdBf (T,F) and canonical Horn DNF $\varphi=\varphi^{(1)}$ of Example 3.1, to which we apply Algorithm CHECK-MINIMAL. Sets $I(v^{(l)};\varphi)$, $l=1,2,\ldots,|T|$, are listed in Example 5.1. It can be seen that $\Phi^*_{v^{(i)}}=\bot$ for $i=1,2,3,4,5$, $\Phi^*_{v^{(6)}}=\Phi^*_{v^{(7)}}=\top$, and $\Phi^*_{v^{(8)}}=\bar{8}$. Clearly, $\Phi^*_{v^{(i)}}$, $i=6,7,8$, has a vector u satisfying the condition in Step 3 of CHECK-MINIMAL. Consequently, this φ does not represent a minimal Horn extension of (T,F).

5.2. Generating a minimal Horn extension. In this subsection, we consider the generation of a minimal canonical Horn DNF of a given pdBf (T,F). To generate a minimal canonical Horn DNF of a given pdBf (T,F), we first construct a canonical Horn DNF φ and then recursively check if some Φ_v^*, $v\in T$, has a vector $u\in\{0,1\}^n$ satisfying $ON(u)\cap\hat{I}(v;\varphi)\neq\emptyset$, $\Phi_v^*(u)=1$, and (5.13). If no, output φ and halt. Otherwise, update φ to a canonical Horn DNF φ' such that $\varphi'(u)=0$ and $\varphi'<\varphi$.

Note that condition (5.13) for u is not restrictive, because Φ_v^* consists of only variables x_j satisfying $j\in\hat{I}(v;\varphi)$. Furthermore, by this restriction, $\Phi_v^*(u)=1$ implies $\Phi_v(u)=1$, and hence we can construct the above φ'. Formally, it can be written as follows.

Algorithm FIND-MINIMAL

Input: A pdBf (T,F).

Output: A minimal canonical Horn DNF φ of (T,F) if (T,F) has a Horn extension; otherwise, "no."

Step 1: If (T,F) has a Horn extension, construct a canonical DNF $\varphi=\bigvee_{v\in T}t_v$; otherwise, output "no" and halt.

Step 2: For each $v\in T$, check if Φ_v^* has a vector $u\in\{0,1\}^n$ satisfying $ON(u)\cap\hat{I}(v;\varphi)\neq\emptyset$, $\Phi_v^*(u)=1$, and (5.13). If no Φ_v^* has such a vector u, then output the current φ and halt. On the other hand, if Φ_v^* has such a vector u, based on this u, define

$$R_u(a;\varphi)=\{t\in R(a;\varphi)\,|\,t(u)=0\},\quad a\in T, \tag{5.14}$$

and reconstruct a canonical Horn DNF φ by

$$\varphi:=\bigvee_{a\in T}t_a,\quad t_a\in R_u(a;\varphi), \tag{5.15}$$

where $t_a\in R_u(a;\varphi)$ is chosen arbitrarily if $|R_u(a;\varphi)|\geq 2$. Return to Step 2.

In Step 2, if we have a desired vector u, $R_u(a;\varphi)$ of (5.14) can be easily obtained as follows:

$$R_u(a;\varphi)=\begin{cases} R(a;\varphi) & \text{if } ON(a)\cap OFF(u)\neq\emptyset,\\ \{(\bigwedge_{j\in ON(a)}x_j)\,\bar{x}_l\,|\,l\in ON(u)\cap I(a;\varphi)\} & \text{otherwise.}\end{cases}$$

THEOREM 5.2. *Given a pdBf (T, F), where $T, F \subseteq \{0,1\}^n$, a minimal canonical Horn DNF φ of (T, F) can be generated in $O(n|T|(|F| + n|T|^2))$ time if (T, F) has a Horn extension.*

Proof. FIND-MINIMAL is similar to CHECK-MINIMAL of subsection 5.1. To show its correctness, we need only prove that (i) $R_u(a;\varphi) \neq \emptyset$ holds for all $a \in T$ in (5.14) and (ii) FIND-MINIMAL will eventually halt.

(i) By the definition of Φ_v^* and the assumption (5.13) on u, $\Phi_v^*(u) = 1$ implies $\Phi_v(u) = 1$. This means that $C_a(u) = 1$ holds for all $a \in T$, where $C_a = (\bigvee_{j \in ON(a)} \bar{x}_j \vee \bigvee_{j \in I(a;\varphi)} x_j)$, and hence some $t \in R(a;\varphi)$ satisfies $t(u) = 0$, by the definition of $R_u(a;\varphi)$. Therefore, $R_u(a;\varphi) \neq \emptyset$ holds for all $a \in T$.

(ii) Let φ' be the DNF constructed by (5.15) from φ during an iteration. Then φ' is clearly a canonical Horn DNF of (T, F), and $f_{\varphi'} < f_\varphi$ holds since $u \in T(\varphi) \setminus T(\varphi')$. More precisely,

$$\sum_{a \in T} |R(a;\varphi')| \ < \ \sum_{a \in T} |R(a;\varphi)| \ \ \big(\leq \ n|T|\big)$$

holds. This is because $R(a;\varphi') \subseteq R(a;\varphi)$ holds for all $a \in T$, and t_v in φ is included in $R(v;\varphi)$ but not in $R(v;\varphi')$ (since $\Phi_v(u) = 1$ implies $t_v(u) = 1$). Thus the number of iterations is at most $n|T|$, which proves (ii).

Finally, let us consider its time complexity. By Theorem 3.1, Step 1 can be done in $O(n|T||F|)$ time. In Step 2, for each $v \in T$, we can obtain Φ_v^* from φ in $O(|\varphi|) = O(n|T|)$ time, since $I(v;\varphi)$ can be computed in $O(|\varphi|) = O(n|T|)$ time by applying F-CHAINING of subsection 5.1 to a vector v. Similar to the proof of Theorem 5.1, for each $v \in T$, a vector u satisfying $ON(u) \cap \hat{I}(v;\varphi) \neq \emptyset$, $\Phi_v^*(u) = 1$, and (5.13) can be computed in $O(n|T|)$ time, if there is at least one such u. Thus $O(n|T|) \times |T| = O(n|T|^2)$ time is required to find the desired vector u. Once we have such a vector u, $t_a \in R_u(a;\varphi)$ can be obtained in $O(n)$ time for each $a \in T$. (See the discussion below Algorithm FIND-MINIMAL.) This means that the new φ of (5.15) can be obtained in $O(n|T|)$ time. Hence each iteration of Step 2 requires $O(n|T|^2)$ time. Since the number of iterations is at most $n|T|$, Step 2 requires $O(n^2|T|^3)$ time in total.

Consequently, Algorithm FIND-MINIMAL can be executed in $O(n|T|(|F|+n|T|^2))$ time. □

Example 5.4. Consider again the pdBf (T, F) of Example 3.1. We demonstrate Algorithm FIND-MINIMAL by assuming that a canonical DNF $\varphi = \varphi^{(1)}$ of Example 3.1 is obtained in Step 1.

First iteration. Sets $I(v^{(l)};\varphi)$, $l = 1, 2, \ldots, 8$, are listed in Example 5.1. We have $\Phi^*_{v^{(i)}} = \bot$ for $i = 1, 2, \ldots, 5$, $\Phi^*_{v^{(6)}} = \Phi^*_{v^{(7)}} = \top$, and $\Phi^*_{v^{(8)}} = \bar{8}$, which are obtained in Example 5.3. For example, $\Phi^*_{v^{(6)}}$ has a vector $u = (111000001)$ satisfying the condition in Step 2 of FIND-MINIMAL. Based on this u, we construct $R_u(v^{(l)};\varphi)$ of (5.14): $R_u(v^{(1)};\varphi) = \{12347\bar{8}\}$, $R_u(v^{(2)};\varphi) = \{12357\bar{9}\}$, $R_u(v^{(3)};\varphi) = \{1236\bar{7}8\}$, $R_u(v^{(4)};\varphi) = \{\bar{1}37\}$, $R_u(v^{(5)};\varphi) = \{1\bar{3}7\}$, $R_u(v^{(6)};\varphi) = \{\bar{1}239\}$, $R_u(v^{(7)};\varphi) = \{12\bar{3}9\}$, and $R_u(v^{(8)};\varphi) = \{1234567, 1234568, 1234569\}$. By (5.15), we obtain a canonical Horn DNF

$$\varphi \ := \ 12347\bar{8} \vee 12357\bar{9} \vee 1236\bar{7}8 \vee \bar{1}37 \vee 1\bar{3}7 \vee \bar{1}239 \vee 12\bar{3}9 \vee 1234567,$$

if 1234567 is chosen from $R_u(v^{(8)};\varphi)$.

Second iteration. Construct $I(v^{(1)};\varphi) = \{8\}$, $I(v^{(2)};\varphi) = \{9\}$, $I(v^{(3)};\varphi) = \{7\}$, $I(v^{(4)};\varphi) = \{1\}$, $I(v^{(5)};\varphi) = \{3\}$, $I(v^{(6)};\varphi) = \{1\}$, $I(v^{(7)};\varphi) = \{3\}$, and $I(v^{(8)};\varphi) = \{7,8,9\}$. Then we have $\Phi^*_{v^{(i)}} = \bot$ for $i = 1,2,\ldots,7$ and $\Phi^*_{v^{(8)}} = \bar{8}$. $\Phi^*_{v^{(8)}}$ has a vector $u = (111111001)$ satisfying the condition in Step 2 of FIND-MINIMAL. By (5.14), we construct $R_u(v^{(1)};\varphi) = \{12347\bar{8}\}$, $R_u(v^{(2)};\varphi) = \{12357\bar{9}\}$, $R_u(v^{(3)};\varphi) = \{1236\bar{7}8\}$, $R_u(v^{(4)};\varphi) = \{\bar{1}37\}$, $R_u(v^{(5)};\varphi) = \{1\bar{3}7\}$, $R_u(v^{(6)};\varphi) = \{\bar{1}239\}$, $R_u(v^{(7)};\varphi) = \{12\bar{3}9\}$, and $R_u(v^{(8)};\varphi) = \{123456\bar{9}\}$. The next canonical Horn DNF φ constructed by (5.15) is

$$\varphi \quad := \quad 12347\bar{8} \vee 12357\bar{9} \vee 1236\bar{7}8 \vee \bar{1}37 \vee 1\bar{3}7 \vee \bar{1}239 \vee 12\bar{3}9 \vee 123456\bar{9}.$$

Third iteration. Construct $I(v^{(1)};\varphi) = \{8\}$, $I(v^{(2)};\varphi) = \{9\}$, $I(v^{(3)};\varphi) = \{7\}$, $I(v^{(4)};\varphi) = \{1\}$, $I(v^{(5)};\varphi) = \{3\}$, $I(v^{(6)};\varphi) = \{1\}$, $I(v^{(7)};\varphi) = \{3\}$, and $I(v^{(8)};\varphi) = \{9\}$. We have $\Phi^*_{v^{(i)}} = \bot$ for $i = 1,2,\ldots,8$. Therefore, the above φ obtained in the second iteration is a minimal Horn DNF of (T,F). This φ is equal to $\varphi^{(2)}$ of Example 3.1.

5.3. Minimum and unique minimal Horn extensions. In this subsection, we first consider the problem of computing a minimum (i.e., with the smallest $|T(f)|$) Horn extension f among all minimal Horn extensions and then the condition for the uniqueness of a minimal Horn extension.

Problem MINIMUM-H-EXTENSION

Input: A pdBf (T,F).

Output: A minimum Horn DNF φ of (T,F) if (T,F) has a Horn extension; otherwise, "no."

THEOREM 5.3. *Problem MINIMUM-H-EXTENSION is NP-hard, even if* $F = \emptyset$.

Proof. We transform Problem VERTEX COVER to this problem, where VERTEX COVER is known to be NP-hard [11]. Let $G = (V,E)$ be an undirected graph, where $V = \{1,\ldots,n\}$. Let us define $T, F \subseteq \{0,1\}^n$ as follows:

$$\begin{array}{lcl} T & = & \{x^A \mid A = V \setminus \{i,j\}, (i,j) \in E\} \text{ and} \\ F & = & \emptyset, \end{array}$$

where x^A denotes the characteristic vector of set $A \subseteq V$ (i.e., $x^A_j = 1$ if $j \in A$ and $x^A_j = 0$ if $j \notin A$). As $F = \emptyset$, this (T,F) obviously has a Horn extension. Let $\varphi = \bigvee_{v \in T} t_v$ with $t_v = (\bigwedge_{j \in P_v} x_j)\bar{x}_{l_v}$ be a canonical Horn DNF that represents a minimum Horn extension of (T,F). We claim that $|T(f_\varphi)| = |E| + \tau(G)$, where $\tau(G)$ denotes the cardinality of a minimum vertex cover of G. This will prove the theorem, since Problem VERTEX COVER (i.e., computing $\tau(G)$) is known to be NP-hard [11] and $|T(f_\varphi)|$ can be computed from such a φ in polynomial time.

To prove the claim, we first show that $|T(f_\varphi)| \geq |E| + \tau(G)$. Since φ is a canonical Horn DNF, we have $P_v = ON(v)$ (where $|ON(v)| = n-2$), $l_v \in I(v)\,(= OFF(v))$, and $T(t_v) = \{v, x^{V \setminus \{l_v\}}\}$. Therefore, $T(f_\varphi) = T \cup \{x^{V \setminus \{l_v\}} \mid v \in T\}$. Since $OFF(v)$ of each $v \in T$ corresponds to an edge of G, set $\{l_v \mid v \in T\}$ forms a vertex cover of G. Therefore,

$$\begin{aligned} |T(\varphi)| &= |T \cup \{x^{V \setminus \{l_v\}} \mid v \in T\}| \\ &= |E| + |\{l_v \mid v \in T\}| \\ &\geq |E| + \tau(G). \end{aligned}$$

Conversely, let $W \subseteq V$ be a minimum vertex cover with $|W| = \tau(G)$. Then define a Horn DNF $\varphi_W = \bigvee_{v \in T} t_v$, where $t_v = (\bigwedge_{j \in ON(v)} x_j)\, \bar{x}_{l_v}$ for some $l_v \in OFF(v) \cap W$. Then $T(t_v) = \{v, x^{V \setminus \{l_v\}}\}$ holds and φ_W is a Horn DNF of (T, F). Furthermore, $|T(\varphi_W)| = |T \cup \{x^{V \setminus \{k\}} \mid k \in W\}| = |E| + |W| = |E| + \tau(G)$. □

However, the uniqueness of a minimal Horn extension can be decided in polynomial time.

Problem UMIN-H-EXTENSION

Input: A pdBf (T, F).

Question: Does (T, F) have the unique minimal Horn extension?

LEMMA 5.8. *Let $\varphi = \bigvee_{v \in T} t_v$ be a minimal canonical Horn DNF of a pdBf (T, F). Then (T, F) does not have the unique minimal Horn extension (which is f_φ) if and only if at least one of the CNFs*

$$\Phi_v^\dagger \;=\; t_v \;\wedge\; \bigwedge_{a \in T} C_a^\dagger, \quad v \in T, \tag{5.16}$$

where $C_a^\dagger = (\bigvee_{j \in ON(a)} \bar{x}_j \vee \bigvee_{j \in I(a)} x_j)$, is satisfiable.

Proof. To show the if-part, let us first assume that a vector u satisfies $\Phi_v^\dagger(u) = 1$. Then obviously $\varphi(u) = 1$ holds, and, for each $a \in T$, there is a term t_a^* in $R(v)$ such that $t_a^*(u) = 0$. By choosing such a term t_a^* for each $a \in T$, we have a canonical Horn DNF $\psi = \bigvee_{a \in T} t_a^*$ of (T, F) such that $\psi(u) = 0$. Therefore, u satisfies $\varphi(u) = 1$ and $\psi(u) = 0$, which proves that (T, F) has at least two minimal Horn extensions.

Conversely, if no $\Phi_v^\dagger$ is satisfiable, then, for every vector u such that $\varphi(u) = 1$, some clause $C_a^\dagger$ satisfies $C_a^\dagger(u) = 0$. By the definition of $I(a)$, this implies that $t_a^*(u) = 1$ holds for all $t_a^* \in R(a)$. Therefore, if a vector u satisfies $\varphi(u) = 1$, then $\psi(u) = 1$ must hold for all canonical Horn DNFs $\psi = \bigvee_{a \in T} t_a^*$, which shows that (T, F) has the unique minimal Horn extension. □

Note that this lemma corresponds to Lemma 5.3, and all other lemmas, theorems, and algorithms in subsection 5.1 are valid, even if Φ_v and $I(v; \varphi)$ are replaced by $\Phi_v^\dagger$ and $I(v)$, respectively. (Recall that $I(v)$ becomes $I(v; \varphi)$ if φ represents $f_{\max}$.) Therefore, in order to solve Problem UMIN-H-EXTENSION, first construct a canonical DNF φ such that f_φ is a minimal Horn extension by Algorithm FIND-MINIMAL in subsection 5.2, and then check the condition in Lemma 5.8 by using Algorithm CHECK-MINIMAL with the above replacement incorporated.

THEOREM 5.4. *Problem UMIN-H-EXTENSION can be solved in $O(n|T|(|F| + n|T|^2))$ time.*

Proof. We consider only its time complexity. A minimal canonical Horn DNF φ can be constructed in $O(n|T|(|F| + n|T|^2))$ time by Theorem 5.2. We also can check the condition in Lemma 5.8 in $O(n|T|^2)$ time (Corollary 5.1) by using the modified CHECK-MINIMAL. □

6. Shortest Horn extensions. Finally, we show that the following problem, related to the knowledge compression for expert systems [13, 14], is intractable:

Problem SHORTEST-H-EXTENSION

Input: A pdBf (T, F) and a positive integer k.

Question: Is there a Horn DNF φ of (T, F) such that $|\varphi| \leq k$?

THEOREM 6.1. *Problem SHORTEST-H-EXTENSION is NP-complete.*

Proof. This problem is in NP, since we can check in polynomial time if a given DNF φ represents a Horn extension of (T, F) and satisfies $|\varphi| \leq k$. Now we transform

Problem VERTEX COVER to this problem, where VERTEX COVER is known to be NP-hard [11]. Let $G = (V, E)$ be an undirected graph, where $V = \{1, 2, \ldots, n\}$. Let us define $T, F \subseteq \{0,1\}^n$ as follows:

$$\begin{aligned} T &= \{x^A \mid A = V \setminus \{i,j\}, (i,j) \in E\}, \\ F &= \{e = (11\ldots1)\}, \end{aligned}$$

where x^A denotes the characteristic vector of set $A \subseteq V$.

We claim that there is a Horn DNF φ of (T, F) such that $|\varphi| \leq k$ if and only if $\tau(G) \leq k$, where $\tau(G)$ denotes the cardinality of a minimum vertex cover of G. Similar to the proof of Theorem 5.3, this will complete the proof.

Let $\varphi = \bigvee_{l \in L} t_l$, where $t_l = \bigwedge_{j \in P_l} x_j \bigwedge_{j \in N_l} \bar{x}_j$, be a Horn DNF of (T, F) such that $|\varphi| \leq k$. Then, for every $l \in L$, the following conditions hold:

(a) $N_l \neq \emptyset$, i.e., $|N_l| = 1$ holds since otherwise $t_l(e) = 1$, which is a contradiction.

(b) $P_l = \emptyset$ holds, since replacing P_l by $\emptyset$ produces a shorter good term t'_l such that $|t'_l| \leq |t_l|$, $t_l(a) = 1$ implies $t'_l(a) = 1$ for all $a \in T$, and $t'_l(e) = 0$.

Furthermore, since $\varphi(a) = 1$ for every $a \in T$, there must exist an $l \in L$ such that $t_l(a) = 1$ (i.e., $N_l \subseteq OFF(a) = \{i, j\}$ for the corresponding edge $(i, j) \in E$). Hence $\cup_{l \in L} N_l$ is a vertex cover of G such that $|\cup_{l \in L} N_l| = |\varphi| \leq k$.

Conversely, if W is a vertex cover with $|W| \leq k$, then $\varphi_W = \bigvee_{j \in W} \bar{x}_j$ is a Horn DNF of (T, F) such that $|\varphi_W| \leq k$. □

7. Conclusion and future research. Because there are in general many Horn extensions for a given pdBf (T, F), in this paper we considered Horn extensions with special properties. In particular, we investigated maximal and minimal Horn extensions and pointed out that the maximal Horn extension is always unique (i.e., maximum) but there are many minimal Horn extensions. The main contribution of this paper is to show that checking if a Horn DNF is minimal and generating a minimal Horn DNF of a pdBf (T, F) both can be done in polynomial time. We can also check in polynomial time if a minimal extension is unique. However, the problems of finding a Horn DNF of a minimum Horn extension and finding a shortest Horn DNF of a pdBf (T, F) are shown to be NP-hard.

A possible topic for future research is the development of an efficient algorithm to generate all minimal Horn extensions of a given pdBf (T, F).

Acknowledgments. The authors greatly appreciate the comments given by two anonymous reviewers, which helped improve the readability of this paper. In particular, one of the reviewers gave us an alternative proof of Lemma 5.7 and suggested a shorter presentation of subsection 5.1 (which was originally written in a form dual to the current version).

REFERENCES

[1] D. Angluin, M. Frazier, and L. Pitt, *Learning conjunctions of Horn clauses*, Mach. Learning, 9 (1992), pp. 147–164.

[2] E. Boros, V. Gurvich, P. L. Hammer, T. Ibaraki, and A. Kogan, *Decompositions of partially defined Boolean functions*, Discrete Appl. Math., 62 (1995), pp. 51–75.

[3] J. C. Bioch and T. Ibaraki, *Complexity of identification and dualization of positive Boolean functions*, Inform. and Comput., 123 (1995), pp. 50–63.

[4] E. Boros, T. Ibaraki, and K. Makino, *Error-free and best-fit extensions of partially defined Boolean functions*, Inform. and Comput., 140 (1998), pp. 254–283.

[5] S. Ceri, G. Gottlob, and L. Tanca, *Logic Programming and Databases*, Springer, Berlin, New York, 1990.

[6] Y. Crama, P. L. Hammer, and T. Ibaraki, *Cause-effect relationships and partially defined Boolean functions*, Ann. Oper. Res., 16 (1988), pp. 299–326.
[7] D. W. Dowling and J. H. Gallier, *Linear-time algorithms for testing the satisfying of propositional Horn formulae*, J. Logic Programming, 3 (1984), pp. 267–284.
[8] R. Dechter and J. Pearl, *Structure identification in relational data*, Artificial Intelligence, 58 (1992), pp. 237–270.
[9] T. Eiter and G. Gottlob, *Identifying the minimal transversals of a hypergraph and related problems*, SIAM J. Comput., 24 (1995), pp. 1278–1304.
[10] M. Fredman and L. Khachiyan, *On the complexity of dualization of monotone disjunctive normal forms*, J. Algorithms, 21 (1996), pp. 618–628.
[11] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, New York, 1979.
[12] M. Golumbic, P. L. Hammer, P. Hansen, and T. Ibaraki, eds., *Horn Logic, Search and Satisfiability*, Ann. Math. Artificial Intelligence, 1 (1990), no. 1–4, Baltzer Science Publishers BV, Amsterdam, 1990.
[13] P. L. Hammer and A. Kogan, *Optimal compression of propositional Horn knowledge bases: Complexity and approximation*, Artificial Intelligence, 64 (1993), pp. 131–145.
[14] P. L. Hammer and A. Kogan, *Horn functions and their DNFs*, Inform. Process. Lett., 44 (1992), pp. 23–29.
[15] A. Horn, *On sentences which are true of direct unions of algebras*, J. Symbolic Logic, 16 (1951), pp. 14–21.
[16] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou, *On generating all maximal independent sets*, Inform. Process. Lett., 27 (1988), pp. 119–123.
[17] H. A. Kautz, M. J. Kearns, and B. Selman, *Horn approximations of empirical data*, Artificial Intelligence, 74 (1995), pp. 129–145.
[18] D. Kavvadias, C. H. Papadimitriou, and M. Sideri, *On Horn envelopes and hypergraph transversals*, in ISAAC'93, Algorithms and Computation, K. W. Ng et al., eds., Lecture Notes in Comput. Sci. 762, Springer, Berlin, 1993, pp. 399–405.
[19] R. Khardon, *Translating between Horn representations and their characteristic models*, J. Artificial Intelligence Res., 3 (1995), pp. 349–372.
[20] K. Makino, K. Yano, and T. Ibaraki, *Positive and Horn decomposability of partially defined Boolean functions*, Discrete Appl. Math., 74 (1997), pp. 251–274.
[21] J. C. C. McKinsey, *The decision problem for some classes of sentences without quantifiers*, J. Symbolic Logic, 8 (1943), pp. 61–76.
[22] D. Pretolani, *A linear time algorithm for unique Horn satisfiability*, Inform. Process. Lett., 48 (1993), pp. 61–66.
[23] W. Quine, *A way to simplify truth functions*, Amer. Math. Monthly, 62 (1955), pp. 627–631.
[24] J. R. Quinlan, *Induction of decision trees*, Mach. Learning, 1 (1986), pp. 81–106.
[25] B. Selman and H. Kautz, *Knowledge compilation using Horn approximations*, in Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), July 14–19, 1991, Anaheim, CA, Amer. Assoc. Artif. Intell., Menlo Park, CA, 1991, pp. 904–909.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2187–2214

FAST APPROXIMATE GRAPH PARTITIONING ALGORITHMS*

GUY EVEN†, JOSEPH (SEFFI) NAOR‡, SATISH RAO§, AND BARUCH SCHIEBER¶

Abstract. We study graph partitioning problems on graphs with edge capacities and vertex weights. The problems of b-balanced cuts and k-balanced partitions are unified into a new problem called minimum capacity ρ-separators. A ρ-separator is a subset of edges whose removal partitions the vertex set into connected components such that the sum of the vertex weights in each component is at most ρ times the weight of the graph. We present a new and simple $O(\log n)$-approximation algorithm for minimum capacity ρ-separators which is based on spreading metrics yielding an $O(\log n)$-approximation algorithm both for b-balanced cuts and k-balanced partitions. In particular, this result improves the previous best known approximation factor for k-balanced partitions in undirected graphs by a factor of $O(\log k)$. We enhance these results by presenting a version of the algorithm that obtains an $O(\log \mathrm{OPT})$-approximation factor. The algorithm is based on a technique called spreading metrics that enables us to formulate directly the minimum capacity ρ-separator problem as an integer program. We also introduce a generalization called the simultaneous separator problem, where the goal is to find a minimum capacity subset of edges that separates a given collection of subsets simultaneously. We extend our results to directed graphs for values of $\rho \geq 1/2$. We conclude with an efficient algorithm for computing an optimal spreading metric for ρ-separators. This yields more efficient algorithms for computing b-balanced cuts than were previously known.

Key words. graph partitioning, approximation algorithms, graph separator, spreading metrics

AMS subject classifications. 05C85, 68R10, 68Q20, 68Q25, 68Q35, 90C05, 94C15

PII. S0097539796308217

1. Introduction.

Two well-studied graph partitioning problems are finding minimum capacity *b-balanced cuts* and *k-balanced partitions* in undirected and directed graphs. The input for these problems consists of a graph having edge capacities and also vertex weights. Given a parameter $0 < b \leq 1/2$, the *b-balanced cut problem* is to find a minimum capacity cut such that the weight of the vertex sets on each side of the cut is at least b times the weight of the graph. Given an integer k, the *k-balanced partitioning problem* is to find a minimum capacity subset of edges whose removal partitions the graph into at most k roughly equally weighted subgraphs that are disconnected from each other. (Section 2 formally defines these problems and their vertex counterparts.) These problems have many applications, among them VLSI layout, circuit testing and simulation, parallel scientific processing, and sparse linear system solving. In essence, graph partitioning is used in these applications either for divide-and-conquer algorithms or for facilitating parallelism.

Since these graph partitioning problems are NP-hard, two approaches have been taken. In the first approach, efficient heuristics were designed (e.g., [8]); however,

*Received by the editors August 16, 1996; accepted for publication (in revised form) June 5, 1998; published electronically July 7, 1999.
http://www.siam.org/journals/sicomp/28-6/30821.html

†Department of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel (guy@eng.tau.ac.il).

‡Computer Science Department, Technion, Haifa 32000, Israel (naor@cs.technion.ac.il). The research of this author was supported in part by grant 92-00225 from the United States–Israel Binational Science Foundation, Jerusalem, Israel.

§NEC Research Institute, 4 Independence Way, Princeton, NJ 08540 (satish@research.nj.nec.com).

¶IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (sbar@watson.ibm.com).

these heuristics did not make any guarantee on the quality of the solution. In the second approach, polynomial-time approximation algorithms were designed [13], and an upper bound on the ratio between the value of an approximate solution and an optimal solution is given. The techniques used by these two approaches differ greatly. Lang and Rao [14] conducted some experiments based on [13] and reported that the technique is successful for large graphs if long running times are allowed and if an additional clean-up phase [8] is used. (See also [1].)

We unify the problems of b-balanced cuts and k-balanced partitions into a new problem called *ρ-separators*. Given a parameter $0 < \rho < 1$, the *ρ-separator problem* is to find a minimum capacity cut that partitions the vertex set into connected components such that the weight of each is at most ρ times the weight of the graph. We also introduce a more general problem called *simultaneous separators*: a collection of subsets $U_1, \ldots, U_s$ of the vertex set is given together with separation parameters $\rho_1, \ldots, \rho_s$. The goal is to find a minimum capacity cut that partitions the vertex set into connected components such that each connected component contains at most ρ_i of the weight of subset U_i for all i.

We present a unified framework that allows us to obtain new graph partitioning algorithms. These algorithms help bridge the gap between heuristics and approximation algorithms in the following ways: First, we obtain improved approximation factors. Second, our algorithm is faster by a factor of $O(|V|)$. Third, our main technique may also be used in conjunction with techniques such as cutting planes. Thus, apart from yielding improved algorithms for graph partitioning, our techniques may be used to design new heuristics that differ significantly from the Kernighan and Lin algorithm [8] and simulated annealing [7].

Our framework is based on *spreading metrics* which provide a direct fractional relaxation of graph partitioning problems. Spreading metrics were introduced by Even et al. [3] to obtain improved approximation factors for certain NP-hard graph problems that are amenable to a divide-and-conquer approach. Informally, a spreading metric on a graph is an assignment of lengths to either the edges or the vertices, so that subgraphs on which the optimization problem is nontrivial are spread apart in the associated metric space. In addition, the volume of the spreading metric provides a lower bound on the cost of solving the optimization problem on the input graph. Spreading metrics are used to find a cut in the graph whose cost depends on the volume of the spreading metric in one of the resulting subgraphs. In [3] this cut defines the divide step, and then each subproblem is solved recursively.

1.1. Our results. We believe that the definition of ρ-separators captures the type of partitioning that is actually required in applications. Namely, instead of limiting the number of resulting parts, which is not always important for divide-and-conquer applications or for parallelism, we limit only the sizes or weights of each part. This enables us to unify the problems of b-balanced cuts and k-balanced partitions. We also consider a generalization called simultaneous separators.

Our approximation algorithms are simple and deal easily with the weighted version. The approximation factors for undirected graphs are summarized in Table 1.1, where n denotes the number of vertices in the graph. Note that in order to perform a fair comparison in Table 1.1, we assume that an optimal spreading metric is used. If an α-approximate spreading metric is used, then the approximation factor for the problems we solve is multiplied by α. (A similar issue applies for previous algorithms with regard to solving linear programs.) In particular, the approximation factor of our k-balanced partitioning algorithm improves over previous algorithms by a factor of

TABLE 1.1

Capacities of found solutions when an optimal spreading metric is used where $\text{OPT}_\rho \triangleq$ *cap. of optimal* ρ*-separator,* $\text{OPT}_b \triangleq$ *cap. of optimal* b*-balanced cut,* $\text{OPT}_k \triangleq$ *cap. of optimal* $(k,1)$*-balanced partition, and* $\text{OPT}_{1/2} \triangleq$ *cap. of optimal bisector.*

Problem	Our work	Previous work
ρ'-separator	$\left(\frac{\rho'}{\rho'-\rho}+o(1)\right)\cdot\ln n\cdot\text{OPT}_\rho$	$O(\log n\cdot\log 1/\rho')\cdot\text{OPT}_\rho$ [13, 12, 17]
b'-balanced cut ($b'\le 1/3$)	$\left(\frac{1-b'}{b-b'}+o(1)\right)\cdot\ln n\cdot\text{OPT}_b$	$8\cdot\left(\frac{1}{3(b-b')}+\ln\frac{1-b}{b-b'}\right)\cdot\ln n\cdot\text{OPT}_b$ [13, 6]
k-balanced partition	$(2+o(1))\cdot\ln n\cdot\text{OPT}_k$	$O(\log n\cdot\log k)\cdot\text{OPT}_k$ [12, 17]
separator	$(4+o(1))\cdot\ln n\cdot\text{OPT}_{1/2}$	$\sim 24.8\cdot\ln n\cdot\text{OPT}_{1/2}$ [13, 6]

$O(\log k)$, and our separator approximation improves over previous algorithms roughly by a factor of 6. Note that we compare the cost of the found solution with the cost of an optimal solution to a more restricted problem; this is the reason for the term "pseudoapproximation" used in the literature. For example, we compute a ρ'-separator and compare its cost with the cost of an optimal ρ-separator, for $\rho<\rho'$. The algorithm for approximating ρ-separators holds for directed graphs with constants four times as large if $\rho\ge 1/2$. Since we require for directed graphs that $\rho\ge 1/2$, we are not able to approximate k-balanced partitions in directed graphs for $k>4$, since this requires $\rho'=2/k<1/2$.

Our use of spreading metrics introduces a *direct* fractional relaxation of the partitioning problems. The connection between the fractional relaxation and the original problem is natural and a repeated evaluation of fractional edge lengths is not required, as opposed to the Leighton–Rao algorithm [13].

The running time of our algorithms is dominated by the complexity of finding a spreading metric. Spreading metrics can be computed by general linear programming algorithms (such as the ellipsoid algorithm). However, we present much more efficient approximate algorithms that are based on the framework presented in the papers of Plotkin, Tardos, and Shmoys [16] and Young [18]. These efficient algorithms apply only to undirected graphs. Specifically, the complexity of our deterministic algorithm which computes a constant approximation for a spreading metric is $\tilde{O}(m^2 n\cdot\frac{\rho'}{\rho'-\rho})$.[1] The randomized version has an expected running time of $\tilde{O}(m^2/(\rho'-\rho))$; moreover, this running time is achieved with probability $1-e^{-\Omega(m)}$. For balanced cuts, $(\rho'-\rho)$ is typically a constant. This improves over the implementation of the Leighton–Rao algorithm suggested by Leighton et al. [11], where the time complexity is $\tilde{O}(mn^3)$ with the use of expanders and $\tilde{O}(mn^4)$ without the use of expanders. (Expanders are needed to reduce the number of commodities from $\Omega(n^2)$ to $\Omega(n)$.)

Since our efficient algorithms approximate spreading metrics rather than compute them exactly, applying them increases the approximation factors for ρ-separators. Specifically, in the case of arbitrary vertex weights, the following error terms are

[1] The notation $\tilde{O}$ ignores polylogarithmic factors, and $|V|=n, |E|=m$.

generated by our approximate algorithm for spreading metrics: (a) a multiplicative error bounded by $2(1+\varepsilon)$ introduced in approximating the optimization oracle (see section 7.3.2); (b) a multiplicative error bounded by 2 introduced in the randomized version (see section 7.3.3); (c) an additive error bounded by $2 \cdot \text{OPT}_\rho$ introduced in the scaling procedure (needed for bounding the optimal cost) (see section 7.2); and (d) additional multiplicative error terms as specified in items (a) and (b) (for the randomized version) introduced in the transformation of a solution into a spreading metric (see section 7.4).

Finally, we present $O(\log \text{OPT})$-approximation factors for all these problems, which for small values of OPT is better than the $O(\log n)$-approximation factors.

1.2. Comparison with previous work. The seminal work of Leighton and Rao [13] is the only previous paper that provides a pseudoapproximation algorithm for b-balanced cuts. Given a graph and two parameters $0 < b \leq 1/2$ and $0 < b' \leq \min\{1/3, b\}$ (typically, $b = 1/2$ and $b' = 1/3$), Leighton and Rao presented an algorithm that finds a cut whose capacity is $O(\frac{\log n}{b-b'} \cdot \text{OPT}_b)$, where OPT_b denotes the capacity of an optimum b-balanced cut. Leighton and Rao approach the problem of finding b-balanced cuts indirectly. They define the notion of a sparse cut, whose fractional relaxation is the dual problem of a multicommodity flow problem. They present an $O(\log n)$-approximation algorithm for computing an optimal sparse cut. The Leighton–Rao algorithm proceeds by accumulating approximate optimal sparse cuts, cutting off in each iteration the smaller part of the graph, until a fraction of b' of the weight of the graph is accumulated. Approximating an optimal sparse cut in each iteration is done in two stages. First, a new linear program is solved. Then a good cut is searched for by trying to find a shallow depth subgraph that contains a majority of the vertices. Finding the right expansion ratio is done by trial and error until a right value is observed. Finally, Leighton and Rao show that the union of the $O(\log n)$-approximate sparse cuts results with an $O(\frac{\log n}{b-b'} \cdot \text{OPT}_b)$ capacity b'-balanced cut.

Our algorithm differs from the Leighton–Rao algorithm in several ways. First, we consider a direct linear programming formulation for ρ-separators. We need to solve this linear program only once; the Leighton–Rao algorithm needs to solve a new linear program after each cut is taken. The use of spreading metrics enables us to control the weight of the pieces that are chopped off in undirected graphs so that chopped off pieces need not be further partitioned. In contrast, in the Leighton–Rao algorithm the only guarantee is that the weight of the chopped off part is at most half the weight of the remaining subgraph. Controlling the weight of the chopped off pieces enables us to find k-balanced partitions in undirected graphs without having to recursively partition the chopped off pieces. Our criterion for finding good cuts (e.g., the expansion ratio) is known in advance, and we do not need a trial-and-error stage in order to compute a cut.

Garg, Vazirani, and Yannakakis [6] presented an $O(\log k)$-approximation algorithm for multicuts in undirected networks. They also described how to approximate sparse cuts and obtain an $8 \log n$-approximation. Our results are related to their work in the following ways. We use their credit scheme, and our cut procedure is basically identical to their region growing technique. In fact, although they present their algorithm as a "cut packing" algorithm, one may interpret it as an algorithm for finding "good" cuts in graphs that have constant diameter.

The k-balanced partitioning problem was considered by Leighton, Makedon, and Tragoudas [12] and by Simon and Teng [17]. Their suggested approximation algo-

rithm is based on recursive bisection or on recursively partitioning the graph with approximate separators, yielding an approximation factor of $O(\log n \log k)$.

In a recent paper, Even et al. [3] employed spreading metrics and improved the approximation factor from $O(\log^2 n)$ to $O(\log n \log\log n)$ for a variety of graph optimization problems. The graph partitioning framework presented here can be used to cast the problems we deal with into the paradigm of [3], yielding somewhat weaker approximation factors than the factors presented here. In fact, since we are unable to apply the algorithm presented here to the k-balanced partitioning problem in directed graphs, the best approximation algorithm for this problem is given by applying our framework along with the recursion presented in [3], yielding an $O(\log n \log\log n)$-approximation factor.

We are able to obtain $O(\log n)$-approximation factors as compared to the $O(\log n \log\log n)$-approximation factors in [3] because we can partition the graph in each iteration into two parts, the smaller part being small enough so that further partitioning of it is not required. Hence, we often refer to such a partitioning as "chopping off" a piece. In [3] we were not able to guarantee that one of the parts is trivial, and hence both parts needed to be recursively partitioned.

The simultaneous separation problem was considered previously only for the case where the number of sets to be partitioned is a constant, e.g., [9, p. 67]. These instances of the simultaneous separation problem have been solved by recursively applying the Leighton–Rao algorithm. Since the number of sets to be partitioned has a logarithmic effect on the approximation factor, this approach yields weaker approximation factors than the approximation factors we present. Note that the Steiner multicut problem, considered in Klein et al. [10], can be modeled as a simultaneous separation problem. However, the separation in the Steiner multicut problem is not required to be balanced at all, and thus our approximation algorithm yields inferior results as compared to [10].

Although the weighted b-balanced cut problem was considered previously by several researchers, the literature lacks a description of an approximation algorithm for the weighted case, and only a brief sketch is available in [13].

The paper is organized as follows. In section 2 we define the problems considered in this paper. In section 3 we consider spreading metrics for ρ-separators in undirected graphs. In section 4 we describe an approximation algorithm for finding a ρ'-separator in undirected graphs. In section 5 we consider ρ'-separators in directed graphs. In section 6 we consider simultaneous separators. Finally, in section 7 we describe an efficient algorithm for computing suboptimal spreading metrics.

2. The problems. In this section we define the problems considered in this paper. We start with the well-known problems of finding *balanced cuts* and *balanced partitions*. Then, we define the problem of finding ρ-separators that captures the two previous problems as a special case. We also define a more general problem that we call *simultaneous separators*. All of these problems are NP-complete [5].

To simplify the presentation, we define the "edge" version of the problems and consider only undirected graphs. We end this section with remarks about the "vertex" version of these problems and the directed case.

Let $G = (V, E)$ be an undirected graph with nonnegative edge capacities $\{c(e)\}_e$ and nonnegative vertex weights $\{w(v)\}_v$. For $S \subset V$, a cut $(S, V - S)$ is a subset of E that disconnects S from $V - S$. For $F \subseteq E$, let $c(F) \triangleq \sum_{e \in F} c(e)$. For $U \subseteq V$, let $w(U) \triangleq \sum_{v \in U} w(u)$.

The b-balanced cut problem. Given a balance parameter $0 < b \leq 1/2$ and a graph

$G = (V, E)$, a *b-balanced cut* in G is a cut $(S, V - S)$ that satisfies $b \cdot w(V) \leq w(S) \leq (1 - b) \cdot w(V)$. The b-balanced cut problem with input $G = (V, E)$ and b is to find a minimum capacity b-balanced cut in G.

When the balance parameter equals $1/2$, the balanced condition is often relaxed to $\lfloor b \cdot w(V) \rfloor \leq w(S) \leq \lceil (1 - b) \cdot w(V) \rceil$. This relaxation avoids having an infeasible constraint in the case of bisection due to "rounding" problems.

The (k, ν)-balanced partitioning problem. Given integer $k \geq 2$ and a real $\nu \geq 1$, a *(k, ν)-balanced partition* of $G = (V, E)$ is a subset of the edges whose removal partitions the graph into at most k parts, where each part consists of a union of connected components and the sum of the vertex weights in each part is at most $\frac{\nu}{k} \cdot w(V)$. The (k, ν)-balanced partitioning problem with input $G = (V, E)$, k, and ν is to find a minimum capacity (k, ν)-balanced partition of G.

The following claim implies that, without loss of generality, we may assume that $\nu \leq 2$.

Claim 2.1. Any (k, ν)-balanced partition, where $\nu \geq 2$, induces a (k', ν')-balanced partition, where $k' < k$ and $\nu' < 2$.

Proof. Let F denote a (k, ν)-balanced partition of $G = (V, E)$, where $\nu \geq 2$. Let $A_1, A_2, \ldots, A_s$ denote the vertex sets of the connected components in the graph $G' = (V, E - F)$, where $s \leq k$. Define $w_i = w(A_i)/w(V)$. By definition, $w_i \leq \frac{\nu}{k}$ for every $1 \leq i \leq s$. Merge connected components, until $w_i + w_j > \frac{\nu}{k}$, for every $i \neq j$. Let F' denote the edges connecting vertices belonging to different components (after the merging of small components). To simplify notation, assume that no merging took place. Below, we prove that $s < \frac{2k}{\nu} \leq k$. We get that F is an (s, ν')-balanced partition, where $\frac{\nu'}{s} = \frac{\nu}{k}$. Substituting $s < \frac{2k}{\nu}$, we get $\nu' < 2$.

We now prove that $s < \frac{2k}{\nu}$. Since $w_i + w_j > \frac{\nu}{k}$, for every $i \neq j$, there is at most one index ℓ for which $w_\ell \leq \frac{\nu}{2k}$. We distinguish between two cases.

Case 1. $w_i > \frac{\nu}{2k}$ for all $1 \leq i \leq s$. In this case $1 = \sum_{i=1}^{s} w_i > s\frac{\nu}{2k}$, and the upper bound on s follows.

Case 2. There exists an index $1 \leq \ell \leq s$ for which $w_\ell \leq \frac{\nu}{2k}$. In this case for all indices $1 \leq i \leq s$, where $i \neq \ell$, $w_i > \frac{\nu}{k} - w_\ell$. Summing up we get $1 = \sum_{i=1}^{s} w_i > (s - 1)(\frac{\nu}{k} - w_\ell) + w_\ell$. This implies

$$s < \frac{k + \nu - 2kw_\ell}{\nu - kw_\ell} = 2 + \frac{k - \nu}{\nu - kw_\ell} \leq 2 + \frac{k - \nu}{\nu - \nu/2} = \frac{2k}{\nu}. \qquad \square$$

From now on we refer to $(k, 2)$-balanced partitions as k-balanced partitions.

The ρ-separator problem. Given $0 < \rho \leq 1$, a *ρ-separator* in $G = (V, E)$ is a subset of edges whose removal partitions the graph into connected components such that the sum of the vertex weights in each component is at most $\rho \cdot w(V)$. The ρ-separator problem with input $G = (V, E)$ and ρ is to find a minimum capacity ρ-separator in G.

The simultaneous separator problem. Let $\boldsymbol{U} \triangleq \{U_1, U_2, \ldots, U_s\}$ denote a set of subsets of vertices, namely, $U_i \subseteq V$, for every i. Let $\boldsymbol{\rho} \triangleq \{\rho_1, \rho_2, \ldots, \rho_s\}$ denote a set of parameters that satisfies $0 < \rho_i \leq 1$, for every i. Given such $\boldsymbol{U}$ and $\boldsymbol{\rho}$, a *$(\boldsymbol{U}, \boldsymbol{\rho})$-simultaneous separator* is a subset of edges whose removal partitions the graph into connected components such that for each such component $H = (U', E')$

$$\forall\, 1 \leq i \leq s \;:\; w(U' \cap U_i) \leq \rho_i \cdot w(U_i).$$

The simultaneous separator problem with input $G = (V, E)$, $\boldsymbol{U}$, and $\boldsymbol{\rho}$ is to find a minimum capacity $(\boldsymbol{U}, \boldsymbol{\rho})$-simultaneous separator in G.

We note that our algorithm can be extended to solve the more general case in which each node v has s weight values associated with it, and the partitioning constraint for U_i and ρ_i is with respect to the ith weight measure.

We next exhibit certain connections between the problems defined herein. Clearly, every b-balanced cut is also a $(1-b)$-separator. The next claim shows that if $b \leq 1/3$, then every $(1-b)$-separator induces a b-balanced cut of at most the same cost. Together, these two assertions imply that the balanced cut problem for $b \leq 1/3$ is equivalent to the ρ-separator problem (where $\rho = 1 - b \geq 2/3$).

Claim 2.2. If $0 < b \leq 1/3$, then every $(1-b)$-separator, F, induces a b-balanced cut $(A, V-A)$ such that $c(A, V-A) \leq c(F)$.

Proof. Let $F \subseteq E$ denote a $(1-b)$-separator of $G = (V, E)$. Let $A_1, A_2, \ldots, A_s$ denote the vertex-sets of the connected components in the graph $G' = (V, E-F)$. If there exists an A_i such that $w(A_i) \geq b \cdot w(V)$, then the cut $(A_i, V - A_i)$ is a b-balanced cut whose capacity is at most the capacity of the ρ-separator. Otherwise, define $i_0 = \max\{i : w(\cup_{j \leq i} A_j) < b \cdot w(V)\}$. Let $A = A_1 \cup \cdots A_{i_0+1}$, then $w(A) < 2b \cdot w(V)$. Since $b \leq 1/3$, we conclude that $b \cdot w(V) \leq w(A) < (1-b) \cdot w(V)$, and hence $(A, V-A)$ is a b-balanced cut whose cost is at most the capacity of the ρ-separator. ∎

It follows from the definitions that every k-balanced partition induces a $2/k$-separator. We claim that every ρ-separator induces a $(\lceil 2/\rho \rceil - 1)$-balanced partition.

Claim 2.3. Every ρ-separator induces a $(\lceil 2/\rho \rceil)$-balanced partition.

Proof. Let F denote a ρ-separator of $G = (V, E)$. Let $A_1, A_2, \ldots, A_s$ denote the vertex sets of the connected components in the graph $G' = (V, E - F)$. Define $w_i = w(A_i)/w(V)$. By definition, $w_i \leq \rho$ for every $1 \leq i \leq s$. Merge connected components, until $w_i + w_j > \rho$, for every $i \neq j$. Let F' denote the edges connecting vertices belonging to different components (after the merging of small components). To simplify notation, assume that no merging took place. Similar to Claim 2.2 it can be shown that $s < 2/\rho$, and the claim follows.

We conclude with some remarks.

1. In the vertex version of the above problems, every vertex has both a capacity and a weight. The problem is then to find a minimum capacity subset of vertices whose removal partitions the graph into connected components, the weight of which satisfies the relevant constraint.
2. The edge version is reducible to the vertex case by introducing dummy vertices on each edge. Hence, we can also consider "heterogeneous" versions of the simultaneous separation problem in which the separation constraints are on subsets containing both edges and vertices.
3. All the problems can be extended to directed graphs. In this case, the definitions should be modified, and "connected components" should be replaced with "strongly connected components."

3. A spreading metric for ρ-separators in undirected graphs. A spreading metric is an assignment of edge lengths that satisfies two properties: (a) it provides a lower bound, and (b) it satisfies a radius guarantee. Obviously, we are also interested in computing spreading metrics efficiently. We obtain a spreading metric by writing a linear program and showing that its optimal solutions are spreading metrics. The integer program corresponding to the linear program is a direct formulation of ρ-separators, as shown in the lower bound property below.

We start by defining a spreading metric. Note that two parameters are involved: our lower bound is with respect to ρ-separators, and our goal is to find a ρ'-separator, for $\rho' > \rho$. Therefore, the radius guarantee is stated with respect to ρ'.

Notation. Given edge lengths $\{d(e)\}_{e\in E}$ and a subset of vertices S, let $\mathsf{dist}_S(v,u)$ denote the distance in the subgraph induced by S between the vertices v and u.

DEFINITION 3.1. *An assignment of nonnegative edge lengths $\{d(e)\}_{e\in E}$ is a spreading metric for ρ-separators which can be used for finding ρ'-separators (where $\rho' > \rho$), if it satisfies the following properties:*

1. *Lower bound: The* volume *of the assignment, defined by $\sum_{e\in E} c(e)d(e)$, is a lower bound on the minimum capacity of a ρ-separator.*
2. *Radius guarantee: Loosely speaking, every "heavy" subset of vertices has at least constant radius. Formally, for every subset $S \subseteq V$ for which $w(S) > \rho' \cdot w(V)$, and for every vertex $v \in S$,*

$$\mathsf{radius}(v,S) > \frac{\rho'-\rho}{\rho'},$$

where $\mathsf{radius}(v,S) \stackrel{\triangle}{=} \max\{\mathsf{dist}_S(v,u) : u \in S\}$.

We obtain a spreading metric for ρ-separators by writing a linear program that attaches edge lengths $d(e)$ to each edge $e \in E$. The linear program is defined as follows:

$$\text{(P1)} \qquad \min \sum_{e\in E} c(e)\cdot d(e)$$

$$\text{subject to (s.t.) } \forall S \subseteq V \quad \forall v \in S : \sum_{u\in S} \mathsf{dist}_V(v,u)\cdot w(u) \geq w(S) - \rho\cdot w(V)$$

$$\forall e \in E : \ 0 \leq d(e) \leq 1.$$

Note that the first type of constraints holds trivially if $w(S) \leq \rho \cdot w(v)$.

One could formulate the linear program using $\mathsf{dist}_S(v,u)$ rather than $\mathsf{dist}_V(v,u)$. Let ($\mathrm{P1}_S$) denote the linear program obtained from (P1) by using $\mathsf{dist}_S(v,u)$ instead of $\mathsf{dist}_S(v,u)$. Since $\mathsf{dist}_S(v,u)$ is never smaller than $\mathsf{dist}_V(v,u)$, it follows that the feasible solutions of (P1) are a subset of the feasible solutions of ($\mathrm{P1}_S$), and hence the cost of a minimum cost solution of ($\mathrm{P1}_S$) is not greater than the minimum cost of (P1). In section 3 we show that the feasible solutions of ($\mathrm{P1}_S$) are also feasible solutions of (P1), and hence the optima of the two linear programs are equal.

In section 7.1 we consider a modification of ($\mathrm{P1}_S$) in which only a subset of the constraints of ($\mathrm{P1}_S$) is kept, namely, constraints corresponding to subsets of weight greater than $\rho' \cdot w(V)$.

Notation. Let τ denote the cost of an optimal (fractional) solution of the linear program (P1).

We now show that an optimal solution of (P1) is a spreading metric, namely, that τ is a lower bound, and that the radius guarantee is satisfied. We then show that an optimal solution of (P1) can be computed in polynomial time.

Lower bound. We show that every ρ-separator, F, induces a feasible $\{0,1\}$-solution of the linear program by defining $d(e) = 1$, if $e \in F$, and $d(e) = 0$, otherwise. Consider a subset $S \subseteq V$ that satisfies $w(S) > \rho \cdot w(V)$, and consider a vertex $v \in S$. We need to show that the constraint corresponding to S and v is satisfied. Let $\mathsf{comp}_F(v)$ denote the connected component that contains v in the graph $G' = (V, E - F)$. Every path from v to a vertex $u \in S - \mathsf{comp}_F(v)$ must contain at least

one edge $e \in F$ whose length equals 1; hence $\mathsf{dist}_V(v,u) \geq 1$. Therefore,

$$\begin{aligned}\sum_{u \in S} \mathsf{dist}_V(v,u) \cdot w(u) &\geq \sum_{u \in S - \mathsf{comp}_F(v)} \mathsf{dist}_V(v,u) \cdot w(u) \\ &\geq w(S - \mathsf{comp}_F(v)) \\ &\geq w(S) - w(\mathsf{comp}_F(v)) \\ &\geq w(S) - \rho \cdot w(V).\end{aligned}$$

The cost of the integral solution induced by a ρ-separator equals the capacity of the ρ-separator. Hence, the cost of an optimal fractional solution of the linear program cannot be greater than the cost of a minimum capacity ρ-separator.

Radius guarantee. Every feasible solution of (P1) satisfies the following radius guarantee.

LEMMA 3.2. *If $S \subseteq V$ satisfies $w(S) > \rho' \cdot w(V)$, then for every vertex $v \in S$,*

$$\mathsf{radius}(v,S) > \frac{\rho' - \rho}{\rho'}.$$

Proof. Fix S and v, and consider the constraint corresponding to them. This constraint gives a lower bound on the weighted sum of the distances from v to the vertices in S. Hence, it provides a lower bound on the weighted average distance from v to the vertices in S. Let $u \in S$ denote a vertex whose distance from v is not less than the weighted average distance of a vertex in S from v. Then,

$$\mathsf{dist}_V(v,u) \quad \geq \quad 1 - \frac{\rho \cdot w(V)}{w(S)} \quad \geq \quad 1 - \frac{\rho \cdot w(V)}{\rho' \cdot w(V)}.$$

Since $\mathsf{dist}_S(v,u) \geq \mathsf{dist}_V(v,u)$, the lemma follows. ∎

Polynomial time computability. The formulation for the linear program (P1) is shorthand for a linear program that contains exponentially many constraints: For every "heavy" subset S, and for every $v \in S$, consider all combinations of paths from v to all other vertices in S, and require that the weighted sum of the lengths of the paths satisfies the corresponding constraint. Since the number of constraints is exponential, naive implementation of known polynomial time LP-solvers would not provide an efficient way of computing spreading metrics. This is because the running time of these solvers is polynomial in both the number of variables and the number of constraints.

We describe below how a spreading metric can be computed in polynomial time using the ellipsoid algorithm [15]. Algorithms for computing spreading metrics with better asymptotic running times are presented in section 7.

Applying the ellipsoid algorithm requires a polynomial-time separation procedure that, given a candidate solution $\{d(e)\}_e$, either finds a violated constraint or proves that it is indeed a solution.

We rewrite the constraints of (P1) as follows:

$$\forall S \subseteq V, \quad \forall v \in S : \sum_{u \in S} (\mathsf{dist}_V(v,u) - 1) \cdot w(u) \geq -\rho \cdot w(V). \tag{3.1}$$

Given a vertex v, the left-hand side of (3.1) is minimized when the subset $S_v = \{u : \mathsf{dist}_V(v,u) \leq 1\}$ is chosen. Therefore, if the constraint corresponding to v and S_v

is satisfied, then all the constraints in which distances are measured from v are also satisfied. This means that if the constraints corresponding to v and S_v are satisfied for every vertex v, then the edge lengths constitute a feasible solution to (P1). Hence, a polynomial-time separation procedure exists.

Since S_v is a sphere of radius 1, it follows that $\mathsf{dist}_V(v,u) = \mathsf{dist}_S(v,u)$ for every $u \in S_v$. Therefore, the linear programs (P1) and (P1$_S$) have the same feasible solutions and the same optima.

4. Finding a ρ'-separator. In this section we describe the approximation algorithm for ρ'-separators. The input consists of a graph $G = (V, E)$ with edge capacities $c(e)$ and vertex weights $w(v)$. The input also contains a spreading metric for a ρ-separator ($\rho' \geq \rho$), represented by edge lengths $\{d(e)\}$ for all $e \in E$.

The algorithm consists of a cut procedure that partitions a subset of vertices whose weight is larger than $\rho' \cdot w(V)$. This procedure cuts off a subset of vertices with weight at most $\rho' \cdot w(V)$, and hence the chopped off pieces need not be further partitioned. Initially, the cut procedure is applied to the whole graph; then it is applied iteratively to the remaining part until its weight is not greater than $\rho' \cdot w(V)$.

The cut procedure presented in this section is identical to the sphere growing procedure in [6]; the only difference is in the parameters used in the definition of the volume and the upper bound on the radius. Our presentation emphasizes the ability to find a cut whose capacity is logarithmic in the volume of the spreading metric when the diameter of the graph is constant (see (4.2)).

4.1. Assigning volumes. Following Garg, Vazirani, and Yannakakis [6] we assign volumes to spheres that are grown around vertices. These volumes will define the credit that is attributed to the spheres in order to pay for the cut disconnecting them from the rest of the graph. Our definition of volume deviates slightly from the definition of [6] so as to reduce the constants.

DEFINITION 4.1. *Let v denote a vertex in a subgraph induced by V'. The* r-sphere centered at v, *denoted by $N(v,r)$, is defined by*

$$N(v,r) \triangleq \{u : \mathsf{dist}_{V'}(u,v) < r\}.$$

Note that $N(v,r)$ includes only vertices whose distance from v is strictly less than r.

Notation. Let $E(v,r)$ denote the set of edges whose endpoints belong to $N(v,r)$, namely, $E(v,r) = E \cap (N(v,r) \times N(v,r))$. Let $\mathsf{cut}(v,r)$ denote the set of edges that belong to the cut $(N(v,r), V - N(v,r))$.

The volume of a sphere is a positive value that we attach to spheres and is the sum of three components: (1) a "seed value" set to $\varepsilon \cdot \tau$, where τ is the cost of the optimal fractional solution, (2) the contribution of edges in $E(v,r)$, and (3) the contribution of edges in $\mathsf{cut}(v,r)$.

DEFINITION 4.2. *The* volume *of $N(v,r)$, denoted by $\mathsf{vol}(v,r)$, is defined by*

$$\mathsf{vol}(v,r) \triangleq \varepsilon \cdot \tau + \sum_{e \in E(v,r)} c(e) \cdot d(e) + \sum_{(x,y) \in \mathsf{cut}(v,r)} c(x,y) \cdot (r - \mathsf{dist}_{N(v,r)}(v,x)),$$

where $\varepsilon = \frac{1}{n \ln n}$.

4.2. The cut procedure. In this section, we describe a procedure for partitioning a subgraph $G' = (V', E')$ of G into two parts, given a spreading metric for a ρ-separator of G. The procedure is applied when $w(V') > \rho' \cdot w(V)$, and it finds a

cut-proc $(V', E', \{c(e)\}_{e\in E'}, \{d(e)\}_{e\in E'})$
$\tilde{r} = \frac{\rho'-\rho}{\rho'}$
choose an arbitrary $v \in V'$.
$T = \{v\}$
v' = closest vertex to T in $V' - T$.
while $c(T, V' - T) > \frac{1}{\tilde{r}} \cdot \ln\left(\frac{\mathsf{vol}(v,\tilde{r})}{\mathsf{vol}(v,0)}\right) \cdot \mathsf{vol}(v, \mathsf{dist}_{V'}(v, v'))$ do
begin
$T = T \cup \{v'\}$
v' = closest vertex to T in $V' - T$.
end
Return T

FIG. 4.1. *The cut procedure.*

sphere $N(v, r)$ that satisfies the following two properties: (1) $w(N(v, r)) \le \rho' \cdot w(V)$, and (2) the ratio between $c(\mathsf{cut}(v, r))$ and $\mathsf{vol}(v, r)$ is logarithmic.

4.2.1. Description. The cut procedure runs Dijkstra's single source shortest paths algorithm from an arbitrary vertex $v \in V'$. In each iteration, before adding the closest vertex $v' \in V'$ to the set of vertices whose distances from v are already determined, the procedure checks whether $r = \mathsf{dist}_{V'}(v, v')$ is a "good" radius. The procedure stops when it finds the first good radius and returns $N(v, r)$ as the set of vertices to be chopped off. The procedure, called "cut-proc," is depicted in Figure 4.1.

THEOREM 4.3. *Given a subgraph $G' = (V', E')$ that satisfies $w(V') > \rho' \cdot w(V)$, and given a spreading metric $\{d(e)\}_e$, procedure cut-proc finds a subset $T \subset V'$ that satisfies $w(T) \le \rho' \cdot w(V)$.*

4.2.2. Proof of Theorem 4.3. Let $v \in V'$ be the vertex chosen by the cut procedure. Consider $\mathsf{vol}(v, r)$ as a function of r, as depicted in Figure 4.2. Note that it is a monotone piecewise linear function whose initial value equals $\varepsilon \cdot \tau$ and whose final value equals $(1 + \varepsilon) \cdot \tau$. The endpoints of the linear segments correspond to distances measured from v to vertices in V'. In other words, if we sort the vertices of V' in ascending distance order from v: $v = v_0, v_1, \ldots, v_n$, then for every $0 \le i < n$ the function $\mathsf{vol}(v, r)$ is linear in the closed interval $r \in [\mathsf{dist}(v_0, v_i), \mathsf{dist}(v_0, v_{i+1})]$. Moreover, the derivate $\mathsf{vol}'(v, r) \triangleq \frac{d\mathsf{vol}(v,r)}{dr}$ in the open interval $(\mathsf{dist}(v_0, v_i), \mathsf{dist}(v_0, v_{i+1}))$ equals $c(\mathsf{cut}(v, r))$. This can readily be seen by observing that in this open interval, only the cut edges contribute to the growth of the volume, and that the coefficient of r equals the sum of the capacities of the cut edges.

The cut procedure searches for a radius r for which the ratio between $\mathsf{vol}'(v, r)$ and $\mathsf{vol}(v, r)$ is logarithmic. We need to show that such a radius $r \le \tilde{r}$ exists and that the procedure finds it. Note that $r \le \tilde{r}$ implies that $w(v, r) \le \rho' \cdot w(V)$ (see Lemma 3.2).

Recall that $\tilde{r} \triangleq \frac{\rho'-\rho}{\rho'}$ and let I_i denote the open interval $(\mathsf{dist}(v_0, v_i), \mathsf{dist}(v_0, v_{i+1}))$. The following claim proves the existence of a good radius.

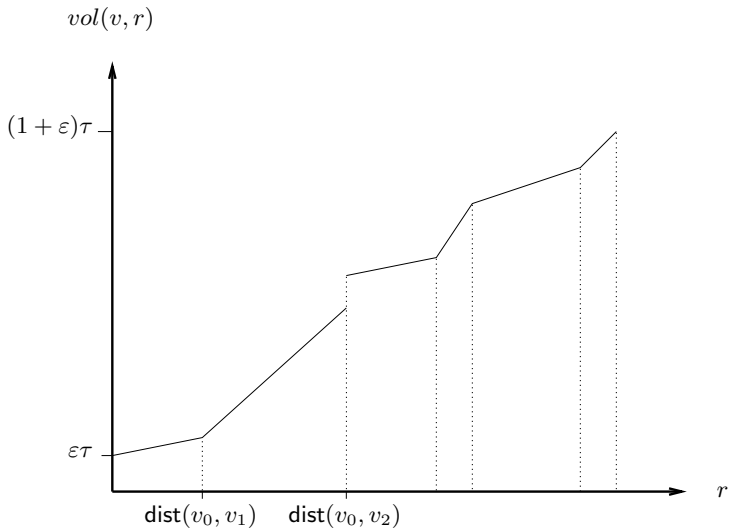


FIG. 4.2. $\mathsf{vol}(v,r)$ *as a function of* r.

Claim 4.4. For every vertex $v \in V'$, there exists an $r \in (0, \tilde{r}] \cap \cup_i I_i$ that satisfies

$$\frac{\mathsf{vol}'(v,r)}{\mathsf{vol}(v,r)} \leq \frac{1}{\tilde{r}} \cdot \ln\left(\frac{\mathsf{vol}(v,\tilde{r})}{\mathsf{vol}(v,0)}\right).$$

Proof. By contradiction, suppose that for every $r \in (0, \tilde{r}] \cap \cup_i I_i$,

$$\frac{\mathsf{vol}'(v,r)}{\mathsf{vol}(v,r)} > \frac{1}{\tilde{r}} \cdot \ln\left(\frac{\mathsf{vol}(v,\tilde{r})}{\mathsf{vol}(v,0)}\right).$$

The left-hand side of the above equation is defined and continuous in the interval $(0, \tilde{r}]$ except for finitely many points. Hence, we may take the integral of both sides, yielding

$$\int_0^{\tilde{r}} \frac{\mathsf{vol}'(v,r)}{\mathsf{vol}(v,r)} dr > \frac{1}{\tilde{r}} \ln\left(\frac{\mathsf{vol}(v,\tilde{r})}{\mathsf{vol}(v,0)}\right) \int_0^{\tilde{r}} dr. \tag{4.1}$$

Using the identity $\int \frac{f'(x)}{f(x)} dx = \ln f(x)$, we can evaluate the left-hand side of inequality (4.1),

$$\begin{aligned}\int_0^{\tilde{r}} \frac{\mathsf{vol}'(v,r)}{\mathsf{vol}(v,r)} dr &= \ln(\mathsf{vol}(v,\tilde{r})) - \ln(\mathsf{vol}(v,0)) \\ &= \ln\left(\frac{\mathsf{vol}(v,\tilde{r})}{\mathsf{vol}(v,0)}\right).\end{aligned}$$

However, the right-hand side of inequality (4.1) also equals the same value (a contradiction) and the claim follows. □

The cut-proc procedure considers only radii in the set $\{\mathsf{dist}_{V'}(v,u)\}_{u \in V'}$. (Note that since T is a sphere in V', it follows that $\mathsf{dist}_{V'}(v,v') = \mathsf{dist}_T(v,v')$ for every vertex $v' \in T$.) In order to enable computability of a good radius, we show that among the

distances $\{\mathsf{dist}_{V'}(v,u)\}_{u\in V'}$ there exists a good radius. The following claim proves this property.

Claim 4.5. Let $v \in V'$ denote an arbitrary vertex. Let r_0 denote the radius whose existence is guaranteed by Claim 4.4. Define r_1 to be

$$r_1 \stackrel{\triangle}{=} \min\left\{\mathsf{dist}_{V'}(v,u) : u \in V' \text{ and } \mathsf{dist}_{V'}(v,u) \geq r_0\right\}.$$

Then, r_1 satisfies

$$c(\mathsf{cut}(v,r_1)) \leq \frac{1}{\tilde{r}} \cdot \ln\left(\frac{\mathsf{vol}(v,\tilde{r})}{\mathsf{vol}(v,0)}\right) \cdot \mathsf{vol}(v,r_1), \tag{4.2}$$

$$w(N(v,r_1)) \leq \rho' \cdot w(V'). \tag{4.3}$$

Proof. Since $N(v,r_1)$ consists of the vertices whose distance from v is strictly less that r_1, it follows that $N(v,r_0) = N(v,r_1)$, and hence these spheres define the same cut. In particular, $c(\mathsf{cut}(v,r_1)) = \mathsf{vol}'(v,r_0)$. Inequality (4.2) follows from Claim 4.4 and from $\mathsf{vol}(v,r_0) \leq \mathsf{vol}(v,r_1)$.

Since $N(v,r_1) = N(v,r_0)$, and since $r_0 \leq \tilde{r}$, Lemma 3.2 implies inequality (4.3), and the claim follows. ∎

Claims 4.4 and 4.5 show that among the set of radii $\{\mathsf{dist}_{V'}(v_0,v_i)\}_i$ there exists a good radius. The cut procedure simply performs an exhaustive search in ascending distance order and stops as soon as it finds the first radius that satisfies inequality (4.2). The found radius also satisfies inequality (4.3) (since it is not greater than r_1), and hence the theorem follows.

4.3. Analysis of the approximation factor. In this section we prove the $O(\log n)$-approximation factor of the proposed algorithm. We also outline how the algorithm may be modified to obtain an $O(\log \tau)$-approximation algorithm.

The cut procedure is called repetitively until the weight of the remaining subset of vertices drops below (or becomes equal to) $\rho' \cdot w(V)$. Each call chops off a sphere, and the chopped off spheres are disjoint. The volume of each sphere consists of the contribution of the edges contained inside, the contribution of the edges of the cut, and the contribution of the seed value. The sum of the contributions of the edges and edges of the cut to the volumes of the chopped off spheres is bounded by τ. The sum of the seed values is bounded by $\varepsilon\tau n$.

The capacity of the cut $(T, V' - T)$ found by the cut procedure satisfies

$$c(T, V' - T) \leq \frac{1}{\tilde{r}} \cdot \ln\left(\frac{\mathsf{vol}(v,\tilde{r})}{\mathsf{vol}(v,0)}\right) \cdot \mathsf{vol}(v, \mathsf{dist}_{V'}(v,v')).$$

But $\mathsf{vol}(v,\tilde{r}) \leq (1+\varepsilon)\tau$ and $\mathsf{vol}(v,0) = \varepsilon\tau$. Hence, the sum of the capacities of the cuts found by the calls to the cut procedure is bounded by

$$\frac{1}{\tilde{r}} \cdot \ln\left(\frac{1+\varepsilon}{\varepsilon}\right) \cdot \tau \cdot (1+\varepsilon n).$$

Plugging in $\varepsilon = \frac{1}{n \ln n}$ yields the following theorem.

THEOREM 4.6. *The algorithm presented above finds a ρ'-separator whose capacity is at most*

$$\left(\frac{\rho'}{\rho' - \rho} + o(1)\right) \cdot \ln n \cdot \tau,$$

where τ is the volume of the spreading metric, and hence is a lower bound on the cost of an optimal ρ-separator.

One could also obtain an $O(\log(\tau/\gamma))$-approximation factor, where γ is the minimum edge capacity. From now on, we assume without loss of generality that the minimum edge capacity is one, and we show how to obtain an $O(\log \tau)$-approximation factor. This is done by the following modifications. (1) Define $\varepsilon = 0$; namely, do not add seed values to the volume of spheres. (2) Let δ denote a positive parameter to be specified later. Start the cut procedure with an initial sphere of radius $\tilde{r}/(1+\delta)$, and substitute all occurrences of $\mathsf{vol}(v, 0)$ with $\mathsf{vol}(v, \tilde{r}/(1+\delta))$. (3) Instead of considering the half-open interval $(0, \tilde{r}]$, consider the half-open interval $(\tilde{r}/(1+\delta), \tilde{r}]$. Now, substitute all occurrences of $\frac{1}{\tilde{r}}$ with $\frac{1+1/\delta}{\tilde{r}}$. (This is the reciprocal of the length of the half-open interval.) After these modifications, the sum of the volumes of the chopped off pieces is bounded by τ. Since $c(e) \geq 1$, for every edge $e \in E$ we have $\mathsf{vol}(v, \tilde{r}/(1+\delta)) \geq \tilde{r}/(1+\delta)$ (otherwise the subgraph is not connected). Thus, the sum of the capacities of the cuts found by the calls to the cut procedure is bounded by

$$\frac{1+1/\delta}{\tilde{r}} \cdot \ln\left(\frac{(1+\delta)\tau}{\tilde{r}}\right) \cdot \tau.$$

Assign, for example, $\delta = e^5 - 1$, and the capacity of the ρ'-separator is bounded by

$$\frac{1.01 \cdot \rho}{\rho' - \rho} \cdot \left(5 + \ln\left(\frac{\rho}{\rho' - \rho} \cdot \tau\right)\right) \cdot \tau.$$

Thus, we obtain an $O(\log \tau)$-approximation factor.

5. ρ'-separators in directed graphs. In this section we describe the approximation algorithm for ρ'-separators in directed graphs when $\rho' \geq 1/2$. This algorithm is very similar to the algorithm for undirected separators; however, our control over the weight of the chopped off pieces is diminished, and we can guarantee only that the weight of chopped off pieces is at most half of the weight of the vertices in the remaining subgraph. Hence, our algorithm is applicable only for values of $\rho' \geq 1/2$ and the approximation factors are larger by a factor of 4. We first define the spreading metrics for directed graphs and later point out the modifications of the undirected ρ-separator algorithm required for directed separators.

5.1. Spreading metrics for directed graphs. The spreading metric in the directed case for a parameter ρ is obtained by the following linear program:

$$\min \sum_{e \in E} c(e) \cdot d(e)$$

$$\text{s.t. } \forall S \subseteq V,\ \forall v \in S:\ \sum_{u \in S} (\mathsf{dist}_V(v,u) + \mathsf{dist}_V(u,v)) \cdot w(u) \geq w(S) - \rho \cdot w(V)$$

$$\forall e \in E:\ 0 \leq d(e) \leq 1.$$

We omit the proofs that an optimal spreading metric is computable in polynomial time and that it is a lower bound on the capacity of an optimal ρ-separator. We also omit the proof that the spreading metric satisfies a modified radius guarantee that is stated below.

LEMMA 5.1. *If $S \subseteq V$ satisfies $w(S) > \rho' \cdot w(V)$, then for every vertex $v \in S$ there exists a vertex $u \in S$ such that*

$$\mathsf{dist}_S(v,u) + \mathsf{dist}_S(u,v) > \frac{\rho' - \rho}{\rho'}.$$

5.2. Finding a ρ'-separator in directed graphs. In each iteration we find a pair of vertices $s,t \in V'$ that achieves the diameter; namely, $\mathsf{dist}_{V'}(s,t) = \max\{\mathsf{dist}_{V'}(u,v) : u,v \in V'\}$. Note that Lemma 5.1 implies that $\mathsf{dist}(s,t) > \frac{\rho'-\rho}{2\rho'}$. After finding s and t, we call the cut procedure twice: the first instance with the remaining subgraph $G' = (V', E')$ and s as the initial vertex, and the second instance with the reversed subgraph $G'_{rev}(V', E'_{rev})$ and t as the initial vertex. (A reversed graph is obtained by reversing the directions of the edges while preserving the edge weights and capacities.) The cut procedure assigns volumes in the same fashion as in the undirected case. However, we modify the definition of $\tilde{r}$ in the cut procedure to be $\tilde{r} = \frac{\rho'-\rho}{4\rho'}$. Let T denote the subset of vertices returned by the first call to the cut procedure, and let T_{rev} denote the subset of vertices returned by the second call. In the following theorem, we claim that $\min\{w(T), w(T_{rev})\} \leq w(V')/2$. If $w(T) \leq W(V')/2$, then we choose the cut $(T, V' - T)$ and chop off T. Otherwise, we choose the cut $(V' - T_{rev}, T_{rev})$ and chop off T_{rev}.

THEOREM 5.2. *Let $G' = (V', E')$ be a directed subgraph that satisfies $w(V') > \rho' \cdot w(V)$. Given a (directed) spreading metric $\{d(e)\}_e$, let $s,t \in V'$ satisfy $\mathsf{dist}(s,t) > \frac{\rho'-\rho}{2\rho'}$. Let T and T_{rev} be the subsets of vertices returned by two calls to the procedure cut-proc: the first with input G' and s as its initial vertex, and the second with input G'_{rev} and t as its initial vertex. Then, the weights $w(T)$ and $w(T_{rev})$ satisfy* $\min\{w(T), w(T_{rev})\} \leq w(V')/2$.

Proof. Let $N = N(s,\tilde{r})$ and $N_{rev} = N_{rev}(t,\tilde{r})$, where $N_{rev}(t,\tilde{r})$ denotes a sphere in the reversed graph (recall that we now use $\tilde{r} = \frac{\rho'-\rho}{4\rho'}$). Since $\mathsf{dist}(s,t) > 2 \cdot \tilde{r}$, it follows that N and N_{rev} are disjoint. Since $w(N) + w(N_{rev}) \leq w(V')$, we conclude that either $w(N)$ or $w(N_{rev})$ is at most $w(V')/2$.

To complete, we show that $T \subseteq N$ and $T_{rev} \subseteq N_{rev}$, and hence $\min\{w(T), w(T_{rev})\} \leq \min\{w(N), w(N_{rev})\} \leq w(V')/2$. This is easy since Claims 4.4 and 4.5 (without inequality (4.3)) are applicable, and hence the theorem follows. ∎

5.3. Approximation factor in directed graphs. In the directed case, the ratio between the capacity of the directed ρ'-separator and the volume of the spreading is four times larger than the corresponding ratio in the undirected case. This is due to the fact that in the directed case $\tilde{r} = \frac{\rho'-\rho}{4\rho}$. Hence the approximation factors increase by a factor of 4 as compared to the approximation factors in the undirected case.

6. Simultaneous separators. In this section we review the modifications that are needed to solve the problem of simultaneous separators with similar approximation factors. As in the other problems, we describe a pseudoapproximation algorithm. The input consists of a graph with edge capacities and vertex weights, a sequence of subsets of vertices $\boldsymbol{U} = \{U_1, U_2, \ldots, U_s\}$, and two sequences of separation parameters $\boldsymbol{\rho} = \{\rho_1, \rho_2, \ldots, \rho_s\}$ and $\boldsymbol{\rho'} = \{\rho'_1, \rho'_2, \ldots, \rho'_s\}$. The required output is a $(\boldsymbol{U}, \boldsymbol{\rho'})$-simultaneous separator, and we compare its cost to an optimal $(\boldsymbol{U}, \boldsymbol{\rho})$-simultaneous separator.

We describe only the undirected case, and the discussion carries over to the directed case if $\rho_i \geq 1/2$, for every $1 \leq i \leq s$.

6.1. A spreading metric for simultaneous separators. The spreading metric for $(\boldsymbol{U}, \boldsymbol{\rho})$-simultaneous separators is obtained by the following linear program:

$$\min \sum_{e \in E} c(e) \cdot d(e)$$

$$\text{s.t. } \forall i, \quad \forall S \subseteq V, \quad \forall v \in S : \sum_{u \in S \cap U_i} \mathsf{dist}(v,u) \cdot w(u) \geq w(S \cap U_i) - \rho_i \cdot w(U_i)$$

$$\forall e \in E : \ 0 \leq d(e) \leq 1.$$

We omit the proofs concerning an optimal spreading metric but state the radius guarantee lemma for this case.

LEMMA 6.1. *If $S \subseteq V$ satisfies $w(S \cap U_i) > \rho_i' \cdot w(U_i)$, then for every vertex $v \in S$*

$$\mathsf{radius}(v, S \cap U_i) > \frac{\rho_i' - \rho_i}{\rho_i'}.$$

The only modification required for finding a simultaneous separator is the definition of $\tilde{r}$. We define $\tilde{r} \triangleq \min_{1 \leq i \leq s} \frac{\rho_i' - \rho_i}{\rho_i'}$. The algorithm proceeds by chopping off parts from the graph using the cut procedure as long as the remaining subgraph does not satisfy the simultaneous separation requirements. The approximation factor obtained is $(1/\tilde{r} + o(1)) \cdot \ln n$. The only difference in the ratio between the capacity of the simultaneous separator and the volume of the spreading metric stems from the definition of $\tilde{r}$.

7. Computing spreading metrics efficiently. Plotkin, Tardos, and Shmoys [16] defined a framework of fractional packing and covering problems and developed fast algorithms for computing approximate solutions therein. Young [18] also devised fast algorithms for finding approximate solutions in this framework. In this section we show how to use this framework to compute spreading metrics efficiently. We note that our results hold only in the case of undirected graphs. We henceforth refer to the algorithm described in [16] as the PST algorithm and to the algorithm described in [18] as the Y algorithm. We first describe the fractional packing setting of [16]. Let $G = (V, E)$ denote a graph, where $|V| = n$ and $|E| = m$. For an optimization problem on graphs, the input consists of the following:

1. A convex set $P \subseteq \mathrm{R}^{\ell}$. (Note that ℓ may be even exponential in n since its influence on the running time is indirect.)
2. A nonnegative linear function $U : P \to \mathrm{R}^m$.

We consider U as m real valued linear functions indexed by the edges of the graph. The goal in the fractional packing problem is to compute λ^* defined by

$$\lambda^* \triangleq \min_{x \in P} \max_{e \in E} U_e(x).$$

Rather than computing λ^* precisely, the PST and Y algorithms do the following. Given an error parameter ε, they find a vector $x' \in P$ such that $U_e(x') \leq (1+\varepsilon)\lambda^*$, for every $e \in E$. Moreover, they also compute a solution to the dual fractional covering problem. This input to the dual problem consists of the following:

1. A convex set $Q \subseteq \mathrm{R}^m$.
2. A nonnegative linear function $U^D : Q \to \mathrm{R}^{\ell}$.

We view each point in Q as an assignment of lengths $d(e)$ to the edges, and we consider U^D as ℓ real valued linear functions. The goal in the fractional covering problem is to compute μ^* defined by

$$\mu^* \triangleq \max_{y \in Q} \min_{1 \le i \le \ell} U_i^D(y).$$

The PST and Y algorithms require the existence of an *optimization oracle*. The input to the optimization oracle is a dual solution which is an assignment of edge lengths $\{d(e)\}_{e \in E}$. The output is a vector $x \in P$ which minimizes the objective function $\sum_{e \in E} U_e(x) \cdot d(e)$. In other words, given a $y \in Q$, the optimization oracle computes an $x \in P$ that minimizes the inner product $y \cdot U(x)$. Both the PST and the Y algorithms still work well even if the optimization oracle returns a suboptimal result. Specifically, if the optimization oracle outputs a vector $x \in P$ for which

$$\sum_{e \in E} U_e(x) \cdot d(e) \le 2 \cdot \min_{x' \in P} \sum_{e \in E} U_e(x') \cdot d(e),$$

then the algorithms find a vector $x' \in P$ such that $U_e(x') \le 2(1+\varepsilon)\lambda^*$, for every $e \in E$.

A key parameter that affects the running time of the PST and Y algorithms is the *width* of the set P with respect to the function U. The width is denoted by $U_{\max}$ and is defined by

$$U_{\max} \triangleq \max_{x \in P} \max_{e \in E} U_e(x).$$

With this setting, the complexities of the PST and Y algorithms are as follows. The complexity of the PST algorithm is

$$O\left(\left(\frac{U_{\max} \cdot \ln(m/\varepsilon) \ln(1/\varepsilon)}{\lambda^* \cdot \varepsilon^2}\right) \cdot (T(\text{compute } U) + T(\text{oracle}))\right),$$

where $T(\text{compute } U)$ denotes the time required to compute $U(x)$ with respect to an intermediate value of x, and $T(\text{oracle})$ denotes the time required by the optimization oracle for answering a single query.

The complexity of the Y algorithm is

$$O\left(\left(\frac{(1+\varepsilon) \cdot U_{\max} \cdot \ln(m)}{\lambda^* \cdot \varepsilon^2}\right) \cdot (T(\text{compute } U) + T(\text{oracle}))\right).$$

Note that ℓ may even be exponentially large without affecting the complexity of the algorithms as long as we can (a) efficiently compute $U(x)$ for the intermediate vectors $x \in P$ that are encountered in the course of running the algorithms, and (b) efficiently compute the oracle.

In the rest of this section we present efficient implementations of the PST and Y algorithms for computing a spreading metric for ρ-separators. We present a deterministic algorithm that has a running time of $\tilde{O}(m^2 n \cdot \frac{\rho'}{\rho' - \rho})$ and a randomized algorithm that has an expected running time of $\tilde{O}(m^2/(\rho' - \rho))$. The following implementation issues are discussed:

- In section 7.1, modification of the linear program (P1) so that it fits the framework of [16], and adjustment of the linear program so that the width of the convex set P is at most $\frac{\rho'}{\rho' - \rho}$.

- In section 7.2, scaling of the problem so that the optimal solution λ^* satisfies $1/\lambda^* = O(m)$.
- In section 7.3, construction of a deterministic optimization oracle that runs in $\tilde{O}(mn)$ time, and a randomized version that succeeds in finding a 2-approximate solution in $\tilde{O}(m/\rho')$ time with high probability.
- In section 7.4, implementation issues such as computing U in $O(m)$ time and deriving a dual solution.

7.1. The modified formulation. In this section we rewrite the linear program (P1$_S$) so that it fits the framework of [16]. This modification will ensure that the width is a constant.

We first define a modified version of (P1$_S$), denoted by (P1′).

$$\text{(P1}') \qquad \min \sum_{e \in E} c(e) \cdot d(e)$$

$$\text{s.t. } \forall S \subseteq V \text{ such that } w(S) > \rho' \cdot w(V), \quad \forall v \in S : \sum_{u \in S} \mathsf{dist}_S(v,u) \cdot w(u) \geq s(w(S))$$

$$\forall e \in E : d(e) \geq 0,$$

where $s(w(S)) = w(S) - \rho \cdot w(V)$.

Observe the following details in (P1′): (a) the constraints that $d(e) \leq 1$ are omitted; (b) the distances $\mathsf{dist}_S(v,u)$ are used; and, most important, (c) we omit the constraints for subsets S whose weight is less than or equal to $\rho' \cdot w(V)$.

Omitting the constraints $d(e) \leq 1$ from (P1$_S$) does not change the cost of an optimal solution. The reason for this is that if $d(e)$ is a feasible solution of (P1$_S$), then so is $d'(e) = \min\{1, d(e)\}$.

It is also easy to show that (i) the cost of an optimal solution of (P1′) is a lower bound on the minimum capacity of a ρ-separator; and (ii) every feasible solution of (P1′) satisfies the radius guarantee as stated in Lemma 3.2. Thus, optimal solutions of (P1′) are also spreading metrics.

We now define a linear program (P2) using the following notation. A *rooted tree* in a graph G is a pair (T, r), where the vertices and edges of the tree T belong to G and r is a designated vertex of T called the root. Removing an edge from T disconnects the tree into two subtrees, where one contains the root r and the other does not contain it. For every edge e in a rooted tree (T, r), define $\omega(T, r, e)$ to be the sum of the weights of the vertices of the subtree of T that are disconnected from the root r by removing edge e. Let $\mathcal{T}$ denote the set of all rooted trees of weight greater than $\rho' \cdot w(V)$.

We define linear program (P2) as follows:

$$\text{(P2)} \qquad \min \sum_{e \in E} c(e) \cdot d(e)$$

$$\text{s.t. for all rooted trees } (T,r) \in \mathcal{T} : \sum_{e \in T} d(e) \cdot \omega(T,r,e) \geq s(w(T))$$

$$\forall e \in E : \; d(e) \geq 0,$$

where $s(w(T)) = w(T) - \rho \cdot w(V)$.

LEMMA 7.1. *The sets of feasible solutions to linear programs* (P1′) *and* (P2) *are equal, and hence their optima are also equal.*

Proof. Consider a rooted tree (T, r), and let $v, u \in T$. Define $\mathsf{dist}_T(v, u)$ to be the distance from v to u in T. It is easy to see that $\sum_{u \in T} \mathsf{dist}_T(r, u) \cdot w(u) = \sum_{e \in T} d(e) \cdot \omega(T, r, e)$. Thus, in (P2) these two expressions are interchangeable.

Given a subset S and a vertex $r \in S$, consider a rooted tree (T, r) of shortest paths in S. For every $u \in S$, it follows that $\mathsf{dist}_S(r, u) = \mathsf{dist}_T(r, u)$. Therefore, every feasible solution of (P2) is also a feasible solution of (P1′).

We now show that every feasible solution of (P1′) is also a feasible solution of (P2). Given a rooted tree (T, r), let S denote the vertex set of T. Let (T', r) denote a shortest-path tree of S. The constraint corresponding to (T', r) is satisfied since it is identical to the constraint corresponding to S and r. Since $\mathsf{dist}_T(r, u) \geq \mathsf{dist}_{T'}(r, u)$, for every vertex $u \in S$, it follows that the constraint corresponding to (T, r) is also satisfied, and the lemma follows. ∎

The *utilization* of edge e by a rooted tree $(T, r) \in \mathcal{T}$ is defined by

$$U_e(T, r) \triangleq \frac{\omega(T, r, e)}{c(e) \cdot s(w(T))}.$$

We define $y(e) \triangleq c(e) \cdot d(e)$ and rewrite the linear program (P2) as follows:

$$\text{(P3)} \qquad \min \sum_{e \in E} y(e)$$

$$\text{s.t. for all rooted trees } (T, r) \in \mathcal{T}\text{: } \sum_{e \in T} y(e) \cdot U_e(T, r) \geq 1$$

$$\forall e \in E : \; y(e) \geq 0.$$

We construct the dual program of (P3) as follows. For every rooted tree $(T, r) \in \mathcal{T}$, define a nonnegative variable $\tilde{x}(T, r)$.

Given a vector $\tilde{x} = \{\tilde{x}(T, r)\}_{(T,r) \in \mathcal{T}}$, define the utilization of edge e by $\tilde{x}$ as follows:

$$U_e(\tilde{x}) \triangleq \sum_{\{(T,r):\; e \in T\}} U_e(T, r) \cdot \tilde{x}(T, r).$$

The dual program of (P3) is defined as follows:

$$\text{(D3)} \qquad \max \sum_{(T,r) \in \mathcal{T}} \tilde{x}(T, r)$$

$$\text{s.t. for all edges } e \in E\text{: } U_e(\tilde{x}) \leq 1$$

$$\text{for all rooted trees } (T, r) \in \mathcal{T}\text{: } \tilde{x}(T, r) \geq 0.$$

We now translate the linear program (D3) into a fractional packing problem (D4) solvable by the PST and Y algorithms. Define a *fractional rooted tree* to be a convex combination of rooted trees. A fractional rooted tree is represented by a nonnegative vector $x = \{x(T, r)\}_{(T,r) \in \mathcal{T}}$ such that $\sum_{(T,r) \in \mathcal{T}} x(T, r) = 1$. Define the convex set $P \subseteq \mathrm{R}^{\ell}$ to be the set of fractional rooted trees, where ℓ denotes the cardinality of $\mathcal{T}$. The fractional packing problem (D4) is to compute an $x^* \in P$ that minimizes $\max_{e \in E} U_e(x)$. The approximate solution $x' \in P$ computed by the

PST algorithm or the Y algorithm satisfies $\max_{e\in E} U_e(x') \leq (1+\varepsilon)\cdot\lambda^*$, where $\lambda^* \triangleq \min_{x\in P}\max_{e\in E} U_e(x)$.

The relation between problems (D3) and (D4) is as follows. Every feasible solution $\tilde{x}$ of (D3) can be scaled to a vector $x \in P$ by setting $x = \tilde{x}/\sum_{T,r}\tilde{x}(T,r)$. The maximum edge utilization of x is bounded by $1/\sum_{T,r}\tilde{x}(T,r)$. Conversely, given a vector $x \in P$ with maximum edge utilization λ, define $\tilde{x} = x/\lambda$ to obtain a feasible solution of (D3) for which $\sum_{T,r}\tilde{x}(T,r) = 1/\lambda$. Thus, every optimal solution of problem (D4) can be scaled to an optimal of (D3) and vice versa. Moreover, this translation by scaling also preserves approximation factors of approximate solutions.

The fractional covering problem (P4) that is dual to (D4) and corresponds to (P3) is defined as follows. Let Q denote the convex set of all vectors $y \in \mathrm{R}^m$ for which $\sum_{e\in E} y(e) = 1$. Define the cost of a rooted tree (T,r) with respect to y by $\mathsf{cost}_y(T,r) \triangleq \sum_{e\in T} y(e)\cdot U_e(T,r)$. Problem (P4) is to find a vector $y \in Q$ that maximizes the cost of a minimum cost tree with respect to y. In other words, define, $\mu^* = \max_{y\in Q}\min_{T,r}\mathsf{cost}_y(T,r)$, and the goal is to find a vector $y \in Q$ for which $\mathsf{cost}_y(T,r) \geq \mu^*$ for all rooted trees (T,r).

To be precise, the PST and Y algorithms find vectors $x \in P$ and $y \in Q$ such that

$$(1+\varepsilon)\cdot\min_{T,r}\mathsf{cost}_y(T,r) \geq \max_{e\in E} U_e(x). \tag{7.1}$$

Combined with linear programming duality, (7.1) implies that the computed costs of the solutions x and y are within a factor of $(1+\varepsilon)$ from the optimum $\lambda^* = \mu^*$. We then scale back the vector y to obtain an approximate solution of linear program (P1′) which constitutes a spreading metric.

We conclude by showing that the width of P with respect to U is constant. Since every point $x \in P$ is a convex combination of rooted trees and since U is linear, it follows that the width is obtained on a single rooted tree.

Claim 7.2. If the capacities of the edges are at least a unit, then the width $U_{\max}$ is at most $\frac{\rho'}{(\rho'-\rho)}$.

Proof. As discussed above,

$$U_{\max} = \max_{(T,r)\in\mathcal{T}}\max_{e\in E} U_e(T,r) = \max_{(T,r,e)}\frac{\omega(T,r,e)}{c(e)\cdot s(w(T))}.$$

Note that $\omega(T,r,e) \leq w(T)$. Since we consider only rooted trees for which $w(T) > \rho'\cdot w(V)$, it follows that $s(w(T)) = w(T) - \rho\cdot w(V) \geq w(T)(1-\rho/\rho')$, and the claim follows. ∎

7.2. Bounding the optimal cost. The running time of the PST and Y algorithms depends on $1/\lambda^*$, where λ^* is the optimal cost of the fractional packing problem (D4). In this section we show how scaling can be used to bound the value of $1/\lambda^*$ by $2m$. Let $\textsc{opt}_\rho$ denote the capacity of an optimal ρ-separator. Since every ρ-separator is a feasible solution to linear program (P1′), it follows that $\textsc{opt}_\rho \geq 1/\lambda^*$. Hence, it suffices to bound $\textsc{opt}_\rho$ by $2m$.

We present a reduction that scales the minimum capacity of a ρ-separator so that it is bounded by $2m$. The cost associated with this reduction is at most $2\cdot\textsc{opt}_\rho$. Namely, the reduction adds edges to the ρ'-separator that we compute, and the sum of the capacities of these edges is at most $2\cdot\textsc{opt}_\rho$. This, of course, suffices for a constant approximation. The overhead of running our reduction is a multiplicative factor of $O(\log m)$, since one needs to execute the PST (or Y) algorithm $O(\log m)$ times.

Suppose that we have a bound t satisfying $\text{OPT}_\rho \le t \le 2\text{OPT}_\rho$. Consider the following scaling. Define

$$\mathsf{Light}_t \triangleq \left\{ e \in E \,:\, c(e) \le \frac{t}{2m} \right\},$$
$$\mathsf{Heavy}_t \triangleq \{ e \in E \,:\, c(e) > t \}.$$

Clearly, edges in Heavy_t do not belong to an optimal ρ-separator. Therefore, contracting edges in Heavy_t does not modify the capacity of an optimal ρ-separator. By contracting we mean that connected components in Heavy_t are merged into a single node, the weight of which equals the sum of the weights of the original vertices, and the capacity of an edge incident to two merged vertices equals the sum of the capacities of the edges connecting the original vertices.

The total capacity of edges in Light_t is bounded by OPT_ρ. Therefore, we can add all the edges in Light_t to our solution, and the overhead of this step is bounded by OPT_ρ. Adding the edges in Light_t to our solution means that we may delete these edges from the graph.

Next, scale the capacities of the remaining edges as follows: $c'(e) = \frac{2m}{t} \cdot c(e)$. Note that the capacities of the remaining edges are between 1 and $2m$.

Let $G' = (V', E')$ denote the graph obtained after contracting edges in Heavy_t, deleting edges in Light_t, and scaling the edge capacities. Let OPT'_ρ denote the capacity of an optimal ρ-separator in G'. The following claims prove the required properties of the reduction from G to G'.

Claim 7.3. The capacity of an optimal ρ-separator in G' satisfies $\text{OPT}'_\rho \le \frac{2m}{t} \cdot \text{OPT}_\rho \le 2m$.

Proof. We show that there is a ρ-separator in G' of capacity at most $2m$. Consider an optimal ρ-separator F in G. Since F is optimal, it does not contain any of the edges in Heavy_t. It follows that if $(u, v) \in F$, $(u, w) \in \mathsf{Heavy}_t$, and $(w, v) \in E$, then $(w, v) \in F$. Therefore, we can transform a ρ-separator F in G into a ρ-separator F' in G' simply by choosing the edges in G' that originate from edges in F. Moreover, $c'(F') \le \frac{2m}{t} \cdot c(F)$. Since $\text{OPT}_\rho \le t$, $\text{OPT}'_\rho \le 2m$. ∎

Claim 7.4. Every $\hat{\rho}$-separator F' in G' can be transformed in linear time into a $\hat{\rho}$-separator F in G such that $c(F) \le \frac{t}{2m} \cdot c'(F') + \text{OPT}_\rho$.

Proof. Every $\hat{\rho}$-separator F' in G' can be transformed into a $\hat{\rho}$-separator in $G - \mathsf{Light}_t$ by choosing the edges in G from which the edges in F' originate. This translation satisfies $c(F) \le \frac{t}{2m} \cdot c'(F)$. Adding the edges of Light_t yields a $\hat{\rho}$-separator in G of cost bounded by $\frac{t}{2m} \cdot c'(F') + \text{OPT}_\rho$. ∎

We conclude from Claims 7.3 and 7.4 that any $\hat{\rho}$-separator F' whose capacity is at most $\alpha \cdot \text{OPT}'_\rho$, for some $\alpha \ge 1$, can be transformed into a $\hat{\rho}$-separator F in G whose capacity is at most $(\alpha + 1) \cdot \text{OPT}_\rho$.

The reduction presented so far enables us to run the approximation algorithms on a graph in which an optimal ρ-separator has capacity at most $2m$. This reduction assumes the existence of 2-approximation for OPT_ρ, which we do not have. Thus, we resort to guessing, starting from $t = 1$ and doubling t for the next guess. Suppose we prescale the graph so that $\text{OPT}_\rho \le m^2$ and all edge capacities remain not less than 1; then only $2 \log m$ guesses are required. For every guess of t, we solve approximately the problem (D4) on the corresponding graph and choose the best solution among the $2 \log m$ solutions. (Note that if our guess of t is "too large," then many edges are contained in Light_t, and Light_t is a ρ-separator. If t is "too small," then Heavy_t spans a subgraph that is too heavy, rendering the problem infeasible.)

We now discuss how to perform prescaling so that $1 \leq \text{OPT}_\rho \leq m^2$. Note that this prescaling keeps all edge capacities not less than 1. After this prescaling, it suffices to consider the "guesses" $1 \leq t = 2^i \leq m^2$ and pick the best solution among the $O(\log m)$ possibilities.

We first order the edges in increasing cost order, namely, $c(e_1) \leq \cdots \leq c(e_m)$. We then compute the index i_0 that satisfies

$$i_0 \triangleq \min\{i \,:\, \{e_1, \ldots, e_i\} \text{ is a } \rho\text{-separator}\}.$$

Note that i_0 can be found using a binary search, and hence it can be computed in $O(m \log m)$ time.

We claim that $c(e_{i_0}) \leq \text{OPT}_\rho \leq i_0 \cdot c(e_{i_0}) \leq m \cdot c(e_{i_0})$. The upper bound follows directly from the definition of i_0. The lower bound follows since if $\text{OPT}_\rho < c(e_{i_0})$, then an optimal ρ-separator would consist of edges in the subset $\{e_1, \ldots, e_{i_0-1}\}$, a contradiction to the definition of i_0.

Following the scaling technique discussed above, define

$$\mathsf{Light} \triangleq \left\{ e \in E \,:\, c(e) \leq \frac{c(e_{i_0})}{m} \right\},$$
$$\mathsf{Heavy} \triangleq \{e \in E \,:\, c(e) > m \cdot c(e_{i_0})\}.$$

Since $c(e_{i_0}) < \text{OPT}_\rho$, the sum of the capacities of the edges in Light is bounded by OPT_ρ, and we add the edges of Light to our solution and delete them from the graph, as before. Since $\text{OPT}_\rho < m \cdot c(e_{i_0})$, the capacity of each edge in Heavy is more than OPT_ρ. Therefore, the edges in Heavy do not belong to an optimal ρ-separator and they can be contracted.

Next, scale the capacities of the remaining edges as follows: $c'(e) = \frac{m}{c(e_{i_0})} \cdot c(e)$. It is easy to see that this achieves the desired prescaling.

7.3. The optimization oracle. The optimization oracle receives as input nonnegative edge lengths $\{d(e)\}_e$ and finds a fractional rooted tree $\tilde{x} \in P$ that minimizes $\sum_{e \in E} d(e) \cdot U_e(\tilde{x})$. Such a vector is called a *min-cost fractional rooted tree* since one may interpret the edge lengths as edge costs.

Claim 7.5. Let $\tilde{x} \in P$ be a min-cost fractional rooted tree. Then, for every rooted tree $(T, r) \in \mathcal{T}$ for which $\tilde{x}(T, r) > 0$,

$$\sum_{e \in (T,r)} d(e) \cdot U_e(T, r) = \sum_{e \in E} d(e) \cdot U_e(\tilde{x}).$$

Proof. Observe that every rooted tree is also a fractional rooted tree. Hence, if there exists a rooted tree (T', r') for which $\tilde{x}(T', r') > 0$ and

$$\sum_{e \in (T',r')} d(e) \cdot U_e(T', r') > \sum_{e \in E} d(e) \cdot U_e(\tilde{x}),$$

then there also exists a rooted tree (T'', r'') for which $\tilde{x}(T'', r'') > 0$ and

$$\sum_{e \in (T'',r'')} d(e) \cdot U_e(T'', r'') < \sum_{e \in E} d(e) \cdot U_e(\tilde{x}),$$

yielding a contradiction to the assumption that $\tilde{x}$ is a min-cost fractional tree. The claim follows. ∎

It follows from Claim 7.5 that in order to compute a min-cost vector, it suffices to find a min-cost tree. Namely, our goal is to find a rooted tree $(T, r) \in \mathcal{T}$ that minimizes

$$\text{cost}(T, r) \triangleq \sum_{e \in (T,r)} \frac{\omega(T, r, e) \cdot d(e)}{c(e) \cdot s(w(T))}.$$

We prove that finding a minimum-cost rooted tree in the case where the vertices have arbitrary weights is NP-complete by a reduction from the SUBSET-SUM problem [5, p. 223].

The input to an instance of the SUBSET-SUM problem consists of a fraction $\alpha \in (0, 1)$ and n positive weights $w_1, w_2, \ldots, w_n$ that satisfy $\sum_{i=1}^{n} w_i = 1$. The question is whether there exists a subset of indices $I \subseteq [1..n]$ such that $\sum_{i \in I} w_i = \alpha$.

The reduction to the weighted optimization oracle is as follows. We choose any pair $\rho' > \rho$ such that $\rho' - \rho = \alpha$. For simplicity, we choose $\rho = (1 - \alpha)/2$ and $\rho' = (1 + \alpha)/2$. Consider a star graph with a center vertex r and n vertices connected to the center, denoted by $v_1, v_2, \ldots, v_n$. Define the edge capacities and the edge lengths all to be 1, namely, $d(r, v_i) = c(r, v_i) = 1$ for all $1 \le i \le n$. Define the vertex weights as follows: $w(v_i) = w_i$ for all $1 \le i \le n$, and $w(r) = \rho/(1 - \rho')$ (which in our case means $w(r) = 1$).

Claim 7.6. The cost of a minimum-cost rooted tree (T^*, r^*) in the instance of the weighted optimization oracle is $1/2$ if and only if the answer to the instance of the SUBSET-SUM problem is affirmative.

Proof. It can readily be verified that since $w(r)$ is relatively large, a minimum-cost tree must be rooted at r. Due to the unit edge lengths and edge capacities, the cost of a rooted tree (T, r) equals

$$\text{cost}(T, r) = \frac{w(T) - w(r)}{w(T) - \rho(1 + w(r))} = \frac{w(T) - 1}{w(T) - 2\rho}.$$

Since $w(r) > \rho(1 + w(r))$, it follows that the cost is minimized when $w(T)$ is minimized, namely, when $w(T) = \rho'(1 + w(r))$. If there exists a tree (T, r) such that $w(T) = \rho'(1 + w(r))$, then its cost is the smallest possible and $\text{cost}(T, r) = 1/2$.

However, there exists such a tree if and only if there exists a subset $I \subseteq [1..n]$ such that

$$w(r) + \sum_{i \in I} w_i = \rho'(1 + w(r)).$$

That occurs if and only if $\sum_{i \in I} w_i = \alpha$, and the claim follows. ∎

The rest of this section is organized as follows:

1. An algorithm for computing the optimization oracle in the case of unit vertex weights is presented. The running time of this algorithm is $\tilde{O}(mn)$.
2. A $(2 + \varepsilon)$-approximation algorithm for the optimization oracle in the case of arbitrary vertex weights is presented. The running time of this algorithm is $\tilde{O}(mn)$.
3. A randomized approximation algorithm for the optimization oracle (for both the weighted and unweighted case) is presented for reducing the running time of the optimization oracle. The randomized algorithm reduces the running time by a factor of $\tilde{O}(n \cdot \rho')$.

7.3.1. Precise optimization oracle for the unit-weight case. We consider here the case where all vertices have unit weight. Define a *k-vertex min-cost tree rooted at* r, denoted by $T(r,k)$, to be a rooted tree whose cost is minimum among the rooted trees (T,r) for which $|T|=k$. We find a min-cost tree by computing, for every root r and for every $1 \le k \le n$, a k-vertex min-cost tree rooted at r. The min-cost tree is then chosen among these n^2 trees. We henceforth focus on computing $T(r,k)$ for a fixed root r. Define

$$\gamma(T,r) \triangleq \mathrm{cost}(T,r) \cdot s(|T|) = \sum_{e \in T} \frac{d(e)}{c(e)} \cdot \omega(T,r,e).$$

Let $d'(e) \triangleq d(e)/c(e)$. Given a rooted tree (T,r), let $\mathsf{dist}'_T(r,u)$ denote the distance in T from r to u with respect to edge lengths $d'(e)$. Clearly,

$$\gamma(T,r) = \sum_{u \in T} \mathsf{dist}'_T(r,u). \tag{7.2}$$

The consequence of (7.2) is that computing $T(r,k)$ reduces to that of computing the k closest vertices to r with respect to edge lengths $d'(e)$. Hence, for a given root r we can compute $T(r,k)$ for all values of k by running Dijkstra's shortest paths algorithm which has complexity $O(m+n\log n)$ [4]. Since we need to run the algorithm for all choices for the root r, we get that the total running time of the min-cost subroutine is $O(mn+n^2\log n)$.

7.3.2. Approximate optimization oracle for the weighted case. First, we extend the definition of $\gamma(T,r)$ from the unweighted case to deal with vertex weights. Define

$$\gamma(T,r) \triangleq \mathrm{cost}(T,r) \cdot s(w(T)) = \sum_{e \in T} \frac{d(e)}{c(e)} \cdot \omega(T,r,e) = d'(e) \cdot \omega(T,r,e).$$

Clearly,

$$\gamma(T,r) = \sum_{u \in T} \mathsf{dist}'_T(r,u) \cdot w(u). \tag{7.3}$$

The optimization oracle in the weighted case searches for a rooted tree $(T,r) \in \mathcal{T}$ that minimizes $\gamma(T,r)/s(w(T))$. The approximate oracle "guesses" an interval for the denominator and searches for rooted trees that minimize the numerator.

Figure 7.1 depicts a $2(1+\varepsilon)$-approximation algorithm for the optimization oracle. Algorithm *apx-opt-oracle* is based on a procedure called apx-min-γ which, given a root r and a threshold t, finds a rooted tree (T,r) of weight at least t such that $\gamma(T,r) \le 2 \cdot \min_{\{(T',r'):w(T') \ge t\}} \gamma(T',r')$. Algorithm apx-opt-oracle calls procedure apx-min-γ from every possible root with a sequence of increasing thresholds and returns a rooted tree of minimum cost among all the computed trees. The following claim summarizes the approximation factor of algorithm apx-opt-oracle.

Claim 7.7. Algorithm apx-opt-oracle finds a $2(1+\varepsilon)$-approximate min-cost rooted tree.

Proof. Consider a min-cost rooted tree (T^*,r^*). Let $t_0,t_1,\ldots$ denote the sequence of thresholds considered by the algorithm. Suppose that $t_i \le w(T^*) < t_{i+1}$. Let (T,r^*) denote the rooted tree that is found by procedure apx-min-γ with respect to threshold t_i and root r^*.

apx-opt-oracle $(V, E, \{w(v)\}_{v\in V}, \{d'(e)\}_{e\in E})$
(assume that ρ, ρ' are known)

Set $(\tilde{T}, \tilde{r})$ to be any rooted tree in $\mathcal{T}$.
$t = \rho' \cdot w(V)$ (set initial threshold to be $\rho' \cdot w(V)$)
while $t \le w(V)$ do
begin
for every root $r \in V$ do
begin
$T =$ *apx-min-*γ $(V, E, \{w(v)\}_{v\in V}, \{d'(e)\}_{e\in E}, r, t)$
if $\frac{\gamma(T,r)}{s(w(T))} < \frac{\gamma(\tilde{T},\tilde{r})}{s(w(\tilde{T}))}$ then $\{\ \tilde{T} = T$ and $\tilde{r} = r\}$
end
$t = (t - \rho \cdot w(V))(1 + \varepsilon) + \rho \cdot w(V)$
end
Return $(\tilde{T}, \tilde{r})$

FIG. 7.1. $2(1+\varepsilon)$-*approximation for min-cost rooted tree.*

The 2-approximation of procedure apx-min-γ implies that $\gamma(T, r^*) \le 2\gamma(T^*, r^*)$. Also, the fact that $w(T^*) < t_{i+1}$ implies that

$$\frac{w(T^*) - \rho \cdot w(V)}{w(T) - \rho \cdot w(V)} < \frac{t_{i+1} - \rho \cdot w(V)}{t_i - \rho \cdot w(V)} = 1 + \varepsilon.$$

Therefore, $\text{cost}(T, r^*) \le 2(1+\varepsilon) \cdot \text{cost}(T^*, r^*)$, and since algorithm apx-opt-oracle returns a rooted tree the cost of which is not greater than $\text{cost}(T, r^*)$, the claim follows. □

We now turn to the procedure apx-min-γ depicted in Figure 7.2. Recall that this procedure finds a rooted tree (T, r) of weight at least t such that $\gamma(T, r) \le 2 \cdot \min_{\{(T',r'):w(T')\ge t\}} \gamma(T', r')$. Note that it is computationally prohibitive to find the exact minimum since for any fixed $t > \rho' \cdot w(V)$, searching for a tree rooted at r of weight at least t that minimizes $\gamma(T, r)$ is NP-hard. The reduction is again from the SUBSET-SUM problem and is analogous to the reduction given above to the problem of searching for a tree rooted at r of weight at least $\rho' \cdot w(V)$ that minimizes $\gamma(T, r)$.

The procedure apx-min-γ grows a shortest-path tree rooted at r. Whenever the total weight of the vertices in the tree exceeds the threshold t, the resulting tree is considered a candidate for a min-cost tree, and the last vertex added to the tree is deleted from the tree. The procedure returns the minimum cost tree among all candidate trees. The following claim summarizes the approximation factor that it obtains.

Claim 7.8. Algorithm apx-min-γ computes a 2-approximate min-cost tree of weight at least t rooted at vertex r.

Proof. Let T^* denote a min-cost tree rooted at r of weight at least t. Given a tree T in *List*, let $last(T)$ denote the last vertex added to T. Consider the first tree T in *List* such that $last(T) \in T^*$. We show that the cost of T is at most twice the cost of T^*.

First, we prove that such a tree T exists. Assume to the contrary that for all the trees T' in *List*, $last(T') \notin T^*$. Since a vertex is added to X only if it is $Last(T')$ for some T' in *List*, we get that $X \cap T^*$ is empty upon termination. Let T_f denote

apx-min-γ $(V, E, \{w(v)\}_{v\in V}, \{d'(e)\}_{e\in E}, r, t)$

```
List = empty list   (list of candidate trees)
X = φ               (set of deleted vertices)
T = {r}             (current tree)
repeat
    let v be a closest vertex to r in the subgraph induced by V − X,
        among all vertices in V − T − X.
    let e = (u, v) be the last edge in a shortest path from r to v
        in the subgraph induced by V − X.          (u ∈ T)
    if w(T) + w(v) ≥ t, then add T ∪ {e} to List and add v to X.
    else add v to T.
until V − T − X is empty
Return min-cost tree in List
```

FIG. 7.2. 2-*approximation for min-cost tree of weight at least* t *rooted at* r.

the last tree that is added to *List*. Since $X \cap T^*$ is empty, it follows that $T^* \subseteq T_f$. Consider the vertex v of T^* that is added last to T_f. After v is added, the weight of the current tree (which is a subtree of T_f) is at least t. Hence, v should be added to X, which is a contradiction.

Consider the following partitioning of $T \cup T^*$ into three sets:

$$A = T - T^*,$$
$$B = (T \cap T^*) - last(T),$$
$$C = (T^* - T) + last(T).$$

We partition T into two: $T - last(T)$ and $last(T)$. Since $last(T) \in T^*$, the cost of $last(T)$ is not greater than the cost of T^*. We show that the cost of $T - last(T)$ is not greater than the cost of T^* as well.

By the definition of T, when T is constructed (before $last(T)$ is added to X), set X does not contain any vertices of T^*. The algorithm adds vertices to T in nondescending order of distance, where the distance is measured in the subgraphs induced by $V - X$. Since $X \cap T = \emptyset$ the distances in these subgraphs are the same as the distances in the whole graph. Therefore, the vertices of A are closer (or at least as close) to r than any vertex in C. Namely, for every $v \in A$, $\mathsf{dist}_V(r, v) \leq \mathsf{dist}_V(r, u)$ for all $u \in C$.

By the definition of T, $w(A) + w(B) < t$. On the other hand, since $T^* = B \cup C$, it follows that $w(B) + w(C) \geq t$. Hence, $w(A) < w(C)$. Together with the fact that the vertices in A are closer to r than any vertex in C, we obtain that the cost of A is smaller than the cost of C, and hence the cost of $T - last(T) = A \cup B$ is smaller than the cost of T^*, and the claim follows. □

The complexity of algorithm apx-min-γ is dominated by the complexity of Dijkstra's shortest paths algorithm which has complexity $O(m+n \log n)$ [4]. The number of iterations in algorithm apx-opt-oracle, for every root, is bounded by $\log(1/\rho')/\log(1+\varepsilon)$ which is at most $O(1/(\rho' \cdot \varepsilon))$. Hence, the total complexity of computing a $2(1+\varepsilon)$-approximate min-cost tree is $\tilde{O}(mn)$.

7.3.3. A randomized algorithm. Consider a tree T, and let c be a root such that the cost (T, c) is minimum among the costs of all rooted trees (T, r), where $r \in T$. The following claim shows that if a root r of T is chosen at random with probability $w(r)/w(T)$, then the expected cost of (T, r) is at most twice the cost of (T, c). Denote the cost of a rooted tree (T, r) by $\mathsf{cost}(T, r)$ and the expected cost by $\mathbf{Exp}[\mathsf{cost}(T, r)]$.

Claim 7.9. If a root r is chosen with probability $w(r)/w(T)$, then $\mathbf{Exp}[\mathsf{cost}(T, r)] \leq 2 \cdot \mathsf{cost}(T, c)$.

Proof. The cost of a tree (T, r) is defined to be

$$\sum_{e \in T} \frac{\omega(T, r, e) \cdot d(e)}{c(e) \cdot s(w(T))}.$$

Suppose that edge e disconnects T into two subtrees, T_1 and T_2, such that $w(T_1) \leq w(T_2)$. Clearly, $\omega(T, c, e) \geq w(T_1)$. On the other hand,

$$\mathbf{Exp}[\omega(T, r, e)] = \frac{2w(T_1) \cdot w(T_2)}{w(T)},$$

yielding the bound $\mathbf{Exp}[\omega(T, r, e)] \leq 2 \cdot \omega(T, c, e)$. Hence, by linearity of expectation,

$$\begin{aligned}\mathbf{Exp}[\mathsf{cost}(T, r)] &= \mathbf{Exp}\left[\sum_{e \in T} \frac{\omega(T, r, e) \cdot d(e)}{c(e) \cdot s(w(T))}\right] = \sum_{e \in T} \frac{\mathbf{Exp}[\omega(T, r, e)] \cdot d(e)}{c(e) \cdot s(w(T))} \\ &\leq \sum_{e \in T} \frac{2 \cdot \omega(T, c, e) \cdot d(e)}{c(e) \cdot s(w(T))} = 2 \cdot \mathsf{cost}(T, c),\end{aligned}$$

completing the proof. ∎

Combining the last claim with the algorithm for computing a min-cost tree rooted at vertex r yields the following randomized algorithm for finding a 2-approximate min-cost tree. The algorithm chooses a vertex $v \in V$ with probability $w(v)/w(V)$ and outputs a min-cost tree T rooted at v. This saves the need to compute min-cost rooted trees for every possible root. Fix an optimal min-cost rooted tree (T^*, r^*). Since $w(T^*) \geq \rho' w(V)$, the probability that $v \in T^*$ is at least ρ', in which case the expected cost of (T, v) is at most twice the cost of (T^*, r^*). With probability bounded by $1 - \rho'$, we fail. To amplify the probability of success, we may need to repeat the above algorithm $\tilde{O}(1/\rho')$ times, yielding that the complexity of the randomized algorithm is $\tilde{O}(m/\rho')$.

7.4. Implementation issues. We first point out how to compute $U(x)$ on the intermediate values of x in $O(m)$ time. The main reason this is possible is that the optimization oracle returns a single rooted tree (T, r). Therefore, the edge utilizations with respect to this rooted tree need to be computed. In the Y algorithm this is all we need, and in the PST algorithm, the new vector x is a weighted average of the old one and the rooted tree returned by the optimization oracle. Since the function U is linear, we simply take the weighted average of the edge utilizations.

Second, we discuss how to scale the vector $y \in Q$ returned by the PST and Y algorithms to obtain a spreading metric. The scaling consists of the following stages:

1. Transform y into a solution y' of (P3) by setting $y'(e) = \frac{y(e)}{\min_{T,r} \mathsf{cost}_y(T,r)}$. Note that in the weighted case, the denominator is only approximated, which adds another error to the approximation factor. (For the deterministic oracle this error is bounded by $2(1 + \varepsilon)$, and for the randomized oracle this error is bounded by $4(1 + \varepsilon)$.)
2. Transform y' into a solution d of (P2) by setting $d(e) = y'(e)/c(e)$.

Acknowledgments. We thank Serge Plotkin and Éva Tardos for their generous help in applying the framework described in [16] for approximating spreading metrics. We thank Naveen Garg for discussions about the constants in previous related approximation algorithms. We thank Neal Young for helpful remarks about his paper [18]. We thank Mark Rosenschein for pointing out the need for a different oracle in the weighted case.

REFERENCES

[1] J. W. Beery and M. K. Goldberg, *Path optimization for graph partitioning problems*, Discrete Appl. Math., 90 (1999), pp. 27–50.

[2] G. Even, J. Naor, B. Schieber, and M. Sudan, *Approximating minimum feedback sets and multicuts in directed graphs*, Algorithmica, 20 (1998), pp. 151–174.

[3] G. Even, J. Naor, S. Rao, and B. Schieber, *Divide-and-conquer approximation algorithms via spreading metrics*, in Proceedings 36th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Piscataway, NJ, 1995, pp. 62–71.

[4] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM, 34 (1987), pp. 596–615.

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

[6] N. Garg, V. V. Vazirani, and M. Yannakakis, *Approximate max-flow min-(multi)cut theorems and their applications*, SIAM J. Comput., 25 (1996), pp. 235–251.

[7] D. S. Johnson, C. R. Aragon, L. A. McGoech, and C. Schevon, *Optimization by simulated annealing: Part* I, *graph partitioning*, Oper. Res., 37 (1989), pp. 865–892.

[8] B. W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell System Tech. J., 49 (1970), pp. 291–307.

[9] T. Kloks, *Treewidth*, Lecture Notes in Comput. Sci. 842, Springer-Verlag, Berlin, Heidelberg, 1994.

[10] P. N. Klein, S. A. Plotkin, S. Rao, and É. Tardos, *Approximation algorithms for Steiner and directed mutlicuts*, J. Algorithms, 22 (1997), pp. 241–269.

[11] T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas, *Fast approximation algorithms for multicommodity flow problems*, J. Comput. System Sci., 50 (1995), pp. 228–243.

[12] T. Leighton, F. Makedon, and S. Tragoudas, *Approximation algorithms for VLSI partition problems*, in Proceedings 1990 IEEE International Symposium on Circuits and Systems, (ISCAS '90), New Orleans, LA, Vol. 4, IEEE Computer Society Press, Piscataway, NJ, 1990, pp. 2865–2868.

[13] T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms*, in Proceedings 29th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Piscataway, NJ, 1988, pp. 422–431; J. ACM, to appear.

[14] K. Lang and S. Rao, *Finding near-optimal cuts: An empirical evaluation*, in Proceedings 4th Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, TX, 1993, SIAM, Philadelphia, 1993, pp. 212–221.

[15] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming*, SIAM Stud. Appl. Math. 13, SIAM, Philadelphia, 1994.

[16] S. Plotkin, D. Shmoys, and É. Tardos, *Fast approximation algorithms for fractional packing and covering problems*, Math. Oper. Res., 20 (1995), pp. 257–301.

[17] H. D. Simon and S.-H. Teng, *How good is recursive bisection?*, SIAM J. Sci. Comput., 18 (1997), pp. 1436–1445.

[18] N. Young, *Randomized rounding without solving the linear program*, in Proceedings 6th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 1995, SIAM, Philadelphia, 1995, pp. 170–178.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2215–2256

AN OPTIMAL ALGORITHM FOR EUCLIDEAN SHORTEST PATHS IN THE PLANE *

JOHN HERSHBERGER† AND SUBHASH SURI‡

Abstract. We propose an optimal-time algorithm for a classical problem in plane computational geometry: computing a shortest path between two points in the presence of polygonal obstacles. Our algorithm runs in worst-case time $O(n \log n)$ and requires $O(n \log n)$ space, where n is the total number of vertices in the obstacle polygons. The algorithm is based on an efficient implementation of wavefront propagation among polygonal obstacles, and it actually computes a planar map encoding shortest paths from a fixed source point to all other points of the plane; the map can be used to answer single-source shortest path queries in $O(\log n)$ time. The time complexity of our algorithm is a significant improvement over all previously published results on the shortest path problem. Finally, we also discuss extensions to more general shortest path problems, involving nonpoint and multiple sources.

Key words. shortest path, shortest path map, Euclidean distance, obstacle avoidance, quadtree, planar subdivision, weighted distance

AMS subject classifications. 68Q25, 68Q20, 68P05, 51-04

PII. S0097539795289604

1. Introduction.

1.1. The background and our result.

The Euclidean shortest path problem is one of the oldest and best-known problems in computational geometry. Given a planar set of polygonal obstacles with disjoint interiors, the problem is to compute a shortest path between two points avoiding all the obstacles. Due to its simple formulation and obvious applications in routing and robotics, the problem has drawn the attention of many researchers in computational geometry; we mention only a few papers most relevant to our work [4, 13, 14, 17, 18, 19, 24].

The problem of computing shortest paths in the presence of a single obstacle has received special attention, due to its applications in various geometric problems involving a simple polygon [4, 13, 14, 17]. The roles of free space and obstacle space have traditionally been reversed in this special case: the interior of the polygon represents the *free space* and the boundary of the polygon represents an impenetrable obstacle. After several years of continued efforts, an optimal, linear-time algorithm is now known for computing a shortest path in a simple polygon [13, 14].

The general case of multiple obstacles, however, has proved to be substantially more difficult. There have been two fundamentally different approaches to the problem: the *visibility graph method* and the *shortest path map method*.[1] The visibility graph method is based on constructing a graph whose nodes are the vertices of the

*Received by the editors July 31, 1995; accepted for publication (in revised form) December 4, 1997; published electronically July 9, 1999. A preliminary version of this paper appeared in the *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, 1993, pp. 508–517. The authors were at DEC Systems Research Center, Palo Alto, CA, and Bellcore, Morristown, NJ, respectively, when this research was conducted.

http://www.siam.org/journals/sicomp/28-6/28960.html

†Mentor Graphics Corporation, 8005 SW Boeckman Rd., Wilsonville, OR 97070 (john_hershberger@mentor.com).

‡Department of Computer Science, Washington University, St. Louis, MO 63130 (suri@cs.wustl.edu).

[1] Several authors have also considered approximation algorithms for the shortest path problem [5, 7]; we consider only the exact shortest path problem.

obstacles and whose edges are pairs of mutually visible vertices. The shortest path between two vertices can be found by running any Dijkstra-type algorithm on this graph [8, 9, 11]. This approach fueled intense research on computing visibility graphs, culminating in an optimal $O(n \log n + E)$ time algorithm by Ghosh and Mount [12], where E is the number of edges in the graph. Unfortunately, the visibility graph can have $\Omega(n^2)$ edges in the worst case, and so any shortest path algorithm that depends on an explicit construction of the visibility graph will have a similar worst-case running time [1, 2, 15, 21, 24]. A "holy grail" of this approach is to build and search only the portion of the visibility graph that is relevant to the shortest path computation, but no noteworthy progress has been made on that front.

The second approach tries to solve a more general problem: for a given source point s, build a shortest path map (a subdivision of the plane) so that all points of a region have the same vertex sequence in their shortest path to s. This map is an encoding of shortest paths from s to *all* points of the plane. The shortest path map approach seems inherently more geometric than the graph-theoretic method based on visibility graphs. Nevertheless, most algorithms using the shortest path map approach also have $\Omega(n^2)$ worst-case running times; however, their running times typically have the form $O(n\,k\,g(n))$, where k is the *number* of obstacles and $g(n)$ is a sublinear function, such as the poly-logarithm [15, 18, 23]. Thus, for a small number of obstacles, these bounds approach the time complexity for a single obstacle. Mitchell has recently published an algorithm for computing a shortest path map that runs in $O(n^{3/2+\epsilon})$ time and space [19], for any $\epsilon > 0$, with the constant in the big-Oh notation depending on ϵ. Mitchell's algorithm uses some advanced range searching data structures to compute the vertices of the shortest path map.

The only lower bound known for the shortest path problem is $\Omega(n \log n)$ in the algebraic computation tree model, and so there remained a relatively large gap between the known upper and lower bounds on the problem. (The lower bound follows easily by a reduction from sorting.) Nevertheless, there had been a general belief in the computational geometry community that an almost-linear-time algorithm must be achievable.

In this paper, we validate this belief by presenting an optimal $O(n \log n)$ time algorithm for computing shortest paths in the presence of polygonal obstacles; n denotes the total number of vertices in all the obstacle polygons. Our algorithm takes the shortest path map approach and builds a subdivision of the plane, which after an additional linear-time preprocessing can be used to answer shortest path queries from a fixed point [10, 16].

A key idea in our algorithm is a special, quad-tree-style subdivision of the plane with respect to an arbitrary set of points P. This subdivision, called a *conforming subdivision*, divides the plane into a linear number of cells using horizontal and vertical edges so that the following critical condition holds: each point of P lies in a separate cell, and there are $O(1)$ cells within distance $\alpha|e|$ of every subdivision edge e, where $|e|$ is the length of e and α is a parameter (we choose $\alpha = 2$ for our application). Though a subdivision into square cells with this property can be obtained using a quad-tree construction of Bern, Eppstein, and Gilbert [3], that subdivision has size $O(n \log A)$, where A is the aspect ratio of the Delaunay triangulation of P. Our subdivision achieves its linear upper bound by enforcing a weaker condition; in particular, cells in our subdivision may be nonconvex and the subdivision itself may not be connected. Nevertheless, our conforming subdivision appears to be a useful tool and is likely to have other applications. In particular, we discuss extensions of our

technique that can handle generalized versions of the shortest path problem. These include versions with multiple sources (the "geodesic Voronoi diagram") or nonpoint sources such as line segments or disks.

1.2. An overview of the algorithm. We use a technique dubbed the *continuous Dijkstra method* in the literature [18, 19, 20]. It simulates the expansion of a wavefront from a point source in the presence of polygonal obstacles. The wavefront at time t consists of all points of the plane whose shortest path distance to the source is t. The boundary of the wavefront is a set of cycles, each composed of a sequence of circular arcs. Each arc, called a *wavelet*, is generated by an obstacle vertex already covered by the wavefront; the vertex is called the *generator* of its wavelet. The meeting point between two adjacent wavelets sweeps along a bisector curve, which is either a straight line or a hyperbola. Simulating the wavefront requires processing *events* that change its topology. These events fall into two categories: wavefront-wavefront collisions and wavefront-obstacle collisions. The ability to process these events efficiently is the key to a fast algorithm for the shortest path problem. Detecting and processing these events quickly, however, appears to be quite difficult, and except for the recent result of Mitchell [19], all previous algorithms employing the continuous Dijkstra method have led to no better than an $\Omega(n^2)$ worst-case time bound.

We introduce two new ideas to speed up the implementation of the wavefront propagation method: a quad-tree-style subdivision of the plane, and an approximate wavefront. Our first idea is to recognize that advancing a wavefront from event to event can be difficult without a sufficiently well-behaved subdivision of the plane to guide the propagation. We build a special subdivision of size $O(n)$ on the vertices of the obstacles, temporarily ignoring the line segments between them. Each cell of this subdivision, called a *conforming subdivision*, has a constant number of straight line edges, contains at most one obstacle vertex, and satisfies the following crucial property: for any edge e of the subdivision, there are $O(1)$ cells within distance $2|e|$ of e. We then insert the obstacle line segments into the subdivision, but maintain both the linear size of the subdivision and its conforming property—except now a *nonobstacle* edge e has the property that there are $O(1)$ cells within *shortest path* distance $2|e|$ of the edge. These cells form the units of our propagation algorithm: in each step, we advance the wavefront through one cell. Since each cell has constant descriptive complexity, we are able to do the propagation in a cell efficiently.

Inside a cell, a wavefront-obstacle event is relatively easy to handle. However, a wavefront-wavefront event is more complex. There are two types of wavefront-wavefront events, depending on whether or not the colliding wavelets are neighbors in the wavefront. The collision of neighboring wavelets occurs when a wavelet is engulfed by the expanding wavelets of its two neighbors. This event is easy to detect and process. The collisions between nonneighboring wavelets, however, are more troublesome, and to process them we introduce our second idea—the approximate wavefront.

When trying to propagate the wavefront across a boundary edge of a cell, we abandon the idea of computing the wavefront exactly; instead, we maintain two separate wavefronts approaching the edge from opposite sides. Each of these wavefronts is an *approximate wavefront*, representing the wavefront that hits the edge from only one side.

We use timers to make a conservative estimate of the time each edge is engulfed by the wavefront, and discard any parts of the wavefront arriving at a cell boundary after a timer at that boundary edge goes off. A critical task of these timers is to ensure that the wavefront-wavefront collisions of the true shortest path map are

detected during approximate wavefront propagation in a small neighborhood of their actual location. The algorithm propagates the approximate wavefront, remembering the wavefront-wavefront collisions and updating the wavefront so that it has enough information to act as an approximate wavefront at any time.

At the end of the propagation phase, we collect all the collision information, then use Voronoi diagram techniques in each cell to compute the collision events in that cell precisely. The collisions determine the edges of the final shortest path map.

This paper contains seven sections. Section 2 describes our conforming subdivision of the free space, and section 6 gives the details of its construction. Section 3 presents the key shortest path properties used by our algorithm. Section 4 describes our algorithm for computing a shortest path map. The data structures and finer details of our algorithm are discussed in section 5. We close in section 7 with some discussion and open problems.

2. A conforming subdivision of the free space. The input to our shortest path problem is a source vertex s and a family of obstacles $\mathcal{O} = \{O_1, O_2, \ldots, O_k\}$, where each obstacle is a simple polygon and the closures of any two obstacles are disjoint. (It is not hard to extend our algorithm to handle more general polygonal obstacles, but for convenience we limit our discussion to disjoint, nonnested obstacles.) The total number of vertices in all the obstacles is n. The plane minus the interiors of all obstacle polygons is called the *free space*, and a path is called *legal* if it lies entirely in the free space—that is, a legal path is disjoint from the interiors of all obstacle polygons in the family $\mathcal{O}$. Given two points in the plane, a *Euclidean shortest path* between them is a legal path of minimum total length connecting the two points.

A key ingredient of our shortest path algorithm is a special subdivision of the plane into *cells* of constant descriptive complexity. We construct this subdivision in two steps: the first step builds a subdivision by considering only the vertices of the obstacle polygons; the second step inserts the obstacle edges into the subdivision. Our algorithm for the first step (constructing a conforming subdivision for points) is somewhat complicated and quite independent of our main topic, the shortest paths, and so we have moved its presentation to section 6 at the end of the paper. In the present section, we assume the construction for points, and describe how to modify this subdivision when obstacle edges are inserted. We start with some preliminary definitions.

2.1. The well-covering regions. Our subdivision is inspired by quad-trees, though it is best implemented bottom-up. A crucial property of our subdivision is the *well-covering* of its internal edges. Given a straight-line subdivision $\mathcal{S}$ of the plane, an edge $e \in \mathcal{S}$ is said to be *well-covered with parameter* α if the following three conditions hold:

(W1) There exists a set of cells $\mathcal{C}(e) \subseteq \mathcal{S}$ such that e lies in the interior of their union. The union is denoted $\mathcal{U}(e) = \{c \mid c \in \mathcal{C}(e)\}$.

(W2) The total complexity of all the cells in $\mathcal{C}(e)$ is $O(\alpha)$.

(W3) If f is an edge on the boundary of the union $\mathcal{U}(e)$, then the Euclidean distance between e and f is at least $\alpha \cdot \max(|e|, |f|)$.

The edge is *strongly* well-covered if the stronger condition (W3′) holds:

(W3′) If f is an edge on *or outside* the boundary of the union $\mathcal{U}(e)$, then the Euclidean distance between e and f is at least $\alpha \cdot \max(|e|, |f|)$.

In either case, the region $\mathcal{U}(e)$ is called the *well-covering region of* e. Our wavefront simulation algorithm cares only about the distance between e and the edges on the boundary of $\mathcal{U}(e)$; that is, it requires its subdivision edges to be well-covered, but

not strongly well-covered. The strong condition on the distance between e and the edges outside $\mathcal{U}(e)$ is used only in our construction of the conforming subdivision (cf. Lemma 2.2).

Let V denote the set of vertices of the obstacle polygons, plus the source vertex s. A subdivision $\mathcal{S}$ is called a (*strong*) *α-conforming subdivision for V* if

(C1) Each cell of $\mathcal{S}$ contains at most one point of V in its closure (interior plus boundary),

(C2) Each edge of $\mathcal{S}$ is (strongly) well-covered with parameter α, and

(C3) The well-covering region of every edge of $\mathcal{S}$ contains at most one vertex of V.

The subdivision is called "conforming" because conditions (C1) and (C3) force it to conform to the distribution of points in V. Figure 2.1 shows an example of a well-covering region in a 1-conforming subdivision. The region $\mathcal{U}(e)$, drawn shaded, is not necessarily a minimal well-covering region; rather, it is the region constructed by our algorithm.

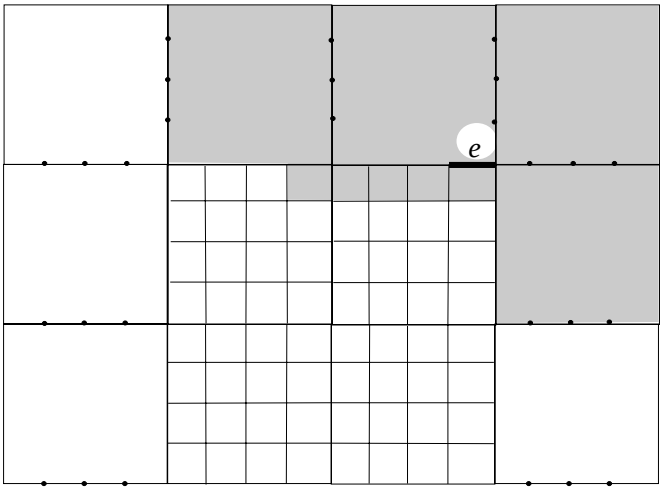


FIG. 2.1. *Part of a strong* 1*-conforming subdivision of a set of points. The shaded region is the union of cells* $\mathcal{U}(e)$ *forming a well-covering region of* e.

Our algorithm is based on a 2-conforming subdivision for V. For convenience, in the rest of the paper we use the term *conforming* to mean 2-conforming; when the conformity parameter is not 2, we state it explicitly.

2.2. Computing a conforming subdivision. Our strong conforming subdivision $\mathcal{S}$ is similar to a quad-tree in that all its edges are horizontal or vertical. However, the cells of $\mathcal{S}$ may be nonconvex and the subdivision itself may be disconnected. Each cell is reasonably well behaved, though there is at most one hole per cell. More specifically, each cell is either a square or a square-annulus (a square minus a square—see Figure 2.2); the boundaries of these squares, however, may be subdivided into a constant number of edges. Each square-annulus also has the following minimum clearance property:

> *Minimum clearance property:* The minimum width of an annulus in the subdivision (the minimum distance from the inner square to the

outer square) is at least one-quarter of the side length of the outer square.

Annuli and square faces are both subject to the uniform edge property:

Uniform edge property:

- Every edge on the outer square of an annulus has length $1/(4\lceil\alpha\rceil)$ times the side length of the outer square. Every edge on the inner square has length $1/(4\lceil\alpha\rceil)$ times the side length of the inner square.
- The lengths of edges on the boundary of a square cell differ by at most a factor of 4.

Our algorithm for computing a strong conforming subdivision of V is presented in section 6; describing it here would cause an unduly long digression. We simply state the main result from section 6.

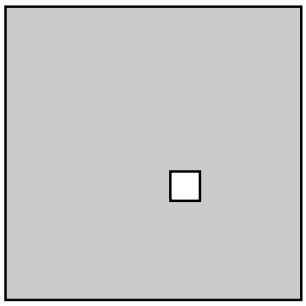


FIG. 2.2. *A square-annulus. The distance from the inner square to the outer square is at least* 1/4 *the side length of the outer square.*

THEOREM 2.1 (Conforming Subdivision Theorem). *For any $\alpha \geq 1$, every set of n points in the plane admits a strong α-conforming subdivision of $O(\alpha n)$ size satisfying the following additional properties:* (1) *all edges of the subdivision are horizontal or vertical,* (2) *each face is either a square or a square-annulus (with subdivided boundary),* (3) *each annulus has the minimum clearance property,* (4) *each face has the uniform edge property, and* (5) *every data point is contained in the interior of a square face. Such a subdivision can be computed in time $O(\alpha n + n \log n)$.*

We modify the strong conforming subdivision of V to accommodate the edges of the obstacles, producing a *conforming subdivision of the free space*. In the modified subdivision, there are two types of edges: the edges introduced by the subdivision construction and the original obstacle edges. To distinguish between them, we call the former *transparent* edges and the latter *opaque* edges; a wavefront can pass through the transparent edges, but it is blocked by the opaque edges. We require that all transparent edges be well-covered in the conforming subdivision of the free space (but not strongly so). Conditions (W1) and (W3) in the definition of well-covering are modified for the subdivision of free space as follows:

($W1_{fs}$) Let e be a transparent edge of $\mathcal{S}$. There exists a set of cells $\mathcal{C}(e) \subseteq \mathcal{S}$ such that e is contained in the closure of the union of cells $\mathcal{U}(e) = \{c \mid c \in \mathcal{C}(e)\}$.

($W3_{fs}$) Let e and f be two transparent edges of $\mathcal{S}$ such that f lies on the boundary of the well-covering region $\mathcal{U}(e)$. Then the shortest path distance between e and f is at least $\alpha \cdot \max(|e|, |f|)$.

Condition ($W3_{fs}$) ensures that e does not touch any transparent boundary edge of $\mathcal{U}(e)$, although it may touch opaque boundary edges.

Figure 2.3 shows an example of a well-covering region with obstacles. Lemma 2.2 shows how to modify a strong conforming subdivision of obstacle vertices to obtain a conforming subdivision of the free space. This subdivision of free space has the additional property that each obstacle vertex is incident to a transparent edge.

Remark. Our shortest path algorithm computes the distance from the source to the endpoints of all the transparent edges. The condition in the following lemma that each obstacle vertex is incident to a transparent edge ensures that the distance to each obstacle vertex is correctly computed.

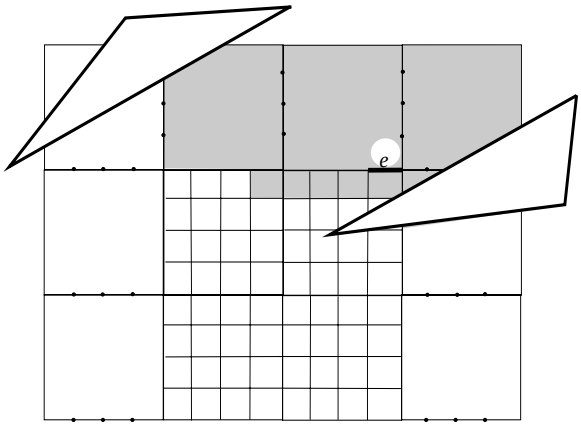


FIG. 2.3. *Part of a* 1*-conforming subdivision of free space. The shaded region is the well-covering region* $\mathcal{U}(e)$.

LEMMA 2.2. *Every family of disjoint simple polygons with a total of* n *vertices admits a* 2*-conforming subdivision of the free space with size* $O(n)$ *in which each obstacle vertex is incident to a transparent edge.*

Proof. Let $\mathcal{S}$ be a strong 2-conforming subdivision for V (the source vertex plus the vertices of the obstacle polygons), constructed according to Theorem 2.1. $\mathcal{S}$ has $O(n)$ vertices, edges, and faces (also referred to as cells), and each face is either a square or a square-annulus. Overlaying the obstacle edges on top of $\mathcal{S}$ cuts the plane into $O(n^2)$ cells. We call a face of this new subdivision $\mathcal{S}_{\text{overlay}}$ *interesting* if its boundary contains an obstacle vertex or a vertex of $\mathcal{S}$. For every vertex of $\mathcal{O}$ and for every vertex of $\mathcal{S}$, we keep intact the cells in $\mathcal{S}_{\text{overlay}}$ to which the vertex is incident (at most four cells per $\mathcal{S}$ vertex and two cells per obstacle vertex). We delete every edge fragment of $\mathcal{S}$ not on the boundary of one of these interesting cells.

Partition each cell containing an obstacle vertex v by extending edges vertically up and down from v. This cuts the cell into at most three convex pieces (since the cell is derived from a square of $\mathcal{S}$). Let c be the square in $\mathcal{S}$ that contains v, and let δ be the length of the shortest edge on the boundary of c. Subdivide each of the added

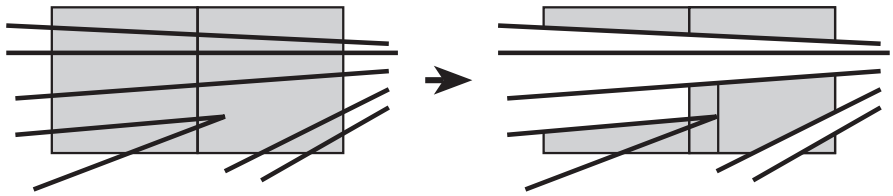


FIG. 2.4. *Constructing a conforming subdivision of the free space, given a strong conforming subdivision for the obstacle vertices. The shaded cells on the right are interesting cells.*

vertical edges incident to v into pieces of length at most δ (this produces $O(1)$ vertical edge fragments, since there are $O(1)$ edges on the boundary of c, all of approximately equal lengths, by the uniform edge property).

In the resulting subdivision, call it $\mathcal{S}'$, all cells are convex except those derived from square-annuli. Every nonconvexity in $\mathcal{S}_{\text{overlay}}$ is derived from a nonconvexity in either $\mathcal{S}$ or $\mathcal{O}$, since each face is the intersection of a face of $\mathcal{S}$ with a face in the arrangement of obstacle segments. Hence all nonconvex faces of $\mathcal{S}_{\text{overlay}}$ are interesting cells. Any face in $\mathcal{S}_{\text{overlay}}$ with an obstacle vertex on the boundary is cut into convex pieces by the vertical edges added through the vertex. The only other nonconvex vertices in $\mathcal{S}_{\text{overlay}}$ are annulus vertices. Each edge fragment that is deleted lies on the common boundary of two uninteresting faces; its deletion creates no new nonconvexities.

If a cell c of $\mathcal{S}$ has p edges on its boundary, then each subcell of c in $\mathcal{S}'$ that contains one of c's vertices has size at most $2p + O(1)$: each convex corner of c may be cut off by an obstacle edge, adding an extra edge; two obstacle edges may enter and exit through the same edge, leaving an obstacle vertex in the cell; and a subcell of an annulus may have up to two additional edges connecting the inner and outer squares. Adding vertical edges through each obstacle vertex splits a cell into at most three subcells, with at most $O(1)$ additional edges shared between them. Because each cell of $\mathcal{S}$ has constant complexity, the same is true of the interesting cells of $\mathcal{S}'$. It follows that the total complexity of the interesting cells is $O(n)$. Each uninteresting cell of $\mathcal{S}'$ (without a vertex of $\mathcal{S}$ or V) has at most eight edges—four edge fragments from $\mathcal{S}$ and four from $\mathcal{O}$. Each vertex in $\mathcal{S}'$ is a vertex of an interesting cell, so $\mathcal{S}'$ has $O(n)$ vertices, and, by planarity, $O(n)$ faces. See Figure 2.4 for a simplified example of the construction of $\mathcal{S}'$. In the remainder of the proof, we show that the portion of $\mathcal{S}'$ outside all obstacles in $\mathcal{O}$ is a *conforming subdivision of the free space*.

Condition (C1) is easily satisfied: each vertex of V lies in its own square cell in $\mathcal{S}$. These cells are interesting, and hence are retained (possibly subdivided) in $\mathcal{S}'$. Each cell of $\mathcal{S}_{\text{overlay}}$ therefore contains at most one vertex of V in its closure.

To show that all transparent edges of $\mathcal{S}'$ are well-covered (condition (C2)), consider such an edge e'. Edge e' may be a fragment of an edge $e \in \mathcal{S}$ (possibly $e = e'$), or it may be a fragment of a vertical edge added incident to an obstacle vertex. In the former case, define $\mathcal{U} = \mathcal{U}(e)$. In the latter case, e' is inside a square c of $\mathcal{S}$; define $\mathcal{U}$ to be the union of $\mathcal{U}(e)$ over all edges of c. Note that the boundary of $\mathcal{U}$ is covered by edge fragments in $\mathcal{S}$ (and hence in $\mathcal{S}_{\text{overlay}}$) but need not be in $\mathcal{S}'$: some edge fragments on the boundary of $\mathcal{U}$ may be erased in the construction of $\mathcal{S}'$. That is, $\mathcal{U}$ is a union of cells of $\mathcal{S}$ (and hence of $\mathcal{S}_{\text{overlay}}$), but not necessarily of $\mathcal{S}'$. Region $\mathcal{U}$ satisfies conditions (W1_{fs}) and (W3_{fs}); the latter holds because $\mathcal{U}$ satisfies condition (W3) for

the transparent edges of $\mathcal{S}$, and hence for those of $\mathcal{S}_{\text{overlay}}$. However, because $\mathcal{U}$ is not necessarily a union of cells of $\mathcal{S}'$, and may be cut into a nonconstant number of pieces by the obstacle polygons, we cannot use it directly as the well-covering region of e' in $\mathcal{S}'$.

We intersect $\mathcal{U}$ with free space. This partitions $\mathcal{U}$ into connected components $R_1, R_2, \ldots$. Exactly one component, call it R_1, contains e'. We show that each R_i is a union of $O(1)$ cells of $\mathcal{S}_{\text{overlay}}$, and hence that it has constant total complexity. We argue that for each cell c in $\mathcal{S}$, only a constant number of $\mathcal{S}_{\text{overlay}}$ subcells of c belong to R_i. If two subcells of c in $\mathcal{S}_{\text{overlay}}$ both belong to R_i, then the obstacle edges separating them must have endpoints either inside $\mathcal{U}$, or contained in one or more holes of $\mathcal{U}$ if $\mathcal{U}$ is multiply connected (see Figure 2.5). If we walk along the boundary of R_i, we visit subcells of c repeatedly. Between each pair of different subcells of c, we traverse the boundary of a different hole of $\mathcal{U}$ (or the outer boundary of $\mathcal{U}$, or the unique obstacle vertex inside $\mathcal{U}$). Because $\mathcal{U}$ has $O(1)$ holes, only $O(1)$ subcells of c belong to R_i.

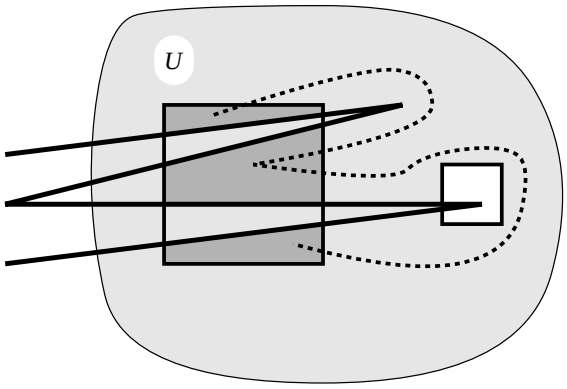


FIG. 2.5. *A cell of $\mathcal{U}$ may be partitioned into many subcells in $\mathcal{S}_{\text{overlay}}$, but only $O(1)$ of them belong to any one R_i.*

For any given component R_i, let $c(R_i)$ be the cells of $\mathcal{S}_{\text{overlay}}$ in R_i; $|c(R_i)| = O(1)$. Corresponding to each $c \in c(R_i)$, there is a unique cell c' in $\mathcal{S}'$ such that $c \subseteq c'$. Cell c is a strict subset of c' if and only if some edge of c was erased during the construction of $\mathcal{S}'$. If c is a strict subset of c', then c' is an uninteresting cell, and hence has at most eight edges. Thus both c and c' have constant complexity. Define

$$c'(R_i) = \{c' \mid c' \in \mathcal{S}' \text{ and } c \subseteq c' \text{ for some } c \in c(R_i)\}.$$

We have $|c'(R_i)| = O(|c(R_i)|) = O(1)$.

If $\mathcal{U}$ is nonconvex, it may be the case that some cell c' of $\mathcal{S}'$ that intersects R_i also intersects another component R_j, that is, $c'(R_i) \cap c'(R_j) \neq \emptyset$ (see Figure 2.6). Let us say that two components are connected, $R_i \sim R_j$, if and only if $c'(R_i) \cap c'(R_j) \neq \emptyset$, and extend $\sim$ to an equivalence relation by transitive closure.

We define $\mathcal{U}' = \mathcal{U}(e')$, the well-covering region for e' in $\mathcal{S}'$, to be the union of $c'(R_i)$ for all R_i in the equivalence class of R_1 under the $\sim$ relation. We argue that $\mathcal{U}'$ has constant complexity. Let $\overline{R}$ be the set of R_i that contain a vertex of $\mathcal{S}$ or $\mathcal{O}$. The set of cells $c'(\overline{R}) = \bigcup_{R_i \in \overline{R}} c'(R_i)$ has $O(1)$ total complexity. Further, if $R_i \notin \overline{R}$, then

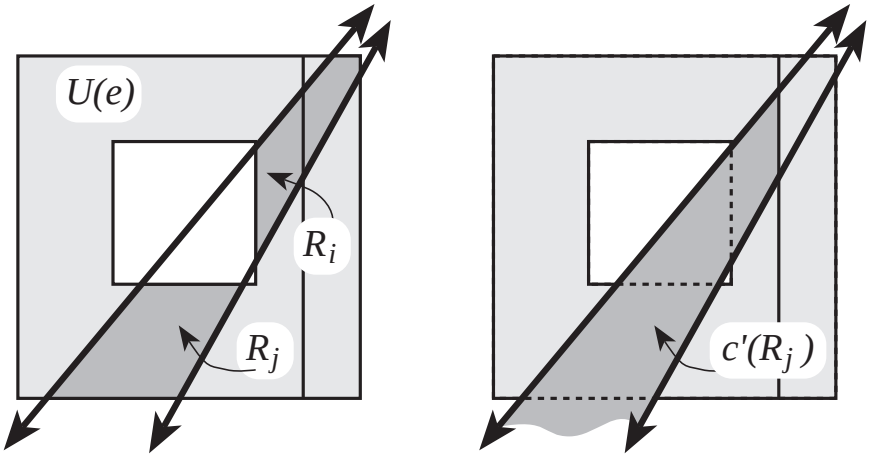


FIG. 2.6. *R_i and R_j are disjoint components of $\mathcal{U}(e)$ in $\mathcal{S}_{\text{overlay}}$. R_i is partitioned by a vertical line inside $\mathcal{U}(e)$, so $c(R_i)$ consists of two cells; $c(R_j)$ is a single cell. $c'(R_j)$ intersects both R_i and R_j, so $R_i \sim R_j$. Note that $c'(R_j)$ may have transparent edges outside $\mathcal{U}(e)$.*

$c'(R_i)$ is a single convex cell with $O(1)$ complexity (because all transparent edges of $c(R_i)$ inside $\mathcal{U}$ have been deleted). If such a cell $c' = c'(R_i)$ does not intersect any component in $\overline{R}$, then the union of $c'(R_j)$ for all $R_j \sim R_i$ is just the single cell c'. On the other hand, if c' does intersect some $R_j \in \overline{R}$, then $c' \cup c'(R_j)$ is identical to $c'(R_j)$. Because edge e' was not deleted, $R_1 \in \overline{R}$. It follows that $\mathcal{U}' \subseteq c'(\overline{R})$, and hence $\mathcal{U}'$ satisfies condition (W2).

The definition of $\mathcal{U}(e')$ implies that every transparent edge f' on the boundary of $\mathcal{U}(e')$ is outside or on the boundary of $\mathcal{U}$. Edge f' is a subset of some edge f of $\mathcal{S}$, so the Euclidean distance from e' to f' is at least $2 \cdot \max(|e'|, |f'|)$. It follows that condition (W3_{fs}) holds. Condition (W1_{fs}) holds by construction.

Let us now establish condition (C3). A well-covering region $\mathcal{U}(e')$ in $\mathcal{S}'$ contains no obstacle vertex that lies outside the well-covering region $\mathcal{U}$ in $\mathcal{S}$ from which $\mathcal{U}(e')$ is derived, since no edges of $\mathcal{S}$ that bound vertex-containing cells are deleted. If e' is a fragment of an edge e of $\mathcal{S}$, then its well-covering region $\mathcal{U}(e')$ in $\mathcal{S}'$ contains at most one obstacle vertex, since the same is true for $\mathcal{U} = \mathcal{U}(e)$ in $\mathcal{S}$. If e' is one of the edges added to $\mathcal{S}'$ inside a vertex-containing square, its well-covering region $\mathcal{U}$ is the union of $O(1)$ well-covering regions of $\mathcal{S}$. Each component region contains the square and its vertex and no other vertex; hence the well-covering region of e' in $\mathcal{S}'$ also satisfies condition (C3).

This completes our proof that $\mathcal{S}'$ is a conforming subdivision of the free space corresponding to the set of obstacles $\mathcal{O}$. □

Our next lemma shows that the conforming subdivision described above can be computed in $O(n \log n)$ time.

LEMMA 2.3. *The linear-size conforming subdivision of free space described in Lemma 2.2 can be built in time $O(n \log n)$.*

Proof. We start with a strong 2-conforming subdivision $\mathcal{S}$ of the obstacle vertices; $\mathcal{S}$ is computed in $O(n \log n)$ time by Theorem 2.1. In $O(n \log n)$ additional time, we build a point-location data structure for the obstacle polygons, so that given a query point q, we can in $O(\log n)$ time find the obstacle edge immediately to the left, right,

above, or below q [10, 16]. The edges of $\mathcal{S}'$ are obstacle edges, transparent edges on the boundary of kept cells, and transparent edges incident to obstacle vertices. To identify the second kind of edges, we trace the boundary of each kept cell separately. Each kept cell is contained in a single cell of $\mathcal{S}$ and has at least one vertex on its boundary, so we trace starting from each vertex. Tracing along an obstacle edge is easy, since the next transparent edge intersected is one of the $O(1)$ edges on the boundary of the current cell in $\mathcal{S}$. We use the point-location structure to trace along transparent edges: the next cell vertex is either a vertex of $\mathcal{S}$ or the first obstacle point hit by the ray that the current point and edge define. This tracing takes $O(n \log n)$ time altogether. The third kind of edges can be computed in $O(n)$ total time by local operations in each cell containing an obstacle vertex. To stitch the three kinds of edges into a single adjacency structure $\mathcal{S}'$, we use an $O(n \log n)$ time plane sweep algorithm [22]. □

This completes our discussion of the conforming subdivision. Our shortest path algorithm, described in section 4, relies heavily on the well-covering property of this subdivision. But first we establish some key geometric properties of shortest paths used by our algorithm.

3. Geometric properties of shortest paths. This section summarizes the properties of shortest paths we use in our algorithm. Most of these definitions and lemmas have appeared earlier [17, 18, 23]; we include them here for completeness.

The triangle inequality implies that a Euclidean shortest path turns only at obstacle vertices. Shortest paths need not be unique, though—for instance, every obstacle polygon O_i has at least one point on its boundary reached by two shortest paths; the two shortest paths together form a cycle enclosing the polygon. We use the notation $\pi(p, q)$ to denote the set of shortest paths connecting two points p and q. The length of any path in $\pi(p, q)$ is the shortest path distance between p and q, denoted $d(p, q)$. (Clearly, if one or both points lie inside an obstacle, there is no legal path between them; their shortest path distance is assumed to be infinite.) If the shortest path between p and q is the line segment $\overline{pq}$, then p and q are said to be mutually *visible*. We occasionally use $d(X, Y)$ to denote the shortest path distance between two sets of points X and Y, which is the minimum $d(x, y)$ over all pairs of points $x \in X$ and $y \in Y$.

We consider the problem of computing shortest paths from a fixed point s to all points of the free space. We define the *weight* of an obstacle vertex to be its shortest path distance to s. Given an arbitrary point p in free space, its *weighted distance* to a visible vertex u is defined as $|\overline{pu}| + d(u, s)$—the straight-line distance from p to u plus the shortest path distance from u to s. Obviously, the shortest path distance $d(p, s)$ is the minimum weighted distance between p and all vertices visible to p.

The *predecessor* of an arbitrary point p is defined as the vertex (or vertices) of V adjacent to p in $\pi(p, s)$; recall that V includes both s and the obstacle vertices. A predecessor of p is necessarily visible from p. (If p and s are mutually visible, then s is a predecessor of p.) The *shortest path map* of a particular source point s, denoted $SPM(s)$, is a subdivision of the plane into two-dimensional regions such that all the points in one region have the same unique predecessor. Points on region boundaries have multiple predecessors. These boundaries are pieces of *bisectors*—a bisector is the locus of points equidistant (by weighted distance) from two obstacle vertices, and it is in general an arc of a hyperbola. Figure 3.1 shows an example of a shortest path map.

The next three lemmas establish some fundamental properties of shortest paths

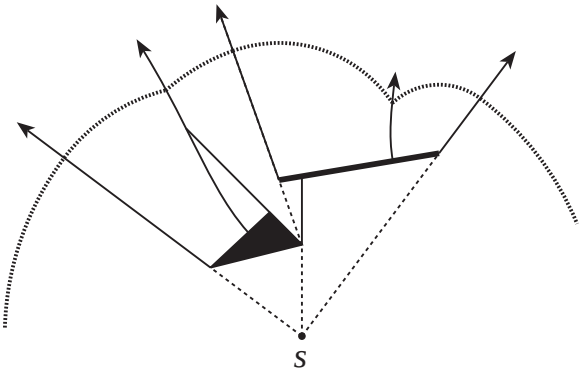


FIG. 3.1. *SPM*(s) *and a wavefront sweeping it.*

and shortest path maps.

LEMMA 3.1. *The set of points in the plane with multiple predecessors has measure zero.*

Proof. A point p with two obstacle vertices u and v as predecessors lies on the bisector of u and v, which is the hyperbola determined by the equation

$$|\overline{pu}| + d(u,s) = |\overline{pv}| + d(v,s).$$

There are at most $O(n^2)$ such hyperbolas, and each has measure zero. □

There are two types of edges in the subdivision *SPM*(s): (portions of) obstacle edges and arcs of hyperbolas determined by pairs of weighted vertices. The hyperbolic arcs may degenerate to straight lines—this happens when the weights of two vertices are equal, or differ by precisely the distance between the vertices; in the latter case the vertex with smaller weight is a predecessor of the other vertex. The vertices of *SPM*(s) are of three types: the obstacle vertices, the intersections of obstacle edges with (bisector) hyperbolic arcs, and the intersections of two or more bisectors; each of the last variety of vertices has three or more predecessors. The following lemma proves a linear upper bound on the total size of a shortest path map.

LEMMA 3.2. *The shortest path map SPM*(s) *has* $O(n)$ *vertices, edges, and faces. Each edge is a segment of a line or a hyperbola.*

Proof. We first observe that each face of *SPM*(s) is star-shaped, with the unique predecessor vertex for the face in its kernel—this follows from Lemma 3.1, which shows that interior points of a face have a unique predecessor.

The key step in the proof is to show that each obstacle vertex is the predecessor vertex for at most one face in *SPM*(s). Consider a vertex u that is the predecessor of a face F, and let *pred*(u) be the set of predecessors of u; observe that $d(u,s) = |\overline{uv}| + d(v,s)$ for any $v \in$ *pred*(u).

By the triangle inequality, if a point p is visible from a vertex $v \in$ *pred*(u), with v, u, p not collinear, then p cannot have u as its predecessor. Consider the subset of the free space that is visible from u but not visible from $v \in$ *pred*(u). Let $R(u,v)$ denote the component of this subset that is incident to u. Then $R(u,v)$ lies in an

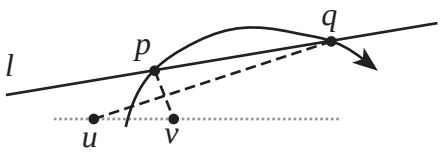


FIG. 3.2. *The intersection of $\overline{qu}$ with $\overline{pv}$ has two predecessors, even though it is not on a bisector—a contradiction.*

angular wedge around u of less than 180°. Define

$$R(u) = \bigcap_{v \in pred(u)} R(u, v).$$

Clearly, $F \subseteq R(u)$. We claim that there is at most one face of $SPM(s)$ in $R(u)$ with u as its predecessor. Suppose there were two faces, F_1 and F_2, both having u as their unique predecessor. The faces F_1 and F_2 have exactly one point in common: the vertex u. In the space between F_1 and F_2, there is a point p arbitrarily close to u with predecessor z such that z is distinct from both u and $pred(u)$. In other words, $|\overline{pu}| + d(u, s) > |\overline{pz}| + d(z, s)$. However, as p moves towards u, the difference in the distance shrinks, and finally $d(u, s) = |\overline{uz}| + d(z, s)$. But then z must be a predecessor of u, contradicting the hypothesis. Thus, a vertex u is a predecessor of at most one face in the shortest path map.

Finally, to prove the linear upper bound on the size of the shortest path map, recall that the number of obstacle vertices is n; the remaining vertices border at least three faces of $SPM(s)$ (for this argument, we count the obstacle polygons as faces of the shortest path map). Since the number of faces is $O(n)$, Euler's formula for planar graphs implies that the total number of vertices is also $O(n)$. This completes the proof. □

LEMMA 3.3. *Let u and v be two obstacle vertices that lie on the same side of a line ℓ. If ℓ intersects the bisector generated by u and v more than once, the intersections lie on opposite sides of the line supporting $\overline{uv}$.*

Proof. If the bisector is a straight line, the claim follows readily. Otherwise, the bisector is a hyperbola, and let us consider an arbitrary point p on this bisector. Every point on $\overline{pu}$ has u as its predecessor, and every point on $\overline{pv}$ has v as its predecessor. Points in the interiors of $\overline{pu}$ and $\overline{pv}$ have only one predecessor since they are not on the bisector (see Figure 3.2). If the half-bisector on one side of $\overline{uv}$ intersects ℓ at two points p and q, then there is an intersection of $\overline{pu}$ with $\overline{qv}$ or $\overline{pv}$ with $\overline{qu}$ that is not on the bisector and yet has two predecessors—a contradiction. □

With these preliminaries in place, we can now describe our shortest path algorithm, which works by propagating a wavefront through the conforming subdivision of the free space.

4. The shortest path algorithm. Our algorithm uses the *continuous Dijkstra method* [18, 19, 20], which simulates a unit-speed wavefront expanding from a point source and spreading among the obstacles. At simulation time t, the wavefront consists of points whose shortest path distance to the source is t. The wavefront is a set of disjoint paths and closed cycles. Each path or cycle is a sequence of circular arcs, called *wavelets*. Each wavelet is centered on an obstacle vertex that is already covered

by the wavefront, called the *generator* of the wavelet. As the wavefront expands, the meeting point of two adjacent wavelets sweeps along a *bisector curve*, which is the hyperbolic bisector of the two wavelets' generators. The endpoints of paths in the wavefront are formed where wavelets meet obstacle boundaries; these endpoints sweep along obstacle boundaries as the wavefront expands. During the wavefront simulation, the topology of the wavefront is changed by *events* of two types: wavefront-wavefront collisions and wavefront-obstacle collisions.

Our shortest path algorithm has two phases: a wavefront propagation phase, followed by a map computation phase. The first phase simulates the wavefront and determines approximate locations of all the wavefront collision events. The second phase uses this information to build the shortest path map in each cell of the conforming subdivision. In the following two subsections, we describe the details of these two phases, deferring the data structures and implementation issues of the propagation until the next section.

4.1. The propagation algorithm. Our algorithm works by propagating the wavefront through the cells of the conforming subdivision of the free space. The wavefront propagates between adjacent cells only across transparent edges; it dies upon meeting an opaque edge.

Propagating the exact wavefront appears to be quite difficult, so we content ourselves with computing two "single-sided" approximations to the wavefront at each transparent edge. Specifically, at each transparent edge, we compute two *approximate wavefronts*, passing through the edge in opposite directions. An approximate wavefront represents the wavefront reaching an edge from one side of the edge only. We can think of an approximate wavefront as labeling each point p on the edge with the time at which the approximate wavefront reaches p. The true distance $d(p, s)$ is the minimum of the two labels from opposite sides of the edge.

Remark. In some cases we can determine that a portion of a wavefront arrives at an edge after the wavefront from the other side of the same edge, and in such cases we drop the part that arrives later. In that sense, an approximate wavefront is not necessarily a complete representation of all the wavelets coming from one side of the edge.

An approximate wavefront at an edge e is represented as a sequence of obstacle vertices weighted with their shortest path distances from s. These vertices are the generators of the wavelets in the approximate wavefront. All the generators in an approximate wavefront sequence lie on the same side of e, since the approximate wavefront passes through e in one direction only. The core of our algorithm is a method for computing an approximate wavefront at an edge e based on the approximate wavefronts of nearby edges. These nearby edges are formalized in the following with the definitions of $input(e)$ and $output(e)$.

We denote by $input(e)$ the set of edges whose approximate wavefronts are used to compute the approximate wavefronts at e. This set consists of the transparent edges on the boundary of $\mathcal{U}(e)$, the well-covering region of e (cf. section 2.1). To compute the approximate wavefront at e, we propagate the approximate wavefronts from $input(e)$ to e inside $\mathcal{U}(e)$. The propagation algorithm introduces bends only at obstacle vertices in the closure of $\mathcal{U}(e)$; that is, the shortest paths corresponding to the wavefront do not bend except at obstacle vertices. Because $\mathcal{U}(e)$ need not be convex (nor even simply connected), nonconvexities of $\mathcal{U}(e)$ may block the wavefronts from some edges of $input(e)$ from reaching e. Typically, the paths corresponding to blocked wavefronts either run into obstacles outside $\mathcal{U}(e)$, or they pass through free

space outside $\mathcal{U}(e)$ and re-enter through other edges of *input*(e).

We denote by *output*(e) the set of edges to which the approximate wavefronts of e will be passed; *output*(e) = *input*$(e) \cup \{f \mid e \in \textit{input}(f)\}$. We set *output*$(e)$ to contain *input*(e) because our algorithm for detecting wavefront collision events depends on *output*(e) having a cycle enclosing e.

LEMMA 4.1. *For any transparent edge e, output(e) contains a constant number of edges.*

Proof. Because $|\mathcal{U}(f)| = O(1)$ for all f, and each $\mathcal{U}(f)$ is a connected set of cells of $\mathcal{S}'$, no edge e can belong to *input*(f) for more than $O(1)$ edges f. □

Our simulation of the wavefront propagation is loosely synchronized. For a transparent edge $e = \overline{ab}$, we define $\tilde{d}(e, s) = \min(d(a, s), d(b, s))$; this is a rough estimate of $d(e, s)$, since $d(e, s) \leq \tilde{d}(e, s) \leq d(e, s) + \frac{1}{2}|e|$. We compute the approximate wavefronts for e at the first time we are sure that e has been completely covered by wavefronts from the edges in *input*(e). This time is $\tilde{d}(e, s) + |e|$, the approximate time at which the expanding wavefront first hits an endpoint of e, plus the length of e. It is a conservative estimate of the time when e is completely run over by the wavefront.

We compute $\tilde{d}(e, s) + |e|$ on the fly for each edge e using a variable *covertime*(e). Initially, for every edge e whose well-covering region $\mathcal{U}(e)$ includes the source point s, we calculate an upper bound on $\tilde{d}(e, s)$ directly, considering only straight-line paths inside $\mathcal{U}(e)$, and set *covertime*(e) to this upper bound plus $|e|$. For all other edges, we initialize *covertime*$(e) = \infty$. Thus *covertime*(e) is not equal to $\tilde{d}(e, s) + |e|$ only for edges $e = \overline{ab}$ such that $\pi(a, s)$ or $\pi(b, s)$ crosses the boundary of $\mathcal{U}(e)$. The simulation maintains a time parameter t, and processes edges in order of their *covertime*$(\cdot)$ values. The main loop of the simulation is as follows:

PROPAGATION ALGORITHM

while there is an unprocessed transparent edge **do**

1. Select the edge e with minimum *covertime*(e), and set $t :=$ *covertime*(e).
2. Compute the approximate wavefronts at e based on the approximate wavefronts from all edges $f \in$ *input*(e) satisfying *covertime*$(f) <$ *covertime*(e). Compute $d(v, s)$ exactly for each endpoint v of e.
3. For each edge $g \in$ *output*(e), compute the time t_g when the approximate wavefront from e first engulfs an endpoint of g. Set *covertime*$(g) :=$ min (*covertime*(g), $t_g + |g|$).

endwhile

The following lemma proves the consistency of our algorithm—it shows that *covertime*() is correctly maintained and that the edges required for processing e are already processed. The details of Step 2 appear in sections 4.1.1 and 4.1.2; the computation of t_g in Step 3 is described in section 5.

LEMMA 4.2. *During the wavefront propagation, the following invariants hold:*

(*a*) *If the wavefront of an edge $f \in$ input(e) contributes to an approximate wavefront of e, then $\tilde{d}(f, s) + |f| < \tilde{d}(e, s) + |e|$.*

(*b*) *The value of covertime(e) is updated a constant number of times.*

(*c*) *The final value of covertime(e) is $\tilde{d}(e, s) + |e|$. This value is reached no later than the simulation clock reaches that time.*

(*d*) *Edge e is processed at simulation time $\tilde{d}(e,s)+|e|$.*

Proof. We establish the invariants separately:

(*a*) Any wavelet that contributes to the approximate wavefront at e must reach e at some time t_e with $d(e,s) \le t_e < \tilde{d}(e,s)+|e|$. Such a wavelet reaches e either by traveling straight from s inside $\mathcal{U}(e)$ or by passing through a transparent edge $f \in \mathit{input}(e)$ at an earlier time t_f, with $d(f,s) \le t_f < \tilde{d}(f,s)+|f|$ and $t_e \ge t_f + d(e,f)$. By condition ($\mathrm{W3_{fs}}$) of a well-covering region with parameter 2, $d(e,f) \ge 2|f|$, and so $t_e \ge d(f,s)+2|f|$. Since $\tilde{d}(f,s) \le d(f,s)+\frac{1}{2}|f|$, we can conclude that $\tilde{d}(f,s)+|f| < \tilde{d}(e,s)+|e|$.

(*b*) The value of $\mathit{covertime}(e)$ is updated only when an edge f is processed such that $f \in \mathit{input}(e)$ or $e \in \mathit{input}(f)$. There are $O(1)$ such edges, by Lemma 4.1.

(*c*), (*d*) We prove these by induction on the simulation clock. Claims (*c*) and (*d*) hold for the edges whose initial $\mathit{covertime}(\cdot)$ values are not infinite. The wavelet that first reaches an endpoint of e (at $t_e = \tilde{d}(e,s)$) passes through some $f \in \mathit{input}(e)$. By induction and the proof of (*a*), f has already been processed before the simulation clock reaches t_e, and so $\mathit{covertime}(e)$ is set to $\tilde{d}(e,s)+|e|$ no later than $t_e = \tilde{d}(e,s)$. The variable $\mathit{covertime}(e)$ cannot be set to any smaller value, because no approximate wavefront can reach the endpoints of e earlier than $\tilde{d}(e,s)$. It follows that e will be processed at simulation time $\tilde{d}(e,s)+|e|$. ∎

LEMMA 4.3. *For every vertex v of our conforming subdivision, the propagation algorithm correctly determines the distance $d(v,s)$ before v is used as a generator in any wavefront.*

Proof. Every vertex v of the conforming subdivision is an endpoint of a transparent edge e. The wavefront that determines $d(v,s)$ either reaches v from s by traveling only inside $\mathcal{U}(e)$, or it passes through an edge $f \in \mathit{input}(e)$ such that $\mathit{covertime}(f) < \mathit{covertime}(e)$. In the former case, initialization computes $d(v,s)$ correctly; in the latter case, Step 2 of the propagation algorithm implies that $d(v,s)$ is correctly computed. If v is an obstacle vertex, it may appear as a generator in a wavefront, but it will not be used until after $d(v,s)$ is computed at time $\tilde{d}(e,s)+|e|$ (Lemma 4.2(*d*)). ∎

While a well-covering region $\mathcal{U}(e)$ has constant complexity, it is not necessarily simply connected; consider, for instance, the case of a square-annulus. Consequently, there may be multiple, topologically distinct paths from a boundary edge $f \in \mathit{input}(e)$ to e. In order to avoid comparing paths of different topologies, we split the wavefront $W(e)$ into topologically equivalent pieces. In particular, let $W(e)$ denote one of the two approximate wavefronts passing through e. In computing $W(e)$ from a set $\{W(f) \mid f \in \mathit{input}(e)\}$, we use topologically constrained versions of the incoming wavefronts, denoted $W(f,e)$. A wavefront $W(f,e)$ is a portion of $W(f)$ that follows a single topological path inside $\mathcal{U}(e)$ from f to e.

If $\mathcal{U}(e)$ contains islands, there are multiple topologically distinct paths from an edge $f \in \mathit{input}(e)$ to e. When we need to refer to multiple topologically distinguished wavefronts from a single edge f to e, we use primed notation: $W(f,e)$, $W(f',e)$, etc.

If two points $p, q \in e$ are hit by a single topologically constrained wavefront $W(f,e)$, then the segments connecting p and q to their predecessors among the generator vertices in $W(f)$ intersect f and e, and the quadrilateral bounded by those segments and f and e is a subset of $\mathcal{U}(e)$. (The paths are not always segments: if an obstacle vertex v lies in the well-covering region of e and the path from f to p turns at v, then the predecessor of p in $W(f,e)$ may be v. Even in this case, the paths from p and q to f can be continuously deformed to each other inside $\mathcal{U}(e)$.) For

any point $p \in e$, the shortest path $\pi(p, s)$ passes through some $f \in \mathit{input}(e)$ (unless $s \in \mathcal{U}(e)$), so constraining the source wavefronts to pass through $\mathit{input}(e)$ does not lose any essential information.

4.1.1. The artificial wavefronts. When we compute the approximate wavefronts at a transparent edge e, we allow limited interaction between waves coming from opposite sides of the edge. This lets us eliminate some waves coming from one side of the edge that are dominated by waves from the other side. The interaction between the wavefronts from two sides is implemented using *artificial wavefronts*. These artificial wavefronts are our only mechanism for pruning the wavefront that arrives second at a transparent edge. We depend on artificial wavefronts to eliminate dominated wavefronts within a constant number of cells of where they first become dominated.

Consider a horizontal transparent edge e, and let v be an endpoint of e. We introduce an *artificial wavefront* with generator v and weight $d(v, s)$ into the computation of both approximate wavefronts at e. The triangle inequality implies that $d(p, s) \leq d(v, s) + |\overline{vp}|$, for any point $p \in e$. If the artificial wavefront reaches $p \in e$ before the wavefront from below e reaches p, then p is surely reached first by the upper wavefront, and so there is no need to propagate the lower wavefront through p. See Figure 4.1 for an illustration. In essence, an artificial wavefront is a convenient mechanism for discarding parts of the actual wavefront that are completely dominated by some other part of the wavefront. A generator of an artificial wavefront is not passed on to $\mathit{output}(e)$ as part of the approximate wavefront, unless it is also a vertex of $\mathcal{O}$.

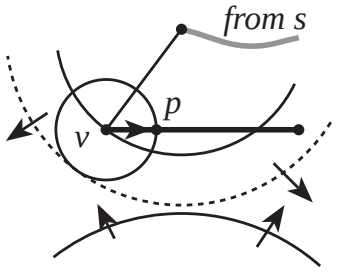


FIG. 4.1. *An artificial wavefront generated by v. If $d(v, s) + |\overline{vp}|$ is less than the time at which the wavefront from below reaches p, then p is reached first by a wavefront from above.*

Remark. An artificial wavefront is just a conceptual device that lets us argue about shortest paths without having to exhibit a specific shortest path. We use this technique in our proofs (e.g., Lemma 4.8) to discard generators at a cell boundary if the wavelet from an artificial wavefront reaches that boundary before the wavelets from those generators. Since the path passing through an artificial generator is no shorter than the true path from the predecessor of the artificial generator, the paths from the losing generators cannot be shortest paths.

When we compute the approximate wavefront passing through e from below (that is, coming from predecessors below e), the contributing wavefronts are the following:

1. All wavefronts $W(f, e)$ for $f \in \mathit{input}(e)$ and f below the line supporting e. (If f intersects the line supporting e, we split $W(f, e)$ in two, and keep only the portion $W(f', e)$ that comes from the part of f below e.)
2. An artificial wavefront expanding from each endpoint of e. An artificial wavefront generator v has weight $d(v, s)$.

The contributing wavefronts for the approximate wavefront passing through e from above are symmetric. The wavefront coming directly from s is handled separately.

The approximate wavefront from below is what the true wavefront would be if we were to block off the wavefront from above by adding extra obstacles. In physical terms, we can imagine replacing the transparent edge e with an (open) opaque obstacle segment. The opaque segment absorbs the wavefront from above, but the open endpoints let the wavefront from above pass through to generate artificial wavefronts. (Open endpoints are needed only to guard against the case in which an actual obstacle segment shares an endpoint of e, in which case replacing e with a closed segment would prevent artificial wavefronts from passing through the endpoint.)

Consider a set of wavefronts that reach e from the same side. We say that a contributing wavefront $W(f)$ *claims* a point $p \in e$ if $W(f)$ reaches p before any other contributor from the same side of e.

LEMMA 4.4. *Let e be horizontal, and let $W(f,e)$ and $W(g,e)$ be two contributors to the approximate wavefront that passes through e from below. Let x and x' be points on e claimed by $W(f,e)$, and let y be a point on e claimed by $W(g,e)$. Then y cannot lie between x and x'.*

Proof. Consider the shortest paths $\pi(x,s)$, $\pi(x',s)$, and $\pi(y,s)$ in the modified environment in which e has been replaced by an open, opaque segment. These paths connect x and x' to f, and y to g, inside $\mathcal{U}(e)$. Shortest paths $\pi(x,s)$, $\pi(x',s)$, and $\pi(y,s)$ do not cross. The subpaths of $\pi(x,s)$ and $\pi(x',s)$ inside $\mathcal{U}(e)$ can be continuously deformed to each other inside $\mathcal{U}(e)$, so g is not between them. It follows that y is not between them, either. ∎

LEMMA 4.5. *Let u and v be two obstacle vertices, both generating wavelets that are considered when the approximate wavefront passing through an edge e from below is computed. Then the bisector generated by u and v intersects e at most once in $SPM(s)$.*

Proof. Suppose the bisector intersects e twice. Without loss of generality assume u lies inside the loop formed by the bisector and e. If the bisector intersects e twice in $SPM(s)$, then the segment from u to its predecessor must intersect e between the two bisector intersections. This means that $d(e,s) < d(u,s)$; in fact, $d(e,s)+2|e| \leq d(u,s)$. Hence $\tilde{d}(e,s) + |e| < d(u,s)$, and u cannot contribute to the approximate wavefront at e: it does not become a generator until after e is processed, contradicting the assumption that both u and v contribute to the approximate wavefront at e. ∎

LEMMA 4.6. *Given $W(f,e)$ for each f below e that contributes to $W(e)$, we can compute the interval of e claimed by each $W(f,e)$ in $O(1+k)$ total time, where k is the total number of generators in all wavefronts $W(f,e)$ that are absent from $W(e)$.*

Proof. For each contributing wavefront $W(f,e)$, we show how to determine the portion of e claimed by $W(f,e)$ if only one other contributing wavefront $W(g,e)$ is present. Lemma 4.4 implies that this portion is contiguous. The intersection of these claimed portions, taken over all other contributors $W(g,e)$, is the part of e claimed by $W(f,e)$ in $W(e)$.

In constant time we determine whether the claim of $W(f,e)$ is left or right of that of $W(g,e)$. If both $W(f,e)$ and $W(g,e)$ reach the left endpoint of e, in constant time, check which one reaches it sooner. Otherwise, one of $W(f,e)$ and $W(g,e)$ reaches a point on e that is left of any point reached by the other, and this point determines the ordering. Without loss of generality, assume that the claim of $W(f,e)$ is left of that of $W(g,e)$.

By Lemma 4.4, we can combine the two wavefronts using only local operations.

Let a denote the generator in $W(f,e)$ claiming the rightmost point on e. Let p_a be the left endpoint of a's interval on e. Similarly, let b denote the generator in $W(g,e)$ claiming the leftmost point on e, and let p_b be the right endpoint of b's interval on e. Compute the bisector of a and b, and let its intersection with e be the point x. (By Lemma 4.5, there is only one intersection point in $SPM(s)$. If the hyperbola generated by a and b intersects e twice, then a is to the left of b at only one of the intersections, and we use that intersection as x.) See Figure 4.2. If x is to the left of p_a, then delete a from $W(f,e)$; if x is to the right of p_b, then delete b from $W(g,e)$; in either case, redefine a, b, p_a, p_b, recompute x, and repeat this test. If p_a is left of p_b and x lies between them, then x is the right endpoint of $W(f,e)$'s claim in the presence of $W(g,e)$.

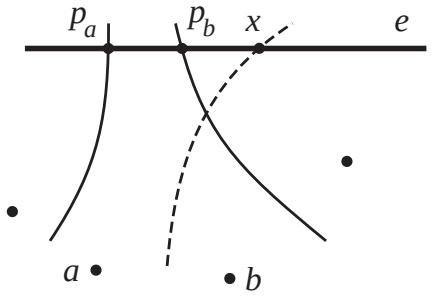


FIG. 4.2. *The contribution of b to $W(e)$ is constrained to be left of p_b and right of x, and therefore does not exist.*

By combining the claimed regions for all contributors $W(f,e)$, we construct the approximate wavefront at e. The time bound follows since we spend constant time per generator that is deleted for each pair of wavefronts, and the total number of wavefronts $W(f,e)$ to be merged is also a constant. This finishes the proof. □

LEMMA 4.7. *Any generator deleted during the construction of an approximate wavefront at edge e does not contribute to the true wavefront at e. Every generator that contributes to the true wavefront at e either is s or belongs to one of the approximate wavefronts at e.*

Proof. The first part is clear—every deleted generator is dominated by some other generator at e. The second part follows by induction from two facts: any wavelet that contributes to the true wavefront at e must come either from s inside $\mathcal{U}(e)$ or through one of the edges in $input(e)$ (by the definition of well-covering). The approximate wavefronts at $input(e)$ are ready before they are needed to construct $W(e)$ (by Lemma 4.2). □

4.1.2. The bisector events. When we propagate an approximate wavefront $W(e)$ to $output(e)$, we may detect bisector events, which are intersections of bisectors with each other or with obstacles. Bisector events are detected in two ways: (1) during the computation of $W(e,g)$ from $W(e)$ for some $g \in output(e)$; (2) during the merging process described in Lemma 4.6.

1. Bisector events of the first kind are detected when we simulate the advance of the wavefront from e to g to compute $W(e,g)$; the details of this simulation are discussed in section 5. In particular, if two generators u and v are nonadjacent in $W(e)$, but become adjacent at any time during the propagation from e to g, then there is a bisector event involving u and v.

2. Bisector events of the second kind are detected during merging. If a generator v contributes to one of the input wavefronts $W(e, g)$ but not to the merged wavefront $W(g)$ at g, then v is involved in a bisector event on the way from e to g. (As a special case, if a generator's claim on $W(g)$ is shortened (but not eliminated) by an artificial wavefront, then that generator is also considered to have a bisector event. This adds at most two extra bisector events for each edge g.)

Our algorithm detects bisector events in a small neighborhood of their actual location in $SPM(s)$. To ensure that all bisector events are properly localized, we *mark* the generators that participate in a bisector event in $O(1)$ cells near where the event is detected: if a generator v is involved in a bisector event in a cell c, then v is guaranteed to belong to a set of marked generators for c. However, the set of marked generators for a cell c may be a superset of the generators that actually participate in bisector events in c. We will show that the total number of generators marked in all the cells is $O(n)$. The precise rules for marking the generators are given below.

Marking Rules for Generators

1. If a generator v lies in a cell c, then mark v in c.
2. Let e be a transparent edge, and let $W(e)$ be the approximate wavefront coming from some generator v's side of e.
   (a) If v claims an endpoint of e in $W(e)$, or if it would do so except for an artificial wavefront, then mark v in all cells incident to the claimed endpoint.
   (b) If v's claim in $W(e)$ is shortened or eliminated by an artificial wavefront, then mark v in the cell on v's side of e.
3. Let e and f be two transparent edges with $f \in output(e)$. Mark v in both the cells that have e as an edge if one of the following events occurs:
   (a) v claims an endpoint of f in $W(e, f)$;
   (b) v participates in a bisector event detected either during the computation of $W(e, f)$ from $W(e)$, or during the merging step at f (Lemma 4.6). (We also mark v as having a bisector event if v's claim on $W(f)$ is shortened by an artificial wavefront.)
4. If v claims part of an opaque edge when it is propagated from an edge e toward $output(e)$, mark v in both cells with e on their boundary.

Rules 2a and 3a both apply when a wavefront claims an endpoint of an edge. The main difference between the two rules is that Rule 2a puts marks in cells near the claimed endpoint, and Rule 3a puts marks in cells near the source edge of the wavefront.

A generator may contribute to a wavefront more than once in the wavefront sequence; each mark applies to only one instance of the generator in the sequence. The following technical lemma is used in the proof of Lemma 4.9 to establish the correctness of the marking rules.

Lemma 4.8. *Let v be a generator that contributes to an approximate wavefront $W(e)$. Suppose there is a point $p \in e$ that is claimed by v in $W(e)$ but not in $SPM(s)$*

(because a wave from the other side of e reaches p first). Then v is marked in the cell c on v's side of e.

Proof. If v is unmarked in c, there must be generators u and w such that u, v, w are consecutive in $W(e)$—otherwise Rule 2 would apply. The bisectors (u, v) and (v, w) must exit from $\mathcal{U}(e)$ through the same transparent edge h—otherwise Rule 3 or 4 would apply. For the same reason, the region bounded by (u, v), (v, w), h, and e is a subset of $\mathcal{U}(e)$—if the region contained a non-$\mathcal{U}(e)$ island, v would claim an endpoint of a boundary edge of that island. Edge h is by definition part of $input(e)$. Consider the point $p \in e$ that is claimed by v in the approximate wavefront $W(e)$ but not in the true wavefront at e, and suppose that the true predecessor of p is $z \neq v$. The vertex z is either an obstacle vertex or the source s. In the former case, z lies outside $\mathcal{U}(e)$ or on its boundary $\partial\mathcal{U}(e)$—by condition (C3), $\mathcal{U}(e)$ contains at most one obstacle vertex, so any vertex not strictly outside $\mathcal{U}(e)$ must be connected to points outside $\mathcal{U}(e)$ by opaque edges. Vertex z may lie strictly inside $\mathcal{U}(e)$ only if $z = s$.

Let us first assume that z lies outside the well-covering region $\mathcal{U}(e)$—the proof simplifies in the other case, which is considered below. Let q denote the intersection point between $\overline{zp}$ and $input(e)$ closest to p (recall that $input(e) \subset output(e)$, and $input(e) \subset \partial\mathcal{U}(e)$). Based on the position of q relative to the bisectors (u, v) and (v, w), we argue that v must have been involved in a bisector event detected by our algorithm, and thus marked in cell c.

First, consider the case in which q lies between the bisectors (u, v) and (v, w) on the edge h. Now, since $|\overline{qp}| \geq |h|$ (by the well-covering property), the endpoints of h are engulfed by a wavefront from z or from some other generator before the wavefront from z reaches p at time $d(z, s) + |\overline{zp}|$. The artificial wavefronts from h's endpoints will cover h before time $d(z, s) + |\overline{zp}| + |h|$. By assumption we have $d(v, s) + |\overline{vp}| > d(z, s) + |\overline{zp}|$. The wavefront from v cannot reach e earlier than $d(v, s) + |\overline{vp}| - |e|$. By well-covering with parameter 2, $d(e, h)$ is at least $|e| + |h|$, and so the wavefront from v reaches h no earlier than $d(v, s)+|\overline{vp}|+|h| > d(z, s)+|\overline{zp}|+|h|$, at which time h is already covered by the artificial wavefront. The claim of v on h is shortened by the artificial wavefront (in fact, v's claim is eliminated completely), and so it must be marked by Rule 3b.

In the second case, q is not between the bisectors (u, v) and (v, w) on h. The segment $\overline{qp}$ must intersect one of the bisectors. Without loss of generality, assume $\overline{qp}$ intersects bisector (u, v). Since every point on $\overline{qp}$ has z as its predecessor in $SPM(s)$, the bisector (u, v) does not reach $\partial\mathcal{U}(e)$ in $SPM(s)$. We show that our propagation and merging algorithms will detect a bisector event for (u, v). Let r be the intersection point between the bisector (u, v) and the edge h. As noted in the discussion after Lemma 4.3, the triangle defined by the segments $\overline{ur}$, $\overline{vr}$, and e is a subset of $\mathcal{U}(e)$. Bisector (u, v) crosses the triangle boundary on e and at r, but nowhere else. The larger region R bounded by e, h, $\overline{ur}$, and bisector (v, w) also is a subset of $\mathcal{U}(e)$, and it contains point p. Because $\overline{qp}$ crosses into R to intersect (u, v), and it does not intersect the (v, w) or h sides of R, $\overline{qp}$ must intersect $\overline{ur}$; let x be the point of intersection. The wavelet from z reaches x before the one from u, so the path $z \to x \to r$, starting at time $d(z, s)$, reaches r before the path $u \to r$, starting at time $d(u, s)$. Observe also that the path $z \to x \to r$ is a legal path—it lies in free space. Now, consider the shortest path from z to r inside the triangle $\triangle zxr$ that does not cross h or any obstacle edge (see Figure 4.3). Because $z \to x \to r$ lies in free space, such a path exists, and is shorter than $z \to x \to r$. This path claims r from the same side as u before the wavelet from u reaches r. (If the path passes through an endpoint of h, then an artificial wavefront claims r; otherwise the last obstacle vertex on the path

claims r.) Thus, a bisector event for (u, v) is detected during the computation of $W(e, h)$ or $W(h)$, and v is marked by Rule 3b.

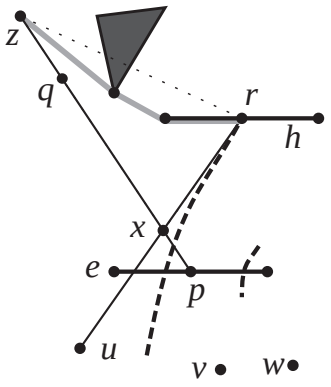


FIG. 4.3. *The shaded path from z to r claims r before the wavelet from u, and from the same side of h as u.*

Next consider what happens if the predecessor vertex z lies on the boundary of the well-covering region $\mathcal{U}(e)$. Let h be a boundary edge of $\mathcal{U}(e)$ incident to z. In this case we detect a bisector event involving v when we advance the wavefront from e to $output(e)$: if z lies between the bisectors (u, v) and (v, w), then v is marked by Rule 3a or 4; if z is not between the bisectors, the segment $\overline{zp}$ intersects one of the bisectors, say (u, v), and we detect a bisector event for (u, v) in advancing the wavefront from e to $output(e)$.

Finally, consider the case in which $z = s$ lies inside $\mathcal{U}(e)$. If z is not between the bisectors (u, v) and (v, w), segment $\overline{zp}$ intersects one of them and the proof is as above. Let r be the intersection of (u, v) with h, and let t be the intersection of (v, w) with h. The convex quadrilateral bounded by subsegments of e, $\overline{ur}$, h, and $\overline{tw}$ is contained inside $\mathcal{U}(e)$. Hence if z is between the bisectors (u, v) and (v, w), the entire segment $\overline{rt}$ is visible from z (that is, $\triangle zrt$ is empty) and so v's claim on h is eliminated by z. Therefore v is marked by Rule 3b. This completes the proof. ∎

LEMMA 4.9. *If a generator v participates in a bisector event of $SPM(s)$ in a cell c, then v is marked in c.*

Proof. If a bisector has an endpoint on an opaque edge of c, it either emanates from an obstacle vertex on the edge, or it is defined by two generators that claim part of the opaque edge. Rules 1 and 4 guarantee that all such generators are marked in c. If a generator v that contributes to an approximate wavefront in c is unmarked, then by Rule 2a there must be transparent edges e and f on the boundary of c such that $W(e)$ and $W(f)$ both contain the generator subsequence u, v, w, for some u and w. Without loss of generality assume $W(e)$ enters c and $W(f)$ leaves c. If v participates in a bisector event of $SPM(s)$ in c, then at least one point p inside the region R bounded by e, f, (u, v), and (v, w) is not claimed by v in $SPM(s)$. Let z be the true predecessor of p. Let r and t be the intersections of (u, v) and (v, w) with f, respectively. Region R is contained in the convex quadrilateral Q bounded by $\overline{ur}$, $\overline{rt}$, $\overline{tw}$, and the line supporting e. Because u, v, w is a subsequence of $W(e)$, no vertex on the same side of e as v claims any point of the side of Q collinear with e; that is, $\overline{zp}$ does not cross that side of Q. If r and t are both claimed by v in $SPM(s)$, then $\overline{ur} \in \pi(s, r)$, and $\overline{wt} \in \pi(s, t)$. In this case $\pi(s, p)$ cannot cross $\overline{ur}$ or $\overline{wt}$, and hence it

must cross $\overline{rt}$. The intersection of $\overline{zp}$ with $\overline{rt}$ is a point q that satisfies the hypothesis of Lemma 4.8, and so v is marked in c. On the other hand, if either r or t is not claimed by v in $SPM(s)$, that vertex satisfies the hypothesis of Lemma 4.8, and so v is marked in c. □

The following technical lemma shows that the approximate wavefronts are not too different from the true wavefronts; this lets us bound the number of marks made by the marking rules.

LEMMA 4.10. *Let B be the set of pairs (e, b) of transparent edges e and bisectors b such that b crosses e in some approximate wavefront, but the same crossing does not occur in $SPM(s)$. Then $|B| = O(n)$.*

Proof. Let (e, b) be a pair in B. Bisector b is defined by two generators u and v. The proof of Lemma 4.8 notes that each generator (except possibly s) is outside or on the boundary of $\mathcal{U}(e)$. That proof also shows that b's intersection with e in some approximate wavefront (that is, the presence of u and v in $W(e)$) is proof that u and v claim points on the boundary of $\mathcal{U}(e)$ (in $input(e)$) in $SPM(s)$. Let $p = b \cap e$. Because (e, b) is not an incident pair in $SPM(s)$, there must be at least one bisector event in $SPM(s)$ that lies in the interior of $\mathcal{U}(e)$ between the line segments $\overline{up}$ and $\overline{vp}$. We can charge the early demise of b to any one of these bisector events.

The segments $\overline{pu}$ and $\overline{pv}$ are disjoint inside $\mathcal{U}(e)$ from the corresponding segments defined by any other pair $(e, b') \in B$—in the modified shortest path problem in which the obstacles are $\mathcal{O} \cup \{e\}$, the segments $\overline{pv}$ and $\overline{pu}$ belong to $\pi(s, p)$, and hence they are disjoint from any other such segments. Thus the sector bounded by $\overrightarrow{pu}$ and $\overrightarrow{pv}$ is disjoint inside $\mathcal{U}(e)$ from the sector defined by any other pair $(e, b') \in B$, so each bisector event inside $\mathcal{U}(e)$ is charged at most once for all pairs in B that have e as the first element of the pair. Each cell in the conforming subdivision belongs to $O(1)$ well-covering regions $\mathcal{U}(e)$. Hence the sum over all transparent edges e of the number of bisector events in $\mathcal{U}(e)$ is only $O(n)$. This total is an upper bound on $|B|$. □

LEMMA 4.11. *The total number of marked generators over all cells is $O(n)$.*

Proof. We begin by defining a *propagation region* for each edge e. For any transparent edge e, let $P(e)$ be the collection of cells through which wavefronts propagate on the way from e to all edges $f \in output(e)$. Clearly $P(e) \subseteq \mathcal{U}(e) \cup \{\mathcal{U}(f) \mid f \in output(e)\}$. The number of cells in $P(e)$ is constant, since $|output(e)|$ is constant, and so is the number of cells in $\mathcal{U}(f)$ for any f. Furthermore, since every cell of $P(e)$ is within a constant number of cells of e, each cell c belongs to $P(e')$ for only a constant number of edges e'.

The total number of generator-cell marks made under Rule 1 is clearly $O(n)$.

Each $P(e)$ has constant complexity, so there are $O(n)$ edge pairs (e, f), where e is transparent and f is either transparent and in $output(e)$, or opaque and inside or on the boundary of $P(e)$. From this it follows that the number of marks made by Rules 2a and 3a is $O(n)$. Similarly, there are $O(n)$ Rule 4 marks in which the wavelet from v claims an endpoint of the opaque edge, or is the first or last nonartificial wavelet in $W(e)$.

Any Rule 4 mark not yet counted involves a generator v that does not reach any opaque edge endpoint when propagated forward from e. Because v is not the first or last nonartificial wavelet in $W(e)$, there is a generator u such that v's claim on e in $W(e)$ is bounded on the left by bisector (u, v). We can assume that (u, v) intersects e in $SPM(s)$; by Lemma 4.10 there are only $O(n)$ bisector-edge pairs that intersect in approximate wavefronts but not in $SPM(s)$. Bisector (u, v) terminates in $P(e)$, either on the opaque edge or in a bisector event before the opaque edge. Let us charge the

marking of v at e to this endpoint of (u, v) in $SPM(s)$. Because each cell belongs to $P(e')$ for a constant number of edges e', each vertex of $SPM(s)$ is charged $O(1)$ times. Since $|SPM(s)| = O(n)$, the number of Rule 4 marks is $O(n)$.

The proofs for Rules 2b and 3b are similar to that for Rule 4. We begin with the proof for Rule 3b. We can assume that the interval claimed by v on e in $W(e)$ is bounded by two bisectors (u, v) and (v, w), for two nonartificial generators u and w; the first and last generators in $W(e)$, counted separately, sum to at most $O(n)$ overall. Furthermore, we can assume that (u, v) and (v, w) both intersect e in $SPM(s)$; there are only $O(n)$ bisector-edge pairs that appear in some approximate wavefront but not in $SPM(s)$ (Lemma 4.10). At least one of the two bisectors fails to reach the boundary of $P(e)$ in $SPM(s)$, because Rule 3b applies, and a detected bisector event implies the existence of an actual bisector event no later than the point of detection; we charge the marking of v to that bisector endpoint. Each bisector event gets charged $O(1)$ times, and there are $O(n)$ bisector events in $SPM(s)$.

To bound the number of Rule 2b marks, consider where the generator v lies. There is at most one generator v inside $\mathcal{U}(e)$, and so $O(n)$ marks for such generators overall. If v lies outside $\mathcal{U}(e)$, there is at least one edge in $input(e)$ where v is marked by Rule 3b because of the shortening of v's claim on e. Charge the Rule 2b mark at e to this Rule 3b mark. There are $O(n)$ Rule 3b marks and hence $O(n)$ Rule 2b marks. □

We defer the finer details of the propagation algorithm to section 5 and instead describe the second phase of the algorithm next, namely, the shortest path map computation.

4.2. Computing the shortest path map. At the end of the propagation phase, approximate wavefronts for all transparent edges have been computed. Furthermore, for every cell c, a set of *marked* generators is known; each marked generator is in the approximate wavefront of one of the boundary edges of c, and all but $O(1)$ of them contribute to a bisector event either in c or in one of $O(1)$ nearby cells. The algorithms of Lemma 4.6 and section 5 let us compute the marked generators in $O(\log n)$ time apiece.

We now show how to break the interior of a cell c into *active* and *inactive* regions such that no vertices of $SPM(s)$ lie in the inactive regions. By Lemma 4.9, no unmarked generator contributes to a bisector event in c. A bisector defined by a marked generator and an unmarked neighbor belongs to $SPM(s)$. All such bisectors are disjoint. They partition c into regions such that each region is claimed only by marked generators or only by unmarked generators. These are the active and inactive regions, respectively (see Figure 4.4). The active regions can be computed in time proportional to the number of marked generators in c, since the order of the generators along the boundary of c is known.

The boundary of an active region consists of $O(1)$ segments. Each segment is a transparent edge fragment, an opaque edge, or a bisector in $SPM(s)$. Let e be a transparent edge fragment bounding an active region, and let $W(e)$ be the wavefront that enters the active region by crossing e. In the absence of wavefronts from other transparent edges, $W(e)$ partitions the active region into pieces we call S-faces, each with a unique predecessor in $W(e)$. These S-faces may not cover the active region, since each point in an S-face must be connected to its predecessor by a segment that intersects e. Denote this partition by $S(e)$. $S(e)$ is essentially a shortest path map, restricted to the active region and considering only generators in $W(e)$. If a point p lies in an S-face of $S(e)$ with predecessor v, then $S(e)$ assigns weight $|\overline{pv}| + d(v, s)$ to

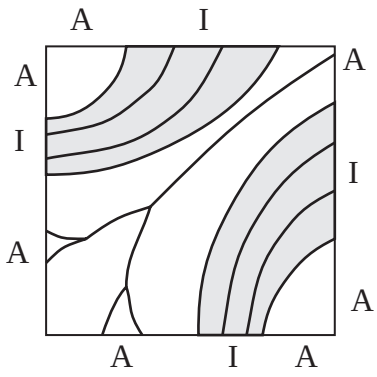


FIG. 4.4. *Active regions (white) and inactive regions (shaded). Each region-bounding bisector is defined by one marked and one unmarked generator.*

p. Points outside any S-face are assigned infinite weight by $S(e)$. We can compute $S(e)$ in $O(m \log m)$ time, where $m = |W(e)|$, by using the propagation algorithm and data structure of section 5.

The following lemma shows how to combine the wavefronts incident to different boundary edges of an active region.

LEMMA 4.12. *Given the approximate wavefronts on the boundary of a cell c and a set of k marked generators in those wavefronts, we can compute the vertices of $SPM(s)$ inside c in time $O(k \log k)$.*

Proof. Consider an active region inside c and two transparent edge fragments e and f on the boundary of this active region. We can use the merge step from a standard divide-and-conquer Voronoi diagram algorithm to compute the portion of the region nearer to $W(e)$ than to $W(f)$, using weighted distance, in time $O(|W(e)| + |W(f)|)$. More specifically, assume that $S(e)$ and $S(f)$ have both been computed. Let $m = |W(e)| + |W(f)|$. Each of $S(e)$ and $S(f)$ defines a distance function on the points of the active region. The pointwise minimum of these two functions determines which points are nearer to $W(e)$ than to $W(f)$ under weighted distance. Consider a point p in the S-face for some generator $v \in W(e)$. Point p belongs to v's S-face in $SPM(s)$ only if all of the segment $\overline{pv}$ is closer to v than to any generator in $W(f)$. The set of points p such that the entire segment from p to its predecessor is closer to $W(e)$ than to $W(f)$ is bounded by a single chain Γ of $O(m)$ hyperbolic arcs. (The number of arcs follows from Lemma 3.2.) To find Γ, first trace along a ray emanating from some generator $v \in W(e)$, marching through $S(e)$ and $S(f)$ simultaneously, until the ray reaches the boundary of c or reaches a point whose weight in $S(f)$ equals its weight in $S(e)$. This takes $O(m)$ time, since a line cuts $O(m)$ edges of $S(e)$ and $S(f)$. Then trace outward from this point along Γ. Each arc of Γ is a hyperbola determined by the generators of the S-faces of $S(e)$ and $S(f)$ containing the current point; trace along the hyperbola until it leaves one of the two S-faces, then follow the hyperbola determined by the next pair of S-faces, etc. This procedure takes $O(1)$ time per arc of Γ, or $O(m)$ time altogether (see Figure 4.5).

The tracing procedure computes the region closer to $W(e)$ than to $W(f)$ for one edge f. Intersecting the results for all such edges f on the boundary of the active region produces the region $R(e)$ claimed by $W(e)$ in $SPM(s)$. Intersecting $R(e)$ with $S(e)$ gives the vertices of $SPM(s)$ to which $W(e)$ contributes. We repeat this computation for each transparent edge fragment to find all the vertices of $SPM(s)$ in

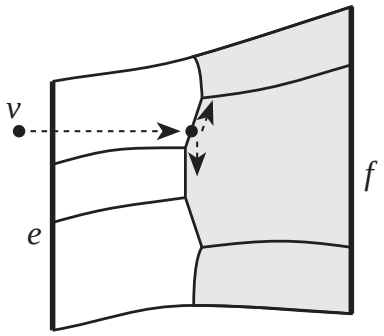


FIG. 4.5. *To find the region closer to $W(e)$ than to $W(f)$ under weighted distance, trace a ray from some $v \in W(e)$ through $S(e)$ and $S(f)$ until it hits a point equidistant from the two wavefronts; then trace outward from the point along the bisector Γ.*

the active region. Applying this algorithm to all active regions finds all vertices of $SPM(s)$ inside c.

The partition $S(e)$ determined by each edge fragment e participates $O(1)$ times in a Voronoi-style merge, so the total cost of merging is $O(k)$. Hence the running time is dominated by the propagation algorithm, which takes $O(k \log k)$ time altogether. □

LEMMA 4.13. *The shortest path map vertices computed cell-by-cell can be combined to build $SPM(s)$ in additional $O(n \log n)$ time.*

Proof. To compute $SPM(s)$, we compute all its edges separately, then use a standard plane sweep to assemble them, as follows. Create a list of the bisector endpoints discovered in the computation of Lemma 4.12, each identified by a key consisting of two generators. Put each three-bisector endpoint into the list three times, once for each bisector. Put each bisector/edge collision in once, labeled with the generators of the bisector. Now sort the list to group together endpoints belonging to each bisector. Take the endpoints belonging to the bisector of a generator pair (v, w) and sort them along the hyperbola determined by the weighted generators v and w. This determines all edges of $SPM(s)$ on the hyperbola. Doing this for all pairs that appear as keys in the sorted list gives all $O(n)$ hyperbolic arcs of $SPM(s)$. Finally, with a standard plane sweep [22], we can combine these arcs with the edges of $\mathcal{O}$ to build the subdivision $SPM(s)$. □

5. An implementation of the wavefront propagation. In this section, we give the implementation details of our algorithm. We describe the data structures used by our algorithm and finer details of the propagation algorithm.

5.1. The data structures. An approximate wavefront is a list of generators (obstacle vertices). Our algorithm performs the following two types of operations on these lists:

1. *Standard list operations:* insert, delete, concatenate, split, find previous and next elements, and search. The search operation locates the position of a query point in the list of bisectors defined by the generators at a particular time.
2. *Priority queue operations:* we assign each generator in the list a priority, and the data structure needs to update priorities and find the minimum priority in the list.

Both of these types of operations can be supported by a data structure based on balanced binary trees, for example, red-black trees, with the generators at the leaves. In particular, the list operations take $O(\log n)$ time each because the maximum list length is $O(n)$. The priority queue operations are supported by adding a priority field to the nodes of the binary tree: each node records the minimum priority of the leaves in its subtree. Each priority queue operation takes $O(\log n)$ time, while the list operations retain their $O(\log n)$ bound.

We also require our data structure to be fully persistent—we need the ability to operate on past versions of any list. Each of the two kinds of operations uses $O(1)$ storage per node of the binary tree, so we can make the data structure fully persistent by path-copying. Each of our operations affects $O(\log n)$ nodes of the tree, including all the ancestors of every affected node. Once we have determined which nodes an operation will affect, and before the operation modifies any node, we copy all of the nodes that will be affected, then modify the copies. This creates a new version of the tree while leaving the old version unchanged. The data structure uses $O(m \log n)$ storage, where m is the total number of data structure operations, and keeps the $O(\log n)$ per-operation time bound quoted above.

LEMMA 5.1. *There is a linear-space data structure that represents an approximate wavefront and supports list operations and priority queue operations in $O(\log n)$ time per operation. The data structure can be made fully persistent at the expense of an additional $O(\log n)$ space per operation.*

5.2. Details of the wavefront propagation. Using the data structures just described, we now show how to propagate an approximate wavefront from edge to edge. In particular, given an approximate wavefront $W(e)$, we show how to compute $W(e, g)$ for every edge $g \in \mathit{output}(e)$. In the process, we also determine the time of first contact between $W(e, g)$ and the endpoints of g.

We describe how to compute $W(e, g)$ for all the transparent edges g on the boundary of e's cell. Because the edges of $\mathit{output}(e)$ belong to a constant number of cells in the neighborhood of e, we can use this primitive to compute $W(e, g)$ for all $g \in \mathit{output}(e)$. When we propagate the wavefront cell-by-cell, we effectively split $W(e, g)$ into multiple pieces, each labeled by the sequence of transparent edges it follows from e to g. We assemble $W(e, g)$ out of these component wavefronts by concatenating pieces that correspond to topologically equivalent paths inside $\mathcal{U}(e)$. (Recall that for a pair e and g, there may be several constrained wavefronts $W(e, g)$, $W(e', g)$, etc., topologically distinguished by the paths they follow among the islands inside $\mathcal{U}(e)$.) Each component piece is a list of generators; adjacent pieces may contain a single duplicate generator, namely, the generator that claims the common endpoint. Before concatenation of the lists, one copy of the duplicate generator is deleted. In Figure 5.1, $W(e, g)$ is assembled from $W(e', g)$ and $W(e'', g)$, where e' and e'' are two edges on the boundary of g's cell.

5.2.1. Preparing the cells for propagation. The propagation algorithm that follows assumes that the cell c is convex. When c is nonconvex, which is the case for subcells of an annulus cell, we temporarily break c into convex subcells by adding transparent edges *parallel to* e through the points of nonconvexity, as illustrated by Figure 5.2.

Let $f \neq e$ be another transparent edge on the boundary of c. Our propagation algorithm assumes the following invariant:

> **Propagation invariant**: When a wavefront $W(e, f)$ is propagated for distance $2|f|$ beyond f, it intersects only a constant number of

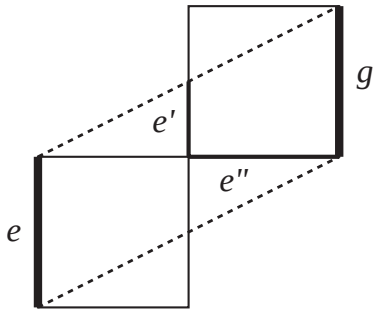


FIG. 5.1. *$W(e, g)$ may reach g via multiple paths.*

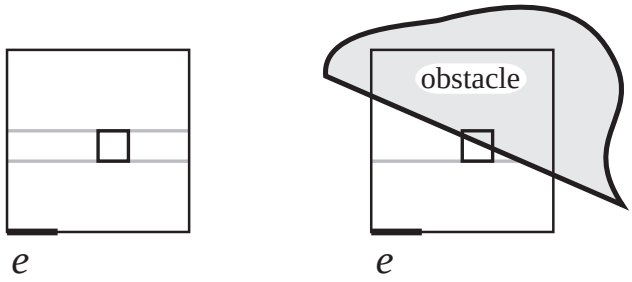


FIG. 5.2. *Preparing nonconvex cells for wave propagation.*

cells of the conforming subdivision of the free space.

The edges of the conforming subdivision $\mathcal{S}'$ already satisfy the propagation invariant, since each edge f is well-covered with parameter 2. However, we need to be more careful in dealing with a cell derived from an annulus. We subdivide each of the newly added, nonconvexity-removing edges into $O(1)$ pieces, each no longer than the edges of $\mathcal{S}$ on the annulus's outer boundary (one-eighth the side length of the outer square, by the uniform edge property of the conforming subdivision). Let H denote the convex hull of e and the inner square of the annulus. If H intersects a newly added edge f, then we further partition $f \cap H$ into pieces no longer than the inner boundary's edges (one-eighth the side length of the inner square). We illustrate this last step in Figure 5.3. Because f is parallel to e, and the inner boundary of the annulus is well separated from the outer boundary (cf. the minimum clearance property of the conforming subdivision $\mathcal{S}$), the total length of edges inside H is proportional to the side length of the inner square. It follows that the partition step creates only $O(1)$ edges.

The subdivided edges satisfy the propagation invariant: for any such edge f, let g' be an edge of c such that $W(e, f)$ leaves c by passing through g'. Edge g' is an edge of $\mathcal{S}'$, the conforming subdivision of free space; it is a fragment of an edge g of $\mathcal{S}$, the conforming subdivision for the vertices. The construction of $\mathcal{S}'$ in Lemma 2.2 ensures that there are $O(1)$ cells of $\mathcal{S}'$ within shortest path distance $2|g|$ of g'. The subdivision of nonconvexity-removing edges guarantees that $|f| \leq |g|$, which implies that the propagation invariant holds for edge f.

5.2.2. Simulating the wavefront propagation across convex cells. So far we have used a wavefront in its static form, namely, as a sequence of generators whose

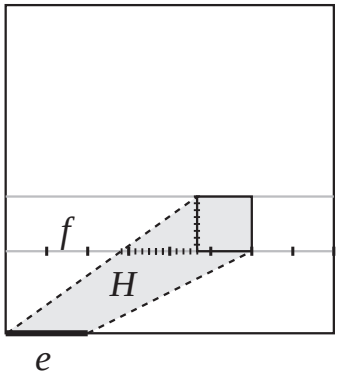


FIG. 5.3. *Subdividing the added edges.*

bisectors intersect an edge in the subdivision. We now describe a dynamic form of the wavefront, in which we track changes in the combinatorial structure of the wavefront as it sweeps across a cell. In particular, we simulate the evolution of a wavefront $W(e)$ as it sweeps across a cell c after entering it through the edge e; the cell c is a convex cell satisfying the propagation invariant. Our simulation detects and processes any bisector events involving the generators of $W(e)$ that may occur inside c. Events are processed in order of increasing distance from s, that is, in simulation time order. Generators are marked as events are processed, though the description below does not necessarily itemize all the marks made.

Let W denote the current dynamic wavefront at any time during the simulation. At the start of the simulation, we have $W = W(e)$, the approximate wavefront that passes through e—it is a list of generators, each claiming some portion of e. Every generator $v \in W$ defines a pair of bisectors with its neighbors in the list. If v is the first generator in the list, then its first bisector is the ray from v through the endpoint of e at v's end of the list; the last bisector for the last generator is defined similarly. If v is an endpoint of e, then there is no first bisector (or last bisector, as appropriate).

To process bisector events in order, we maintain the corresponding generators of W in a priority queue. The *priority* of a generator v is the weighted distance to the point at which the two bisectors defined by v intersect *beyond* e; the priority is infinite if the bisectors do not intersect beyond e. Specifically, if the bisectors defined by v and its neighbors intersect ahead of e, either in c or beyond it, at a point p, then $priority(v) = |\overline{vp}| + d(v, s)$. Our simulation of the wavefront propagation processes these bisector events in order of increasing priority *up to some maximum priority* t_{stop}, which is determined by the shape of c, as explained below. This limit t_{stop} is the minimum of individual $t_{stop}(f)$ values for each transparent edge f on c. Initially, we set $t_{stop} = \infty$ and $t_{stop}(f) = \infty$ for all f. We also initialize an empty set T, which is used to hold generators whose priorities need to be reset after the simulation.

At each step of the simulation, we extract the event with minimum priority from the queue; let v be the generator vertex producing this event. If the event occurs inside c (that is, the intersection point corresponding to the event lies in c), then we delete v from the generator list and recompute the priorities of its neighbors. We mark v in $W(e)$ for the cell c; in addition, we also mark v for a constant number of cells near c to satisfy Rule 3 of section 4.1.2.

If, however, v's event occurs outside c, then we set $priority(v) = \infty$, and add v

to the set T. The generator list is not changed in this case, because we have found the correct intersection between the boundary of c and the wavelet from v, at least locally. If we were to process all the bisector events of W in strict time order, the generators on either side of v might participate in further bisector events outside c before the last bisector event inside c occurred. However, we are not interested in those events now. Setting $priority(v)$ to ∞ avoids processing those events outside c.

We compute the intersection points of the two bisectors defined by v with the boundary of c. If either intersection lies on an opaque edge, or if they lie on different transparent edges with an opaque edge between, mark the generator v for cell c and $O(1)$ neighbors to satisfy Rule 4 of section 4.1.2. If either of the intersection points, say x, lies on a transparent edge, say f, then we update t_{stop} as follows:

$$t_{stop}(f) = \min(t_{stop}(f),\ d(v,s) + |\overline{vx}| + |f|),$$
$$t_{stop} = \min(t_{stop},\ t_{stop}(f)).$$

The second term of the minimum in the first line above is a time at which the wavefront W certainly will have swept over f; it is also no more than $2|f|$ greater than the time at which the wavefront W first contacts f.

When we reach priority t_{stop}, either $t_{stop} = \infty$ and all events inside c have been processed, or $t_{stop} < \infty$ and there is a transparent edge f on the boundary of c with $t_{stop}(f) = t_{stop}$. The definition of $t_{stop}(f)$ ensures that all the bisector events needed to produce $W(e, f)$ have been processed. We compute the static wavefront $W(e, f)$ from the current dynamic wavefront W, as follows. We first locate the endpoints of f in W by searching outward from one of the bisectors in W that intersects f—there is at least one such bisector. At this point we mark the endpoint-claiming generators to satisfy Rule 2. We split the current generator list at the endpoints of f; this breaks up the wavefront into three parts: one that passes through f (in fact, once the generator priorities are reset, this part becomes $W(e, f)$), and the other two that pass on the left and right of f. We continue with the simulation process on the latter two pieces independently, after we have reset t_{stop} in each piece to be the minimum of $t_{stop}(g)$ over the transparent edges g in that piece.

If we stop because $t_{stop} = \infty$, we split the current generator list at all the transparent edge endpoints, producing $W(e, f)$ for each transparent edge f, plus some wavefront pieces that hit only opaque edges.

If no transparent edges remain in some piece, all bisectors in the piece hit an opaque edge. We mark all the generators in that piece for cell c and in $O(1)$ nearby cells to satisfy Rule 4, as well as making all necessary marks for Rules 2 and 3.

When we finish, we reset the priority of each vertex in the temporary set T, based on the bisectors it defines with its neighbors in the (new) list. This ensures that each wavefront fragment $W(e, f)$ has its priorities set properly.

Once we have computed the wavefront $W(e, f)$, we determine the time of first contact between this wavefront and each endpoint of f. Each endpoint p lies in the region claimed by some $v \in W(e)$; v is the first or last generator in $W(e, f)$. The time of first contact is $d(v, s) + |\overline{vp}|$. (Because of visibility constraints, p may not be claimed by any generator in $W(e)$; recall that $W(e, f)$ is constrained to reach f by paths passing through e and contained in $\mathcal{U}(f)$. In this case the time of first contact is infinite.)

The propagation algorithm performs $O(1)$ priority queue and list operations per bisector event processed, plus $O(1)$ per edge of the conforming subdivision. Each operation takes $O(\log n)$ time and space. Because the wavefront data structure is

fully persistent, all the modifications to a single wavefront list $W(e)$ are independent: for example, a wavefront $W(e, f)$ may share generators with a wavefront $W(e, g)$, for $f, g \in \mathit{output}(e)$, but that overlap causes no problems.

We summarize the main result of the preceding discussion in the following lemma.

LEMMA 5.2. *Every bisector event processed in the procedure above either* (1) *lies inside* c, (2) *involves a generator whose region is truncated by an opaque edge of* c, (3) *is associated with* $t_{stop}(f)$ *being set to a finite value for the first time for some transparent edge* f *of* c, *or* (4) *lies within shortest path distance* $2|f|$ *of a transparent edge* f *of* c. *If the number of events is* m, *then the procedure takes* $O(m \log n)$ *time.*

As argued in the proof of Lemma 4.11, our simulation of the wavefront propagation discovers a bisector event for a generator v within a constant number of cells of a true bisector event for v in the shortest path map $SPM(s)$. By the propagation invariant, the bisector events processed during the propagation of a wavefront $W(e)$ across a cell c lie within a constant number of cells near the edge e (cf. Lemma 5.2 (4)). We conclude that a generator v is marked for a constant number of cells in the vicinity of each of the true bisector events involving v. Thus, the total number of events processed and generators marked during the wavefront propagation is $O(n)$. This concludes the proof of our main result.

THEOREM 5.3. *Let* $\mathcal{O}$ *be a family of polygonal obstacles in the plane with pairwise disjoint interiors and a total of* n *vertices. Given a point* s, *we can construct the shortest path map from* s *with respect to* $\mathcal{O}$ *in time* $O(n \log n)$ *and space* $O(n \log n)$.

The shortest path map $SPM(s)$ can be preprocessed for point location, after which a shortest path query from s to any point t in the plane can be answered in time $O(\log n)$ [10, 16]. A shortest path $\pi(s, t)$ can be computed in additional time $O(k)$, where k is the number of edges in the path.

6. Constructing a conforming subdivision. This section contains the proof of Theorem 2.1. It gives an algorithm to construct an α-conforming subdivision for a set V of n points in the plane. The main part of the algorithm constructs a 1-conforming subdivision of size $O(n)$ in $O(n \log n)$ time. The following lemma shows how to transform this subdivision into an α-conforming subdivision of size $O(\alpha n)$ in $O(\alpha n)$ additional time.

LEMMA 6.1. *Let* V *be a set of* n *points, and let* $\mathcal{S}_1$ *be a* 1*-conforming subdivision for* V *of size* $O(n)$. *For any* $\alpha > 1$, *we can build an* α*-conforming subdivision* $\mathcal{S}_\alpha$ *for* V *with complexity* $O(\alpha n)$ *in time* $O(\alpha n)$. *If* $\mathcal{S}_1$ *is a strong* 1*-conforming subdivision, then* $\mathcal{S}_\alpha$ *is a strong* α*-conforming subdivision.*

Proof. Subdivide each edge of $\mathcal{S}_1$ into $\lceil \alpha \rceil$ equal-length pieces. Define the well-covering region of each edge e in $\mathcal{S}_\alpha$ to be the same as the well-covering region in $\mathcal{S}_1$ of the edge of $\mathcal{S}_1$ of which e is a fragment. These operations can be performed in $O(\alpha n)$ time. We show below that the subdivision thus defined satisfies properties (C1)–(C3). Text in [brackets] applies if $\mathcal{S}_1$ is strongly 1-conforming.

- (C1) $\mathcal{S}_\alpha$ has the same set of cells as $\mathcal{S}_1$, so each cell of $\mathcal{S}_\alpha$ contains at most one point of V in its closure.
- (C2) Each internal edge e_α of $\mathcal{S}_\alpha$ is well-covered with parameter α, since it satisfies conditions (W1), (W2), and (W3) [(W3′)]. Let e_1 be the edge of $\mathcal{S}_1$ of which e_α is a fragment. Let $\mathcal{C}_\alpha(e_\alpha)$ be the set of cells of $\mathcal{S}_\alpha$ whose union $\mathcal{U}_\alpha(e_\alpha)$ is the well-covering region of e_α. Define $\mathcal{C}_1(e_1)$ and $\mathcal{U}_1(e_1)$ analogously.
  - (W1) $\mathcal{U}_\alpha(e_\alpha)$ covers the same area as $\mathcal{U}_1(e_1)$, so e_α is contained in its interior.
  - (W2) Each edge of each cell in $\mathcal{C}_1(e_1)$ is divided in $\lceil \alpha \rceil$ pieces in $\mathcal{C}_\alpha(e_\alpha)$, so the total complexity of $\mathcal{C}_\alpha(e_\alpha)$ is $O(\alpha)$.

(W3) [(W3′)]
Let f_α be an edge of $\mathcal{S}_\alpha$ on [or outside] the boundary of $\mathcal{U}_\alpha(e_\alpha)$, and let f_1 be the edge of $\mathcal{S}_1$ from which it is derived. The Euclidean distance between e_α and f_α is at least as large as the distance between e_1 and f_1, which is at least $\max(|e_1|, |f_1|) \geq \max(\alpha|e_\alpha|, \alpha|f_\alpha|)$.

(C3) Well-covering regions in $\mathcal{S}_\alpha$ are the same as in $\mathcal{S}_1$, so each contains at most one vertex of V.

This establishes the lemma. □

Before we describe the construction of the 1-conforming subdivision, we need a few definitions.

6.1. The *i*-boxes and *i*-quads. We fix a Cartesian coordinate system in the plane. For any integer i, an ith-order grid in this coordinate system is the arrangement of all lines $x = k2^i$ and $y = k2^i$, where k ranges over all integers. Each face of this grid is a square of size $2^i \times 2^i$, whose lower-left corner lies at a point $(k2^i, l2^i)$ for a pair of integers k, l. We call each such face an *i-box*.

Any 4×4 array of i-boxes is called an *i-quad*. Though an i-quad has the same size as an $(i+2)$-box, it is not necessarily an $(i+2)$-box because it may not be a face in the $(i+2)$-order grid. The four nonboundary i-boxes of an i-quad form its *core*; that is, the core of an i-quad is a 2×2 array of i-boxes. Observe that an i-box b may have up to four i-quads that contain b in their cores.

Our algorithm for building a 1-conforming partition of the point set V is a bottom-up procedure. The algorithm simulates a growth process in which we grow a square box around each data point, until the entire plane is covered by these boxes. The simulation works in discrete *stages* numbered $-2, 0, 2, 4, \ldots$. It produces a subdivision of the plane into orthogonal cells. The key object associated with a data point p in stage i is an i-quad containing p in its core. In fact, the following stronger condition holds inductively: each $(i-2)$-quad constructed in stage $(i-2)$ lies in the core of some i-quad constructed in stage i.

In each stage, we maintain only a minimal set of quads. The set of quads in stage i is denoted $\mathcal{Q}(i)$. This set is partitioned into equivalence classes under the transitive closure of the *overlap* relation—two quads q and q' are in the same equivalence class if there is a sequence of quads $q = q_0, q_1, \ldots, q_m = q' \in \mathcal{Q}(i)$ such that q_j and q_{j+1} overlap (have a common interior point) for all $j = 0, 1, \ldots, m-1$. Let $\{S_1(i), \ldots, S_k(i)\}$ denote the partition of $\mathcal{Q}(i)$ into equivalence classes in the ith stage, and let $\equiv_i$ denote the equivalence relation.

The region of the plane covered by quads in one class of this partition is called a *component*. Each component in stage i is either an i-quad or the union of i-quads. We can classify each component as either a *simple* component or a *complex* component. A component at stage i is simple if (1) its outer boundary is an i-quad and (2) it contains exactly one $(i-2)$-quad of $\mathcal{Q}(i-2)$ in its interior. Otherwise, the component is complex.

6.2. The invariants. As the algorithm progresses, we draw the boundaries of certain components. Each boundary edge is a straight line segment, parallel to one of the axes, and together these edges subdivide the plane into orthogonal cells. The critical property of our subdivision is the following *conforming property*:

> **Invariant 1:** For any edge e and cell c of the subdivision, c has an interior point within distance $|e|$ of e if and only if c and e are

incident (their closures intersect). Thus there are at most six cells within distance $|e|$ of any edge e.

Our algorithm draws edges of increasing lengths, and so we never need to subdivide previously drawn edges inside a component. In order to help maintain Invariant 1, we will also enforce the following auxiliary invariant.

> **Invariant 2:** The boundary of each complex component in stage i is subdivided into edges of length 2^i that are aligned with the ith-order grid.

Our algorithm does not actually draw the outer boundary of a simple component until just before it merges with another component to form a complex component. Indeed, this is crucial to ensure that the final subdivision has only $O(n)$ size, and not $\Theta(n \log A)$, where A is the maximum aspect ratio of a triangle in the Delaunay triangulation of the input points [3].

There are two main parts to our algorithm—one involves growing the $(i-2)$-quads of stage $(i-2)$ to i-quads of stage i, and the other involves computing and maintaining the equivalence classes and drawing subdivision edges to satisfy Invariants 1 and 2. These tasks are performed by procedures *growth* and *build-subdivision*, respectively. We postpone the discussion of *growth* till later, but introduce the necessary terminology to allow us to describe *build-subdivision*.

Given an i-quad q, $\mathit{growth}(q)$ is an $(i+2)$-quad containing q inside its core. For a family S of i-quads, $\mathit{growth}(S)$ is a minimal set of $(i+2)$-quads satisfying the following:

$$\forall\, q \in S,\ \exists\, \bar{q} \in \mathit{growth}(S) \quad \text{s.t.} \quad \bar{q} = \mathit{growth}(q).$$

As mentioned earlier, up to four $(i+2)$-quads may qualify for the role of $\mathit{growth}(q)$. We will describe later how the procedure *growth* chooses $\mathit{growth}(q)$, but for now we will use $\mathit{growth}(q)$ as a unique $(i+2)$-quad returned by the procedure *growth*. We also use the notation $\bar{q}$ to denote $\mathit{growth}(q)$.

6.3. Details of build-subdivision. By proper scaling and translation of the plane, we assume that either the horizontal or the vertical distance between any two points in V is at least 1, and no point coordinate is a multiple of $1/4$. For every point $p \in V$, we compute a (-2)-quad with p in the upper-left (-2)-box of its core; this choice ensures that quads of different points are disjoint. These quads form the initial set of quads $\mathcal{Q}(-2)$—each quad in $\mathcal{Q}(-2)$ forms its own singleton component under the equivalence class in stage -2. We regard all quads in $\mathcal{Q}(-2)$ as simple components. We draw a (-2)-box around each point p. Each of these (-2)-boxes is contained in the core of its (-2)-quad. (The (-2)-quads are *not* drawn.) Invariants 1 and 2 are both clearly satisfied at this stage. The pseudo-code below describes the details of the algorithm *build-subdivision*. This pseudo-code is correct, but not particularly efficient; an efficient implementation is presented in section 6.5.

ALGORITHM *build-subdivision*

while $|\mathcal{Q}(i)| > 1$ **do**

1. Increment i : $\quad i = i + 2$.
2. ($*$ *Compute* $\mathcal{Q}(i)$ *from* $\mathcal{Q}(i-2)$. $*$)
   (a) Initialize $\mathcal{Q}(i) = \emptyset$.
   (b) **for** each equivalence class S of $\mathcal{Q}(i-2)$ **do**
   $\mathcal{Q}(i) = \mathcal{Q}(i) \cup \mathit{growth}(S)$.
   (c) **for** every pair of i-quads $q, q' \in \mathcal{Q}(i)$ **do**
   if $q \cap q' \neq \emptyset$, set $q \equiv_i q'$.
   (d) Extend $\equiv_i$ to an equivalence relation by transitive closure, and compute the equivalence classes.
3. ($*$ *Process simple components of* $\equiv_{i-2}$ *that are about to merge with some other component.* $*$)
   for each $q \in \mathcal{Q}(i-2)$ **do**
   (a) Let $\bar{q} = \mathit{growth}(q)$ as computed in Step 2.
   (b) **if** q *is a simple component of* $\mathcal{Q}(i-2)$
   but $\bar{q}$ *is not a simple component of* $\mathcal{Q}(i)$ **then**
   Draw the boundary box of q and subdivide each of its sides into four edges at the $(i-2)$-order grid lines.
4. ($*$ *Process complex components.* $*$)
   for each equivalence class S of $\mathcal{Q}(i)$ **do**
   Let $S' = \{q \in \mathcal{Q}(i-2)$ s.t. $\mathit{growth}(q) \in S\}$.
   if $|S'| > 1$ **then** ($*$ S *is complex* $*$)
   (a) Let $R_1 = \cup_{q \in S'}$ {the core of $\mathit{growth}(q)$}.
   (b) Let $R_2 = \cup_{q \in S'}$ {the region covered by q}.
   (c) Draw $(i-2)$-boxes to fill the region between the boundaries of R_1 and R_2.
   (d) Draw i-boxes to fill the region between the boundaries of R_1 and S; break each cell boundary with an endpoint incident to R_1 into four edges of length 2^{i-2}, to satisfy Invariant 1.

endwhile

LEMMA 6.2. *The subdivision computed by the algorithm build-subdivision satisfies Invariants* 1 *and* 2.

Proof. We prove by induction that the invariants hold inside the family of quads $\mathcal{Q}(i)$, for all i. The initial family of quads $\mathcal{Q}(-2)$ clearly satisfies the two invariants. We show that no step of the algorithm *build-subdivision* ever violates these invariants. Step 2 computes $\mathit{growth}(S)$ for each equivalence class of $\mathcal{Q}(i-2)$ and then computes $\mathcal{Q}(i)$. No new edges are drawn in this step.

The only edges drawn in Step 3 are on the boundaries of simple components. Let q be an $(i-2)$-quad that is a simple component of $\mathcal{Q}(i-2)$. By definition, the single $(i-4)$-quad of $\mathcal{Q}(i-4)$ contained in q lies in its core and thus is separated from the outer boundary of q by a gap of at least 2^{i-2} on all sides. Hence the edges already drawn in the core satisfy Invariant 1: they have length no more than 2^{i-2} (actually

2^{i-4}, except when $i = 0$), and are separated from the boundary of q by a gap of at least 2^{i-2}. We draw the boundary of q in Step 3; since any previously drawn edges within q lie in its core, the new edges satisfy Invariant 1. Invariant 2 holds vacuously.

Step 4 subdivides the region covered by each complex component S. Again, the boundary of S is separated from any components of $\mathcal{Q}(i-2)$ contained in it by a gap at least the width of an i-box. Step 4(c) adds $(i-2)$-boxes to pad the region covered by $\mathcal{Q}(i-2)$ out to the boundaries of i-boxes. By Invariant 2, the newly drawn boxes satisfy Invariant 1 with respect to the previously drawn edges; they clearly satisfy Invariant 1 with respect to each other's edges. Step 4(d) packs the area between the core and the boundary of S with i-boxes, and breaks the segments incident to previously drawn cells into four pieces to guarantee Invariant 1 with respect to those cells. (The previously drawn edges on the core boundary have length 2^{i-2}, so by induction the cells incident to them have side lengths at least 2^{i-2}. It follows that the cells inside the core satisfy Invariant 1 with respect to the newly drawn segments of length 2^{i-2}.) The segments on the boundary of S are unbroken, so Invariant 2 holds at the next stage of the algorithm. This completes the proof. □

LEMMA 6.3. *The subdivision produced by build-subdivision has size $O(n)$.*

Proof. We show that the algorithm draws a linear number of edges altogether. The number of edges drawn in Step 3 is proportional to the number drawn in Step 4—we draw a constant number of edges in Step 3 for each simple component that merges to form a complex component at the next stage. The number of edges drawn in Step 4 for a complex component S is $O(|S'|)$, the number of $(i-2)$-quads whose growths constitute S. The key observation in proving the linear bound is that the total size of $\mathcal{Q}$ decreases every two stages by an amount proportional to the total number of quads in complex components. This fact, which we prove in the next subsection (Lemma 6.5), can be expressed as follows: If e_i edges are drawn in stage i, then

$$|\mathcal{Q}(i+2)| \;\leq\; |\mathcal{Q}(i-2)| \;-\; \Theta(e_i).$$

That is, there exists an absolute constant β such that

$$\beta e_i \;\leq\; |\mathcal{Q}(i-2)| - |\mathcal{Q}(i+2)|.$$

If we sum this inequality over all even $i \geq 0$, the right-hand side telescopes, and we obtain

$$\beta \sum_i e_i \;\leq\; |\mathcal{Q}(-2)| + |\mathcal{Q}(0)| - 2.$$

Since $|\mathcal{Q}(-2)| = n$, we have $\sum_i e_i \leq (2n-2)/\beta$. The total number of edges in the subdivision is $O(n)$. □

LEMMA 6.4. *The subdivision that build-subdivision produces is strongly* 1*-conforming and satisfies the following additional properties:* (1) *all edges of the subdivision are horizontal or vertical,* (2) *each face is either a square or a square-annulus (with subdivided boundary),* (3) *each annulus has the minimum clearance property,* (4) *each face has the uniform edge property, and* (5) *every point of V is contained in a square face.*

Proof. Strong 1-conformity is a consequence of Invariant 1, as we now show. Condition (C1) is trivially true, since each point is initially enclosed by a square. To establish well-covering (Condition (C2)), let $I(e)$ be the union of the (at most six) cells incident to an edge e. By Invariant 1, the distance from e to any edge outside or

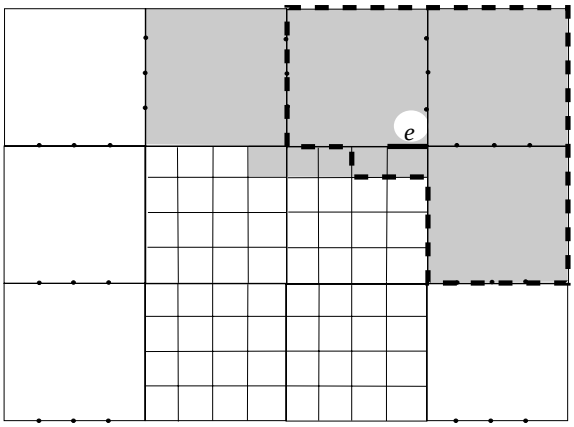


FIG. 6.1. *A well-covering region $\mathcal{U}(e)$. The boundary of $I(e)$ is shown dashed.*

on the boundary of $I(e)$ is at least $|e|$. Edge e may be collinear with other edges of the two cells on whose boundary it lies. We define $\mathcal{C}(e)$ to be the set of cells incident to any of these collinear edges; $\mathcal{U}(e)$, the union of these cells, is a superset of $I(e)$. See Figure 6.1. Because the two cells with e as a boundary edge meet only along edges collinear with e, this definition of $\mathcal{U}(e)$ means that for any edge f on or outside the boundary of $\mathcal{U}(e)$, $I(f)$ does not contain both cells incident to e. But this implies, by Invariant 1, that e is on or outside the boundary of $I(f)$, and hence the distance from e to f is at least $|f|$. Edge e certainly lies in the interior of $\mathcal{U}(e)$ (Condition (W1)). Condition (W2) follows because $\mathcal{C}(e)$ is the union of $I(e')$ for $O(1)$ edges e' collinear with e, $|I(e')| \leq 6$ for each e', and each cell has constant complexity. As noted above, the minimum distance between e and any edge f on or outside the boundary of $\mathcal{U}(e)$ is at least $\max(|e|, |f|)$, which establishes Condition (W3′). Condition (C3) follows from the observation that a well-covering region $\mathcal{U}(e)$ includes a vertex v of V if and only if e is an edge of the square containing v. This is because each vertex-containing square is the inner square of a square-annulus in the subdivision. No edge belongs to two such squares, so Condition (C3) holds.

Properties (1)–(5) hold by construction. This completes the proof. □

6.4. The algorithm *growth*(). In this subsection, we describe our algorithm for computing $growth(S)$ for a set of i-quads S, and prove that the number of quads decreases every two stages by an amount proportional to the total complexity of the complex components. Let $S \subset \mathcal{Q}(i)$ be a set of i-quads forming a complex component under the equivalence relation $\equiv_i$. Recall that $growth(S)$ is a minimal set of $(i+2)$-quads such that each i-quad of S lies in the core of some $(i+2)$-quad in $growth(S)$. We will show that

$$|growth(growth(S))| \;\leq\; \kappa|S|,$$

for an absolute constant $0 < \kappa < 1$. The pseudo-code below describes an unoptimized version of our algorithm for computing $growth(S)$. The algorithm works by building a graph on the quads in S.

In this algorithm, Step 1 builds a graph whose nodes are the i-quads of S; two quads q_1 and q_2 have an edge between them if their union $q_1 \cup q_2$ lies in some 2×2 array of $(i+2)$-boxes. The maximum node degree of this graph is $O(1)$ since only a

ALGORITHM *growth*(S)

0. Set $growth(S) = \emptyset$.
1. **for** each pair of quads $q_1, q_2 \in S$ **do**
   if $q_1 \cup q_2$ can be contained in a 2×2 array of $(i+2)$-boxes, **then**
   Put an edge between q_1 and q_2.
2. Compute a maximal matching in the graph computed in Step 1.
3. **for** each edge (q_1, q_2) in the maximal matching **do**
   Choose an $(i+2)$-quad $\bar{q}$ containing q_1, q_2 in its core.
   Set $growth(q_1) = growth(q_2) = \bar{q}$, and add $\bar{q}$ to $growth(S)$.
4. **for** each unmatched quad $q \in S$ **do**
   Set $growth(q) = \bar{q}$, where $\bar{q}$ is an $(i+2)$-quad containing q in its core.
   Add $\bar{q}$ to $growth(S)$.

constant number of i-quads can touch any i-quad q. Thus, a maximal matching in this graph has $\Theta(|E|)$ edges. Each i-quad at stage i maps to an $(i+2)$-quad at stage $(i+2)$. Since each matching edge corresponds to two i-quads that map to the same $(i+2)$-quad, it clearly follows that

$$|growth(S)| \;=\; |S| - \Theta(|E|).$$

The crucial fact to prove is that $|E|$ is a constant fraction of $|S|$ at stage $(i+2)$.

LEMMA 6.5. *Let $S \subset \mathcal{Q}(i)$ be a set of two or more i-quads such that $growth(S)$ is a complex component under the equivalence relation $\equiv_{i+2}$. Then $|growth(growth(S))| \leq \kappa|S|$, for an absolute constant $0 < \kappa < 1$.*

Proof. We show that either $|growth(S)| < (3/4)|S|$, or at least half of the quads of $growth(S)$ can be contained in a 2×2 array of $(i+2)$-boxes with some other quad of $growth(S)$.

If $|growth(S)| < (3/4)|S|$, then we are done, because the following inequality obviously holds: $|growth(growth(S))| \leq |growth(S)| \leq (3/4)|S|$. Therefore, suppose that $|growth(S)| \geq (3/4)|S|$. Then at least half the i-quads of S are not matched in Step 2 of the function *growth*(), and their growths contribute more than half of the $(i+2)$-quads of $growth(S)$. Consider one such i-quad $q \in S$. Since S is a nonsingleton equivalence class, there exists another i-quad $q' \in S$ that overlaps q. Let $\bar{q} = growth(q)$ and $\bar{q}' = growth(q')$. By assumption, $\bar{q} \neq \bar{q}'$. The cores of $\bar{q}$ and $\bar{q}'$ both contain the overlap region $q \cap q'$, so the cores must overlap. Therefore both cores are contained within a 3×3 array of $(i+2)$-boxes, and both the $(i+2)$-quads $\bar{q}$ and $\bar{q}'$ are contained within a 5×5 array of $(i+2)$-boxes. This ensures that $\bar{q}$ and $\bar{q}'$ are joined by an edge in the graph of $growth(S)$: any two $(i+2)$-quads whose bounding box is contained in a 5×5 array of $(i+2)$-boxes can be covered by a 2×2 array of $(i+4)$-boxes. Hence the number of edges in the maximal matching of $growth(S)$ is $\Omega(|S|)$, which proves the inequality $|growth(growth(S))| \leq \kappa|S|$ for some $\kappa < 1$. □

Since the number of edges drawn at stage i, call it e_i, is proportional to the number of $(i-2)$-quads whose growths belong to complex components, the preceding lemma establishes the earlier claim that

$$|\mathcal{Q}(i+2)| \;\leq\; |\mathcal{Q}(i-2)| \;-\; \Theta(e_i).$$

For any $q, q' \in S$, we have $growth(q) = growth(q')$ only if q and q' are touching—their closures intersect—otherwise q and q' cannot be contained in the core of the

same $(i+2)$-quad. We use this fact to implement the procedure *growth*(S) to run in time $O(|S| \log |S|)$: each quad of S touches at most a constant number of other quads, and we can compute which quads touch using an $O(|S| \log |S|)$ plane sweep algorithm [22]. From the set of touching pairs we can compute the graph edges in Step 1 of *growth*(S) in $O(|S|)$ additional time. All other steps of *growth*(S) take time proportional to the graph size, which is $O(|S|)$.

6.5. An $O(n \log n)$ implementation of build-subdivision. In order to keep the time complexity of *build-subdivision* independent of the aspect ratio of the points, we process a simple component only when it is about to merge with another component. In other words, the amount of processing is proportional to the number of boundary edges drawn at any stage. Except for Step 2(b), which computes growths, and Step 2(c), which detects overlapping i-quads, all other steps can be implemented to run in time proportional to the number of edges drawn in the subdivision. (Steps 3 and 4 use the adjacency information computed in Step 2(c) to run in linear time.)

We maintain the simple components and the complex components of $\mathcal{Q}(i)$ in two separate sets. We compute *growth*(S) explicitly for the complex components, but only implicitly for the simple components. Suppose that q is a singleton component of $\mathcal{Q}(i-2j)$, and *growth*$^j(q) \in \mathcal{Q}(i)$ is the result of applying the *growth*() operator j times. If *growth*$^k(q)$ is simple for all positive $k \leq j$, then *growth*$^j(q)$ can be determined in constant time from q using the floor operation. The set of simple components of $\mathcal{Q}(i)$ is maintained as a set of singletons from earlier stages; when we determine that a simple component is about to merge with another component (Step 2(c)), we compute the simple component explicitly. The transitive closure can be computed in time proportional to the total size of the complex components, which is proportional to the number of edges drawn at this stage. Since the final subdivision has size $O(n)$, all the work except that in Steps 2(b) and 2(c) takes a linear amount of time. In the following we show how to use a minimum spanning tree algorithm to implement Step 2(c) in $O(n \log n)$ time.

6.5.1. The merging of i-quads. Before we present the algorithm, we discuss the distance properties satisfied by points that lie in the same equivalence class in stage i. We say that a quad q is a *containing i-quad* of a point $u \in V$ if $q \in \mathcal{Q}(i)$ and u lies in q's core. A point u *belongs* to an equivalence class $S \in \mathcal{Q}(i)$ if there is a containing i-quad of u in S.

LEMMA 6.6. *Let u be a point of V and let $q \in \mathcal{Q}(i)$ be a containing i-quad of u. Then the minimum L_∞ distance between u and the outer boundary of q is 2^i.*

Proof. The lemma depends on the property that u lies in the core of q. Since q has side length 2^{i+2}, and u lies at least a quarter of this distance away from the outer boundary, the lemma follows. □

In the following, the notation $d_\infty(u, v)$ denotes the distance between the points u and v under the L_∞ norm.

LEMMA 6.7. *Let u and v be two points of V that belong to different equivalence classes of $\mathcal{Q}(i)$. Then $d_\infty(u, v) > 2 \times 2^i$.*

Proof. Let q_u and q_v be two containing i-quads for u and v, respectively. Since u and v lie in different equivalence classes, these i-quads do not intersect. By Lemma 6.6, each of the points lies at least a distance 2^i away from the outer boundaries of their i-quads, which immediately gives the lower bound on $d_\infty(u, v)$ stated in the lemma. □

LEMMA 6.8. *Let $u, v \in V$ be two points and let q_u, q_v, respectively, be two i-quads of $\mathcal{Q}(i)$ containing them. If $q_u \cap q_v \neq \emptyset$, then $d_\infty(u, v) < 6 \times 2^i$.*

Proof. By Lemma 6.6, the maximum distance between u and the outer boundary of q_u is at most 3×2^i. The same holds for v and q_v, which implies the upper bound on $d_\infty(u, v)$. □

6.5.2. Minimum spanning trees. Let V_S be the set of points in the core of some component $S \in \mathcal{Q}(i)$. Our implementation of *build-subdivision* is based on the observation that the longest edge of the L_∞ minimum spanning tree of V_S has length less than 6×2^i. To make this observation more precise, we define $G(i)$ to be the graph on V containing exactly those edges whose L_∞ length is at most 6×2^i, and define $MSF(i)$ to be the minimum spanning forest of $G(i)$.

LEMMA 6.9. *The points contained in any component of $\mathcal{Q}(i)$ belong to a single tree of $MSF(i)$.*

Proof. Let S be a component of $\mathcal{Q}(i)$. By Lemma 6.8, the points contained in S can be linked by a tree with edges shorter than 6×2^i. For any bipartition of the points of V_S, the minimum weight edge linking the two subsets is shorter than 6×2^i. The minimum spanning tree of V_S has all edges shorter than 6×2^i, and therefore V_S belongs to a single tree of $MSF(i)$. □

LEMMA 6.10. *If i-quads q_1 and q_2 belong to different components of $\mathcal{Q}(i)$, then their points belong to different trees of $MSF(i-2)$.*

Proof. Every edge from a point in q_1's component to any point outside that component has length greater than 2×2^i, by Lemma 6.7. The points of quads q_1 and q_2 are in the same tree of $MSF(i-2)$ only if every bipartition of V that separates the points of q_1 from those of q_2 is bridged by an edge of length less than $6 \times 2^{i-2}$. But the bipartition separating the points of q_1's component of $\mathcal{Q}(i)$ from the rest of V has bridge length greater than $2 \times 2^i > 6 \times 2^{i-2}$. □

Our algorithm is based on an efficient construction of $MSF(i)$ for all i such that $MSF(i) \neq MSF(i-2)$. The standard algorithm for computing a geometric minimum spanning tree is well suited to our needs. We compute the L_∞ Delaunay triangulation of V in $O(n \log n)$ time [6], then run Kruskal's MST algorithm [8]. Kruskal's algorithm inserts the $O(n)$ Delaunay edges into the current minimum spanning forest in sorted order from shortest to longest; any edge that joins two trees of the forest is retained, and all other edges are dropped. For each edge e added to the forest, we compute $k = 2\lceil \frac{1}{2} \log_2(|e|/6) \rceil$, which determines the stage k at which e is added to $MSF(k)$. By stopping just before each stage change, we produce $MSF(i)$ for each even i such that $MSF(i) \neq MSF(i-2)$ in $O(n \log n)$ total time.

The implementation of *build-subdivision* below replaces Steps 1 and 2 of *build-subdivision* with more efficient code based on minimum spanning trees. First, we process only stages at which something happens: $MSF(i)$ changes, or there are complex components of $\mathcal{Q}(i)$ whose *growth* computation is nontrivial. (This optimization is not usually significant; it matters only if the ratio of maximum to minimum point separation is greater than 2^n.) Second, we compute $growth(S)$ only for complex components and for simple components that will merge with another component soon, and compute the equivalence classes of $\mathcal{Q}(i)$ only for this same set of quads. Simple components that are well separated from others are not involved in the computation.

The running time of this algorithm is dominated by the $O(k \log k)$ required for a plane sweep [22] of $k = |\mathcal{Q}(i, T)|$ quads in Step 2(c–d). There are $O(k)$ quads in complex components either in $\mathcal{Q}(i, T)$ or in $\mathcal{Q}(i+2, T)$, so there are $O(k)$ edges drawn for these quads at stage i or $i+2$. We amortize this cost by charging $O(\log k)$ per edge of the subdivision, getting $O(n \log n)$ time overall. The computation of the Delaunay triangulation and the minimum spanning forest contributes a term of the

IMPLEMENTATION OF *build-subdivision*

For each $T \in MSF(i)$, maintain the corresponding set of i-quads in $\mathcal{Q}(i)$ that are the containing quads for the vertices of T. Call this set $\mathcal{Q}(i, T)$.

Initialize $i = -2$. Initialize $MSF(-2)$ to be a forest of singleton vertices. For each vertex $v \in V$, $\mathcal{Q}(-2, \{v\})$ is a singleton quad with v in its core.

Maintain a set $\mathcal{N}$ of trees in $MSF(i)$ such that for each $T \in \mathcal{N}$, $|\mathcal{Q}(i, T)| > 1$; that is, T's component is not a singleton quad. Initialize $\mathcal{N} = \emptyset$.

while $|\mathcal{Q}(i)| > 1$ **do**
  $i_{old} = i$;
  if $|\mathcal{N}| > 0$ **then** $i = i + 2$
  else Set i to the smallest even $i' > i$ such that $MSF(i') \neq MSF(i)$.
  foreach edge e of $MSF(i)$ not in $MSF(i_{old})$ **do**
    Let T_1 and T_2 be the trees linked by e.
    foreach $T_x \in \{T_1, T_2\}$ **do**
      if $T_x \in \mathcal{N}$ **then**
        Remove T_x from $\mathcal{N}$.
      else
        Compute the singleton $(i-2)$-quad in $\mathcal{Q}(i-2, T_x)$.
    Join T_1 and T_2 to get T', and put T' in $\mathcal{N}$.
    Set $\mathcal{Q}(i-2, T') = \mathcal{Q}(i-2, T_1) \cup \mathcal{Q}(i-2, T_2)$.
  end
  ($*$ *Invariant: if* $T \in \mathcal{N}$*, then* $\mathcal{Q}(i-2, T)$ *is correctly computed.* $*$)
  foreach $T \in \mathcal{N}$ **do**
    2(a) Initialize $\mathcal{Q}(i, T) = \emptyset$.
    2(b) **for** each equivalence class S of $\mathcal{Q}(i-2, T))$ **do**
      $\mathcal{Q}(i, T) = \mathcal{Q}(i, T) \cup growth(S)$.
    2(c–d) Compute the equivalence classes of $\mathcal{Q}(i, T)$ by plane sweep.
    3–4 Perform Steps 3 and 4 of *build-subdivision* on $\mathcal{Q}(i, T)$.
    if $|\mathcal{Q}(i, T)| = 1$ **then** Delete T from $\mathcal{N}$.
  end
endwhile

same asymptotic magnitude.

We have established the following lemma.

LEMMA 6.11. *Algorithm build-subdivision can be implemented to run using* $O(n \log n)$ *standard operations on a real RAM, plus* $O(n)$ *floor and base-2 logarithm operations.*

Lemmas 6.1, 6.3, 6.4, and 6.11 establish the main theorem of this section.

CONFORMING SUBDIVISION THEOREM. *For any* $\alpha \geq 1$*, every set of* n *points in the plane admits a strong* α*-conforming subdivision of* $O(\alpha n)$ *size satisfying the following additional properties:* (1) *all edges of the subdivision are horizontal or vertical,* (2) *each face is either a square or a square-annulus (with subdivided boundary),* (3) *each annulus has the minimum clearance property,* (4) *each face has the uniform edge property, and* (5) *every data point is contained in the interior of a square face. Such*

a subdivision can be computed in time $O(\alpha n + n \log n)$.

7. Extensions and concluding remarks. We have presented a worst-case optimal algorithm for the planar, Euclidean shortest path problem. Our algorithm uses the wavefront propagation method and builds a shortest path map, which can be used to answer shortest path queries from a fixed source in logarithmic time. We introduced several new ideas and techniques in order to implement the wavefront propagation optimally. Perhaps the most original contribution of our paper is the idea of a conforming subdivision—it is a quad-tree-like subdivision that seems especially useful for line segments. We expect this subdivision to find other applications in computational geometry.

Our wavefront simulation is highly "local" in the sense that all interactions among bisectors occur within "small" regions (well-covering regions). Obviously, we still require the bisectors to satisfy some global properties, such as the ones stated in Lemmas 3.2 and 3.3, but the locality of processing allows our algorithm to extend to several more general instances of the shortest path problem. These include generalizations involving the shape and number of sources. We sketch below the modifications necessary for some of these extensions.

Nonpoint sources. When the source is not a point, but rather a more complex geometric shape such as a line segment or a disk, then the initial wavelet originating from the source has a more complicated form: it is the Minkowski sum of a disk and the source. However, the intermediate generators are still just the obstacle vertices, and they generate circular wavelets. Thus, except for initialization and propagating the initial wavelets, the rest of the wavefront propagation algorithm does not change. The initialization involves computing "direct" distances to all the cells that are within a constant number of cells of the source, which can be done easily in $O(n \log n)$ time.

Multiple sources. Computing shortest paths in the presence of multiple sources is equivalent to computing a "geodesic Voronoi diagram": a partition of the free space into regions so that all points in a region have the same nearest source and the combinatorial structure of the shortest path to that source is also the same for all points in the region. To help visualize the process, we might imagine that the wavefront of each source has a distinct color; in the end, the region claimed by each source acquires the color of its source.

During the initialization, we compute direct distances between each source and the corners of its well-covering regions; if well-covering regions overlap, we use the Voronoi diagram of the sources to decide which corner is closer to which source. Again, this can be done in $O(n \log n)$ time initially. We maintain a common priority queue for all the sources, and as each obstacle vertex is claimed, it acquires the color of its claiming source. Knowing the color of each generator helps us determine whether a bisector is bounding two regions belonging to the same source or two different sources. In all other respects, the processing of bisectors in cells is the same as in the original algorithm.

Other generalizations. The ideas mentioned above also work for multiple sources with specified release times. In particular, each source has associated with it an initial "delay" and its wavelet is issued after the specified delay. The delays are easily handled by our algorithm: just add the delay time of each source to its initial priority queue entries. The rest of the algorithm proceeds as before.

Open problems. Finally, we conclude with two open problems.

1. Can the space complexity of our algorithm be reduced to linear?
2. Does our wavefront propagation method extend to the shortest path problem on the surface of a convex polytope?

Acknowledgment. We are grateful to an anonymous referee for a thoughtful and thorough review; the referee's suggestions significantly improved the presentation of our results.

REFERENCES

[1] T. Asano, *An efficient algorithm for finding the visibility polygons for a polygonal region with holes*, Trans. IECE Japan, E-68 (1985), pp. 557–559.
[2] T. Asano, L. Guibas, J. Hershberger, and H. Imai, *Visibility of disjoint polygons*, Algorithmica, 1 (1986), pp. 49–63.
[3] M. Bern, D. Eppstein, and J. R. Gilbert, *Provably good mesh generation*, in Proceedings of the 31st IEEE Symposium on Foundations of Computer Science, St. Louis, MO, 1990, pp. 231–241.
[4] B. Chazelle, *A theorem on polygon cutting with applications*, in Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, Chicago, IL, 1982, pp. 339–349.
[5] L. P. Chew, *There are planar graphs almost as good as the complete graph*, J. Comput. System Sci., 39 (1989), pp. 205–219.
[6] L. P. Chew and R. L. Drysdale, *Voronoi diagrams based on convex distance functions*, in Proceedings of the ACM Symposium on Computational Geometry, Baltimore, MD, 1985, pp. 235–244.
[7] K. L. Clarkson, *Approximation algorithms for shortest path motion planning*, in Proceedings of the 19th ACM Symposium on Theory of Computing, New York, NY, 1987, pp. 56–65.
[8] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1993.
[9] E. W. Dijkstra, *A note on two problems in connection with graphs*, Numer. Math., 1 (1959), pp. 269–271.
[10] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, *Optimal point location in a monotone subdivision*, SIAM J. Comput., 15 (1986), pp. 317–340.
[11] M. Fredman and R. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM, 34 (1987), pp. 596–615.
[12] S. K. Ghosh and D. M. Mount, *An output-sensitive algorithm for computing visibility graphs*, SIAM J. Comput., 20 (1991), pp. 888–910.
[13] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan, *Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons*, Algorithmica, 2 (1987), pp. 209–233.
[14] J. Hershberger and J. Snoeyink, *Computing minimum length paths of a given homotopy class*, Comput. Geom., 4 (1994), pp. 63–97.
[15] S. Kapoor and S. N. Maheshwari, *Efficient algorithms for Euclidean shortest paths and visibility problems with polygonal obstacles*, in Proceedings of the 4th ACM Symposium on Computational Geometry, Urbana-Champaign, IL, 1988, pp. 172–182.
[16] D. Kirkpatrick, *Optimal search in planar subdivisions*, SIAM J. Comput., 12 (1983), pp. 28–35.
[17] D. T. Lee and F. P. Preparata, *Euclidean shortest paths in the presence of rectilinear barriers*, Networks, 14 (1984), pp. 393–410.
[18] J. S. B. Mitchell, *A new algorithm for shortest paths among obstacles in the plane*, Ann. Math. Artificial Intelligence, 3 (1991), pp. 83–106.
[19] J. S. B. Mitchell, *Shortest paths among obstacles in the plane*, Internat. J. Comput. Geom. Appl., 6 (1996), pp. 309–332.
[20] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou, *The discrete geodesic problem*, SIAM J. Comput., 16 (1987), pp. 647–668.
[21] M. H. Overmars and E. Welzl, *New methods for computing visibility graphs*, in Proceedings of the 4th ACM Symposium on Computational Geometry, Urbana-Champaign, IL, 1988, pp. 164–171.
[22] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.
[23] J. Reif and J. Storer, *Shortest paths in the plane with polygonal obstacles*, J. ACM, 41 (1994), pp. 982–1012.
[24] H. Rohnert, *Shortest paths in the plane with convex polygonal obstacles*, Inform. Process. Lett., 23 (1986), pp. 71–76.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2257–2284

TIGHT LOWER BOUNDS FOR *st*-CONNECTIVITY ON THE NNJAG MODEL*

JEFF EDMONDS†, CHUNG KEUNG POON‡, AND DIMITRIS ACHLIOPTAS§

Abstract. Directed *st*-connectivity is the problem of deciding whether or not there exists a path from a distinguished node s to a distinguished node t in a directed graph. We prove a time–space lower bound on the probabilistic NNJAG model of Poon [*Proc. 34th Annual Symposium on Foundations of Computer Science*, Palo Alto, CA, 1993, pp. 218–227]. Let n be the number of nodes in the input graph and S and T be the space and time used by the NNJAG, respectively. We show that, for any $\delta > 0$, if an NNJAG uses space $S \in O(n^{1-\delta})$, then $T \in 2^{\Omega(\log^2(n/S))}$; otherwise $T \in 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$. (In a preliminary version of this paper by Edmonds and Poon [*Proc. 27th Annual ACM Symposium on Theory of Computing*, Las Vegas, NV, 1995, pp. 147–156.], a lower bound of $T \in 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$ was proved.) Our result greatly improves the previous lower bound of $ST \in \Omega(n^2/\log n)$ on the JAG model by Barnes and Edmonds [*Proc. 34th Annual Symposium on Foundations of Computer Science*, Palo Alto, CA, 1993, pp. 228–237] and that of $S^{1/3}T \in \Omega(n^{4/3})$ on the NNJAG model by Edmonds [*Time-Space Lower Bounds for Undirected and Directed ST-Connectivity on JAG Models*, Ph.D. thesis, University of Toronto, Toronto, ON, Canada, 1993]. Our lower bound is tight for $S \in O(n^{1-\delta})$, for any $\delta > 0$, matching the upper bound of Barnes et al.[*Proc. 7th Annual IEEE Conference on Structure in Complexity Theory*, Boston, MA, 1992, pp. 27–33]. As a corollary of this improved lower bound, we obtain the first tight space lower bound of $\Omega(\log^2 n)$ on the NNJAG model. No tight space lower bound was previously known even for the more restricted JAG model.

Key words. lower bounds, space–time tradeoffs, space complexity, connectivity

AMS subject classifications. 68Q15, 68Q25, 68R10, 05C20, 05C40

PII. S0097539795295948

1. Introduction.

The *st*-connectivity problem (STCON) is a fundamental problem in computer science, as it is the natural abstraction of many search processes. Its space and time–space complexities are of special interest because there are many applications such as game searching, program verification, and databases in which the size of the input graph is too large compared to the size of the internal memory of a machine. In these applications algorithms that run in small space, and preferably in small time simultaneously, are required. STCON is also important in computational complexity theory because it is complete for NSPACE($\log n$) under logarithmic space reductions. Both STCON and the corresponding problem for undirected graphs, USTCON, are hard for DSPACE($\log n$) since any problem solvable deterministically in logarithmic space can be reduced to either problem. (See Lewis and Papadimitriou [22] and Savitch [28].) Thus, showing that there is no deterministic logarithmic space algorithm for STCON that would separate the classes DSPACE($\log n$) and NSPACE($\log n$),

*Received by the editors December 13, 1995; accepted for publication (in revised form) June 2, 1997; published electronically August 3, 1999.

http://www.siam.org/journals/sicomp/28-6/29594.html

†Department of Computer Science, York University, Toronto, ON M3J 1P3, Canada (jeff@cs.yorku.ca). A major portion of this work was done while the author was at the International Computer Science Institute, Berkeley, CA.

‡Department of Computer Science, City University of Hong Kong, Hong Kong. This work was partially supported by Texas Advanced Research Projects Grant 003658386. A major portion of it was done while the author was at the Deptartment of Computer Science, University of Toronto, Canada (ckpoon@cs.cityu.edu.hk).

§Department of Computer Science, University of Toronto, Toronto, ON M5S 3G4, Canada (optas@cs.toronto.edu).

while devising such an algorithm would prove that $\mathrm{DSPACE}(f(n)) = \mathrm{NSPACE}(f(n))$ for any space-constructible function $f(n) \in \Omega(\log n)$ [28]. STCON is also a candidate problem for separating the classes of SC and NC [20]. Below we mention the previous works that are most relevant to our paper. For more information on graph connectivity, we refer the reader to the beautiful survey paper by Wigderson [31].

1.1. Previous work. The most commonly used algorithms for *st*-connectivity, breadth- and depth-first search run in optimal time $O(m+n)$ and use $O(n \log n)$ space. At the other extreme, Savitch [28] provided an algorithm that uses $O(\log^2 n)$ space and requires time exponential in its space bound (i.e., time $n^{O(\log n)}$). Tompa [30] showed that STCON cannot be solved in polynomial time and sublinear space simultaneously by the repeated squaring method. However, Barnes et al. [3] gave a polynomial time algorithm for STCON that uses space $S \in n/2^{\Theta(\sqrt{\log n})}$, providing the first polynomial time, sublinear space algorithm. This shows that the repeated squaring method is too restricted. In fact, their algorithm implies a general time–space upper bound of $T \in 2^{O(\log^2(\frac{n \log n}{S}))} \times n^3$ for $S \in \Omega(\log^2 n)$.

A natural question is whether the upper bounds of Savitch and Barnes et al. are tight. Unfortunately, proving nontrivial lower bounds for natural decision problems on any general model of computation, such as Turing machines and branching programs, appears to be beyond the reach of current techniques. Thus, it is natural to consider *structured* computational models [12] whose basic operations are based on the structure of the input, as opposed to being based on the bits in the input's encoding. A natural structured model for STCON is the "jumping automaton for graphs," or JAG, introduced by Cook and Rackoff [13]. A JAG moves a set of pebbles on the graph. There are two basic operations—moving a pebble along a directed edge in the graph and jumping a pebble from its current location to the node occupied by another pebble. Although the JAG model is structured, it is powerful enough to simulate most known algorithms for STCON and related problems. For example, depth-first and breadth-first search, random walks [1], and the algorithms of Savitch and Barnes et al. can all be simulated on a JAG (see [13, 27]). To our kowledge, all known deterministic or probabilistic algorithms for directed STCON are implementable on a JAG. However, it is not clear how a nondeterministic JAG can simulate Immerman's and Szelepcsényi's $O(\log n)$-space algorithm for directed *st*-nonconnectivity ($\overline{\text{STCON}}$) [19, 29]. This motivated Poon [26] to introduce the more general node-named JAG (NNJAG) model, an extension of the JAG, where the computation is allowed to depend on the names of the nodes on which the pebbles are located. Using this added power, Poon [26] showed how to simulate the Immerman/Szelepcsényi algorithm on a nondeterministic NNJAG.

Cook and Rackoff [13] proved a lower bound of $\Omega(\log^2 n/\log\log n)$ on the space required for a JAG to compute STCON. Within the $\log\log n$ factor, this is tight with Savitch's algorithm. Berman and Simon [7] extended this result to the probabilistic JAG model. More precisely, they showed that any probabilistic JAG that solves STCON within $2^{\log^{O(1)} n}$ expected time requires $\Omega(\log^2 n/\log\log n)$ space. Their probabilistic JAG is allowed to flip a coin in each step and is able to solve STCON with 1-sided error, using $O(\log n)$ space and $O(n^n)$ expected time (see Gill [18]). In the following, we will refer to such a probabilistic machine as a coin-flipping machine. Poon [26] further generalized the bound, showing that $S \in \Omega(\frac{\log^2 n}{\log\log n + \log\log T})$ for any coin-flipping probabilistic NNJAG with space S and expected time T.

Regarding the time–space tradeoff, there are many lower bounds proved for UST-

CON on various weaker variants of the JAG model [6, 11, 13]. Edmonds [15] was the first to prove a time–space lower bound for USTCON on the regular JAG model (with bounded space). All these results apply to (directed) STCON, which contains USTCON as a special case. However, USTCON appears to be easier than STCON both in terms of space and time–space complexity. For example, Nisan, Szemerédi, and Wigderson [24] showed that USTCON can be solved in $O(\log^{1.5} n)$ space on a deterministic Turing machine. There is also a randomized $O(\log n)$ space, polynomial time algorithm (by Aleliunas et al. [1]) and a deterministic $O(\log^2 n)$ space, polynomial time algorithm (by Nisan [23]) for this problem. Although it is not known whether the algorithms in [24, 23] can be simulated on a JAG or NNJAG, USTCON can indeed be solved in $O(\log n)$ space and polynomial time on a JAG due to the existence of polynomial length universal traversal sequences [1]. Thus, one cannot hope to get superpolynomial time lower bounds for STCON by establishing similar bounds for USTCON.

The first nontrivial lower bound explicitly for STCON was given by Barnes and Edmonds [4]. They showed that $ST \in \Omega(n^2/\log n)$ on the JAG model. In fact their result was proved on a more powerful variant of JAG called *many states, big step JAG* which, unlike an ordinary JAG, is capable of traversing trees in $O(\log n)$ space. Using a proof technique completely different from [4], Edmonds [14] showed that $S^{1/3}T \in \Omega(n^{4/3})$ on the NNJAG model. These results still do not yield superpolynomial lower bounds on time no matter how small S is. In view of this large gap between the upper and lower bounds and the fact that the Barnes et al. algorithm was obtained by combining several rather simple ideas, it seemed that further improvements to the upper bound were quite possible.

1.2. New results. Rather surprisingly, in a preliminary version of this paper by Edmonds and Poon [16], a lower bound of $T \in 2^{\Omega(\log^2(\frac{n\log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$ is obtained. This implies that superpolynomial running time is necessary to solve the problem whenever S is smaller than $(n\log n)/2^{\omega(\sqrt{\log n \cdot \log\log n})}$. The bound also nearly matches the upper bound of $T \in 2^{O(\log^2(\frac{n\log n}{S}))} \times n^3$ (which is superpolynomial for $S \in (n\log n)/2^{\omega(\sqrt{\log n})}$) by Barnes et al. [3]. Here, by a more careful choice of parameters and a tighter analysis, we prove that for any $\delta > 0$, a probabilistic NNJAG with 2-sided error, using space $S \in O(n^{1-\delta})$, requires expected time $T \in 2^{\Omega(\log^2(n/S))}$, matching the upper bound of [3].

In this paper, we define an S-space probabilistic NNJAG as a distribution of S-space deterministic NNJAGs. Hence, the probabilistic NNJAG must use time $T \in 2^{O(S)}$ or else it will cycle. From this fact and the time–space tradeoff, we obtain the first tight space lower bound of $\Omega(\log^2 n)$ on a probabilistic NNJAG with 2-sided error. No tight space lower bound was previously known even for the more restricted JAG model. However, a coin-flipping probabilistic JAG or NNJAG (as defined in [7, 26]) can run usefully for up to $2^{2^{O(S)}}$ expected time. As mentioned before, it can solve STCON with $O(\log n)$ space and $O(n^n)$ expected time. Thus, one can prove only a time–space lower bound on this coin-flipping model. Since a coin-flipping probabilistic NNJAG with space S and time T can be simulated on our probabilistic NNJAG, using time T and space $S + \log T$, our result is valid on the coin-flipping model for $S \in \Omega(\log^2 n)$ (since $\log T \in O(S)$). For space $S \in O(\log^2 n)$, our result still implies a lower bound of $T \in 2^{\Omega(\log^2 n)}$ on the coin-flipping model. However, for $S \in O(\frac{\log^2 n}{\log\log n})$, Poon [26] gives a stronger lower bound of $T \in 2^{(2^{\Omega(\log^2 n/S)})}$. For example, when $S \in O(\log n)$, his result implies that $T \in 2^{n^c}$ for some constant $c > 0$.

This paper borrows a lot of techniques from [14]. The bound is proved for the probabilistic NNJAG model by transforming the machine into a structured branching program, and applying a progress argument introduced by Borodin et al. [10] and also used in many proofs of time–space trade-off lower bounds, including [8, 5, 9, 33]. Roughly, the argument is that for every short path of the computation, the probability that lots of progress is made, conditional on the fact that this computation path is followed, is less than 2^{-S}. (With space S there are at most 2^S different such subcomputations.) Our proof, however, is complicated by the fact that this is not true for some "lucky" computation paths, and hence a number of new techniques are required to overcome this. In addition, the argument is applied recursively, yielding a substantially greater lower bound than would be possible without recursion. We note that similar recursive techniques have also been used in [13, 7, 33, 15, 26].

1.3. Organization of this paper. We first define the NNJAG model in section 2. In section 3, we give the statement of our main result and its corollaries. In sections 4 and 5, we describe the families of graphs used to defeat the NNJAG. In section 6, we define a notion of progress for an NNJAG on such families of graphs. In section 7, we enhance and stylize the NNJAG model to simplify our proof. Sections 8 through 12 contain the technical proof of the lower bound. Section 8 contains the proof of an inductive statement, Lemma 8.3, from which our main result follows. The proof makes forward references to Lemmas 8.1 and 8.2, which are proved in sections 10 through 12 and section 9, respectively. Section 13 gives the conclusion and some open problems.

2. The NNJAG model. A (deterministic) NNJAG [26] J is a finite state automaton with p distinguishable pebbles, q states, and a transition function Δ. The transition function Δ can depend nonuniformly on the size n of the input graph, and the values of p, q can be functions of n. The input to J is a triple (G, s, t), where G is an n-node graph containing nodes s and t. For every node in G, its out-edges are labeled with consecutive integers starting at 0. The nodes in G are also labeled from 0 to $n-1$. We define the *instantaneous description* (id) of J as the pair (Q, Π), where Q is the current state and Π is a mapping of pebbles to nodes, specifying the current location of each pebble in the graph. When J is in id (Q, Π), the transition function Δ determines the next move for J based on (1) the state Q and (2) the mapping Π. A move is either a *walk* or a *jump*. A walk (P, i, Q') consists of moving pebble P along the edge labeled i that comes out of the node $\Pi(P)$ and then assuming state Q'. (If there is no such edge, the pebble just remains on the same node.) A jump (P, P', Q') consists of moving pebble P to the node $\Pi(P')$ and then assuming state Q'. The NNJAG J is initialized to state Q_0 with all its pebbles on node s. It is said to *accept* an input (G, s, t) if it enters an accepting state on this input. An NNJAG solves STCON for n-node graphs if for every input (G, s, t), where G is an n-node directed graph, it accepts the input if and only if there is a directed path from s to t in G. We define the space used by the NNJAG as $p \log n + \log q$, i.e., as the number of bits needed to specify an id. The time used is the number of moves it has made. For simplicity, we assume that the labels of nodes s and t are always fixed (say, as 0 and $n-1$, respectively). Hence, s and t are not part of the input.

A probabilistic NNJAG J is defined as a distribution on deterministic NNJAGs. On a given input, it first chooses probabilistically a deterministic NNJAG from the distribution and then runs this deterministic NNJAG on the input. The space used is taken as the maximum over all the deterministic NNJAGs in the distribution and the expected (worst case) time is the expected (worst case) running time over the

distribution. We say that J solves STCON with 2-sided error if for every input (G, s, t) the probability of J entering an accepting state is at least $3/4$ when there is a path from node s to t and is at most $1/4$ otherwise.

3. Statement of results. Our main result is the following.

THEOREM 3.1. *If J is a probabilistic NNJAG that solves* STCON *on n-node graphs while taking expected time T and using space S, then $T \in 2^{\Omega(\log^2(n/S))}$ when $S \in O(n^{1-\delta})$, where $\delta > 0$ and $T \in 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$ otherwise.*

The proof of Theorem 3.1 follows by applying Yao's lemma [32] to the following theorem.

THEOREM 3.2. *For any δ, $\epsilon > 0$ there is a distribution $\mathcal{D}$ on n-node graphs such that*

1. *$Pr_{G \in \mathcal{D}}[\, G \in$ STCON $\,] = 1/2$, and*
2. *for any deterministic NNJAG, using space $S \in O(n^{1-\delta})$ and (worst case) time $T \notin 2^{\Omega(\log^2(n/S))}$, or $S \in \omega(n^{1-\delta})$ and $T \notin 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n))} \times (nS/\log n)^{1/2}$,*

$$\Pr_{G \in \mathcal{D}}[J \text{ is correct on input } G] < \frac{1}{2} + 2\epsilon.$$

Proof of Theorem 3.1. Theorem 3 of [32] states that for any randomized algorithm $\mathcal{J}$ that has probability of error at most λ and any input distribution $\mathcal{D}$, the expected time of $\mathcal{J}$ on the worst case input is at least half the average time of the best deterministic algorithm that errs with probability at most 2λ on random input chosen from $\mathcal{D}$. By Theorem 3.2, the latter quantity is at least $T \times (1 - 2\lambda - \frac{1}{2} - 2\epsilon)$, where $T \in 2^{\Omega(\log^2(n/S))}$ for $S \in O(n^{1-\delta})$ and $T \in 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$ otherwise. Putting λ as some constant less than $\frac{1}{4} - \epsilon$ and since $S \in O(n^{1-\delta})$ for some $\delta > 0$, we get the required lower bound on a probabilistic NNJAG that errs with probability at most λ. □

Theorem 3.2 is strong enough to yield an optimal space lower bound for the deterministic NNJAG model, as an immediate corollary.

COROLLARY 1. *Any probabilistic NNJAG that solves* STCON *requires $\Omega(\log^2 n)$ space.*

Proof. Once the deterministic NNJAG to be used is chosen from the distribution, the probabilistic NNJAG becomes deterministic. Hence, while using space at most S, the NNJAG cannot take more than $2^{O(S)}$ steps without going into an infinite loop. If an NNJAG J uses space $S \notin \Omega(\log^2 n)$, then for sufficiently large n the number of steps it can take is smaller than the lower bound implied by Theorem 3.2 and the result follows. □

4. Layered graphs. From now on, we let δ be a fixed positive constant. A (d, x, f)-*layered graph*, first defined in [4], is a graph consisting of d layers, each containing x nodes. The jth node in layer i is denoted by (and named) $u_{\langle i,j \rangle}$. (Hence, the NNJAG always knows the location of a pebble in terms of i, j.) Every node has at most f outgoing edges to some (not necessarily distinct) nodes in the next layer. Here, we will set $f = \Theta((n \log n/S)^{1/2})$ for $S \in O(n^{1-\delta})$ and $f = 2$ otherwise.

Let $D = \lceil 80 \log n/\log f \rceil$ (so that $f^D \geq n^{80}$). Note that D is constant with respect to n if $S \in O(n^{1-\delta})$ and $D \in \Theta(\log n)$ otherwise. The distribution $\mathcal{B}(x)$ is a distribution on (D, x, f)-layered graphs. Each graph $G \in \mathcal{B}(x)$ will have $x/2$ *hard paths* (to be defined shortly) of length D and is obtained as follows. In each layer i, except the top layer, we pick (without replacement) a sequence of $x/2$ nodes,

uniformly at random. Let us denote the jth node picked as $v_{\langle i,j\rangle}$. (It is the node $u_{\langle i,j'\rangle}$ for some j'.) These nodes are called the *hard nodes*. The remaining $x/2$ nodes in that layer are called the *easy nodes*. For layer 1, we choose the sequence of nodes $u_{\langle 1,1\rangle}, u_{\langle 1,2\rangle}, \ldots, u_{\langle 1,x/2\rangle}$ as the sequence of hard nodes. We shall put in edges so that if an NNJAG walks a pebble $D-1$ steps starting from a hard node in the top layer, then it is difficult for the pebble to be on a hard node when it reaches layer D.

First, the hard nodes are connected by the edges $(v_{\langle i,j\rangle}, v_{\langle i+1,j\rangle})$ for each $i \in [1 \ldots D-1]$ and each $j \in [1..x/2]$. The path from $v_{\langle 1,j\rangle}$ to $v_{\langle D,j\rangle}$ is called the jth *hard path*. The nodes $v_{\langle 1,j\rangle}$ and $v_{\langle D,j\rangle}$ are called the *root* and *goal* of the jth hard path, respectively. Thus, there are $x/2$ hard paths, roots, and goals in G. The edge labels are chosen independently and uniformly from $[0 \ldots f-1]$. Thus, for each root r the vector of edge labels on the hard path rooted at r, denoted $\vec{\ell}_r$, is chosen uniformly at random from $[0..f-1]^{D-1}$.

For each layer $i \in [1..D-1]$, each hard node $v_{\langle i,j\rangle}$ will have further $f-1$ outgoing edges, and each easy node will have f outgoing edges. The destinations of these edges are chosen independently (with replacement) at random from the set of easy nodes in layer $i+1$. In this way, the in-degree of each hard node is kept to 1.

5. Recursively layered graphs. Set $\chi = \Theta((\frac{n^3 S}{\log n})^{1/4})$ for $S \in O(n^{1-\delta})$ and $\chi = \Theta((\frac{nS}{\log n})^{1/2})$ otherwise. Set $K = \lfloor \frac{\log(n/(4\chi))}{\log 2D} \rfloor$. Thus, $K \in \Theta(\log(\frac{n \log n}{S}))$ for $S \in O(n^{1-\delta})$ and $K \in \Theta(\log(\frac{n \log n}{S})/\log\log n)$ otherwise. Moreover, $K \leq \log n$ since $S \geq \log n$. We first construct, recursively, $K+1$ distributions $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_K$ on layered graphs, where $\mathcal{H}_k$ is a distribution on $(D^k, 2^k\chi, f)$-layered graphs. Each such graph has χ *super goals*. In addition, for $k > 0$, each graph in $\mathcal{H}_k$ has $D^{k-1}2^{k-1}\chi$ hard paths of length D, each one with a goal. Our input distribution $\mathcal{D}$ of n-node graphs in Theorem 3.2 is formed by adding a few nodes and edges to each graph in $\mathcal{H}_K$.

The distribution $\mathcal{H}_0$ contains only one graph, which is simply a layer of χ isolated nodes. These nodes are the super goals. For $k > 0$, a graph G in $\mathcal{H}_k$ is formed as follows. We choose a graph G' from $\mathcal{H}_{k-1}$ and replace each layer i of G' with a graph G_i chosen from $\mathcal{B}(2^k\chi)$. Note that each G_i has $2^k\chi/2 = 2^{k-1}\chi$ hard paths and that each layer of G' has the same number of nodes. We identify the jth hard path of G_i (i.e., the path from $v_{\langle 1,j\rangle}$ to $v_{\langle D,j\rangle}$ of G_i) with the jth node in layer i (i.e., $u_{\langle i,j\rangle}$) of G'. Every edge that goes into $u_{\langle i,j\rangle}$ of G' will now go into $v_{\langle 1,j\rangle}$ of G_i, and every edge that goes out of $u_{\langle i,j\rangle}$ of G' will go out of $v_{\langle D,j\rangle}$ of G_i. The easy nodes in G_i are not connected to any node outside G_i.

Since G is uniquely determined by G' and $G_1, \ldots, G_{D^{k-1}}$ (and viceversa), we often denote G as a tuple $\langle G_1, G_2, \ldots, G_{D^{k-1}}; G'\rangle$. The graph G' is called the *collapsed* graph of G, denoted $C(G)$. The set of hard paths (respectively, roots and goals) of G is the union of all the sets of hard paths (respectively, roots and goals) in $G_1, G_2, \ldots, G_{D^{k-1}}$. Hence, G has $D^{k-1}2^{k-1}\chi$ hard paths (and the same number of roots and goals) in total. The χ super goal nodes in G are the goal nodes of the χ hard paths in $G_{D^{k-1}}$, representing the χ super goal nodes in $G' \in \mathcal{H}_{k-1}$. Note that the super goals are on the bottom level of G and are associated with the χ nodes in the graph from $\mathcal{H}_0$. The edges within each of the G_is are called the *base edges* of G. The other edges, i.e., those connecting the G_is, are in one-to-one correspondence with the edges in the collapsed graph $C(G)$ of G, and hence they are called the *collapsed edges*. Note that graphs in $\mathcal{H}_1$ have base edges but not collapsed edges, and the graph in $\mathcal{H}_0$ does not have any edge at all.

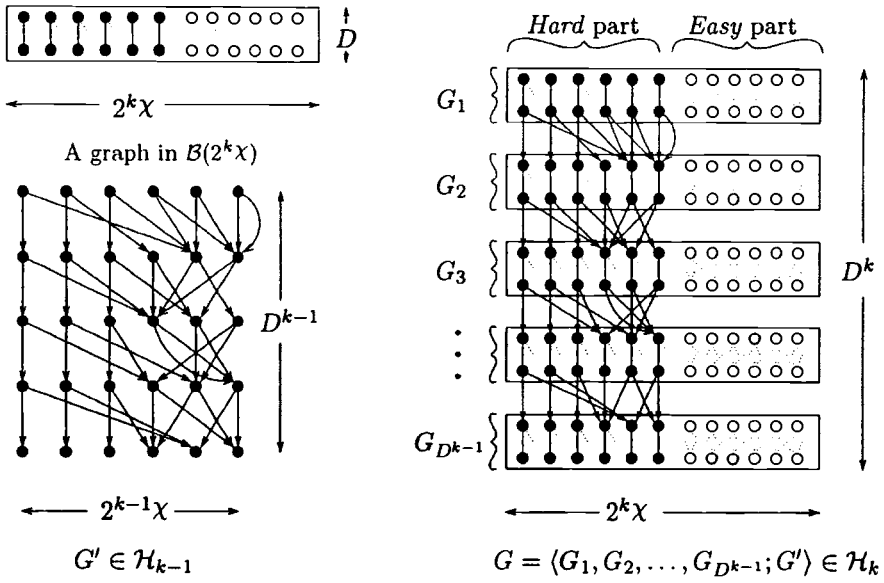


FIG. 1. *An example:* $f = 2$.

Figure 1 shows a graph $G \in \mathcal{H}_k$ on the right and its collapsed graph $C(G) \in \mathcal{H}_{k-1}$ and the symbol for a base graph in $\mathcal{B}(2^k\chi)$ on the left. We rearranged the nodes so that all the hard nodes in the G_i's appear on the left half.

For each $k \in [0..K]$, we obtain a distribution $\mathcal{G}_k$ by adding to each graph in $\mathcal{H}_k$ the following *auxiliary* nodes and edges (see Figure 2):

(A1) a directed path $(s = w_1, w_2, \ldots, w_{2^k\chi})$ with $w_1 = s$ and, for each $j \in [1..2^k\chi]$, an edge from w_j to $u_{\langle 1,j \rangle}$ of G;
(A2) the isolated node t;
(A3) a special isolated node, referred to as the *lost* node;
(A4) a number of isolated nodes so that the total number of nodes in the graph is exactly n.

The lost node is introduced for technical reasons that will become clear in section 7. These auxiliary nodes and edges are fixed for each graph $G \in \mathcal{G}_k$. Hence, for $k > 0$, G can still be specified by a tuple $\langle G_1, G_2, \ldots, G_{D^{k-1}}; G' \rangle$, where $G_1, \ldots, G_{D^{k-1}}$ are in $\mathcal{B}(2^k\chi)$ and G' is in $\mathcal{H}_k$. The *collapsed* graph of G, denoted by $C(G)$, is the graph G' augmented with the auxiliary nodes and edges needed to form a graph in $\mathcal{G}_{k-1}$ from a graph in $\mathcal{H}_{k-1}$. Thus, $C(G)$ is in $\mathcal{G}_{k-1}$. Note that, excluding the nodes added in (A4), each graph in $\mathcal{G}_k$ consists of a $(D^k, 2^k\chi, f)$-layered graph, a path with $2^k\chi$ nodes, the node t and the lost node. These add up to a total of $(2D)^k\chi + 2^k\chi + 2 \leq 4(2D)^k\chi \leq n$ nodes for $k \leq K$ by our choice of χ and K. Hence, we are not adding a negative number of nodes in (A4). Finally, the distribution $\mathcal{D}$ of Theorem 3.2 is defined as follows. First choose a graph $G' \in \mathcal{G}_K$ and then uniformly at random choose one of the χ super goals in G' as the *special node*. With probability $1/2$ connect the special node to the isolated node t to form a graph G. Clearly, $\Pr_{G \in \mathcal{D}}[G \in \text{STCON}] = 1/2$.

6. Defining progress. Consider the computation of an NNJAG J on input G. We will analyze the progress of J during different phases of the computation. In the following definition, a *subcomputation* A refers to a sequence of moves taken by the

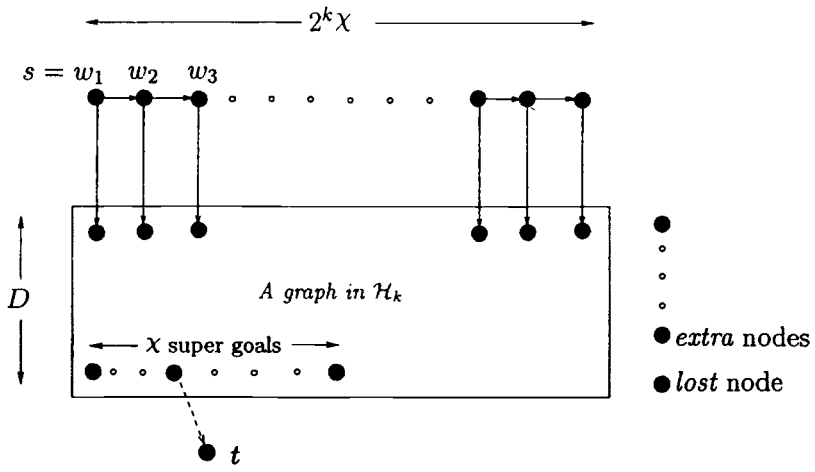


FIG. 2. *A graph in* $\mathcal{G}_k$.

NNJAG, starting from certain id (Q, Π). Once we recast an NNJAG as a branching program in section 7, one can think of A as a subbranching program.

DEFINITION 1. *For any subcomputation A and any input $G \in \mathcal{G}_k$, $w_A(G)$ is the number of different goals in G that were pebbled (i.e., reached by a pebble) at any time during A. Similarly, $w^*_A(G)$ is the number of super goals in G that were pebbled during A.*

Note that when A begins, some pebbles may already be sitting on a goal node. These goals will be counted as progress in $w_A(G)$. However, there can be at most $S/\log n$ such progress. The following lemma shows why reaching the other goals is difficult for an NNJAG.

LEMMA 6.1. *If at some step T' a particular hard path does not contain any pebble and at some later step T'' a pebble arrives at the goal of this path, then each edge in that path must be traversed by some pebble between step T' and T''.*

Proof. Observe that every node on a hard path has in-degree 1 and that in the NNJAG model a pebble can arrive at a node only if the node is already occupied by some pebble or if it walks to the node. □

We point out that it is not necessary for a general computation model to find out the hard path before it can inspect the edge connections of the associated goal node. This is the only significant difference between a general model and an NNJAG model that we will employ in our proof.

Recall that an input $G = \langle G_1, \ldots, G_{D^{k-1}}; G' \rangle \in \mathcal{G}_k$ consists of the collapsed graph $G' \in \mathcal{G}_{k-1}$ and the base graphs $G_1, \ldots, G_{D^{k-1}} \in \mathcal{B}(2^k\chi)$. The NNJAG has to learn both the structure of the base graphs and that of the collapsed graph. Obviously, $w_A(G)$ measures how much A has learned about the base graphs. The following lemma shows that $w_A(G)$ is also a good estimate of the number of different collapsed edges traversed during a subcomputation A.

LEMMA 6.2. *The number of different collapsed edges of an input graph $G \in \mathcal{G}_k$ that can be traversed during a subcomputation A of an NNJAG is at most $f \times w_A(G)$.*

Proof. An NNJAG can traverse an edge (u, v) only if there is a pebble on node u before the traversal. If the edge is a collapsed edge, u must be a goal. Since every goal has out-degree at most f, pebbling one such node allows the NNJAG to traverse

at most f different collapsed edges. □

Lemma 8.3, to follow, uses Lemmas 6.1 and 6.2 recursively to prove that it is hard for an NNJAG to reach the χ super goal nodes. Roughly speaking, the argument goes as follows. Suppose we have proved that it is hard to visit the super goals of graphs chosen from $\mathcal{G}_{k-1}$ within time T_{k-1}. Consider a graph $G = \langle G_1, \ldots, G_{D^{k-1}}; G' \rangle$ in $\mathcal{G}_k$ and an NNJAG J with time T_k. We will prove, using Lemma 6.1, that for any input $G' \in \mathcal{G}_{k-1}$, it is hard for J to visit many goals in the graphs $G_1, \ldots, G_{D^{k-1}} \in \mathcal{B}(2^k\chi)$ within time T_k. In particular, Lemma 6.2 implies that no more than T_{k-1} different edges in G' are traversed. To conclude the argument, we show that J is effectively an NNJAG trying to reach, within time T_{k-1}, the super goals for graphs chosen from $\mathcal{G}_{k-1}$, which is difficult by the inductive assumption.

It should be pointed out that J can traverse the same edge many times (which is natural, since J cannot remember the result of too many edge traversals with limited space). Therefore, we cannot directly claim that J runs in T_{k-1} time on inputs from $\mathcal{G}_{k-1}$. For this reason, we measure the time of an NNJAG using the *s-height*, $h_A()$, of the corresponding branching program A. Precise definitions of *s*-height will be given in section 7. Here, we just state that an NNJAG running in time T will have $h_A(G) \leq T$ for any G. Thus Lemma 8.3 will imply that if J is an NNJAG that uses space $S \in O(n^{1-\delta})$ and time $T \notin 2^{\Omega(\log^2(n/S))}$ or space $S \in \omega(n^{1-\delta})$ and time $T \notin 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$, then for any $\epsilon > 0$, $\Pr_{G \in \mathcal{D}}[\, w^*_J(G) > \epsilon\chi \,] < \epsilon$. Below we show how Theorem 3.2 follows from this last statement.

Proof of Theorem 3.2. Choose $G \in \mathcal{D}$. Recall that this can be done by choosing $G_a \in \mathcal{G}_K$ and then choosing one of its χ super goals to be special, uniformly at random. Let G_b be the same as G_a except with an edge from the special node to t. Then G is uniformly chosen to be G_a or G_b. If G_a is such that $w^*_J(G_a) > \epsilon\chi$, i.e., J reaches a lot of super goals, then assume that J gives the correct answer on G. From Lemma 8.3, the probability of this event is less than ϵ. If J pebbles at most $\epsilon\chi$ super goals, then the probability that J pebbles the special node is at most ϵ because the NNJAG cannot tell that a super goal is special unless it pebbles the node. Finally, if J does not pebble the special node it cannot learn whether there is an edge from the special node to t. Therefore, in this case, the computations on G_a and G_b are the same and hence the probability of giving the correct answer for G is $1/2$. Thus, the probability of giving the right answer for G is less than $1/2 + 2\epsilon$. □

7. An NNJAG as a branching program. We will introduce a variant of the NNJAG model which we call the *pebble location redundant* NNJAG model. The reason is that while the new model maintains all the power of an NNJAG it helps us prove a collapsing lemma. In particular, we shall show that it is helpful to construct a pebble location redundant NNJAG J' for graphs in $\mathcal{G}_{k-1}$ from a pebble location redundant NNJAG J for graphs in $\mathcal{G}_k$. We call this the "collapsing" of J to J'.

An NNJAG is said to be pebble location redundant if the current state always determines the current location of all the pebbles and, hence, the state alone is sufficient to specify the id of the NNJAG. More formally, this means that there is a function $\widehat{\Pi}$ such that if the NNJAG is in state Q, then $\widehat{\Pi}(Q)$ specifies the locations of all the pebbles. As a first step in getting a pebble location redundant NNJAG, we enhance a standard NNJAG as follows. First, we allow it to jump a pebble to the lost node (which is isolated), and for any $j \in [1..2^k\chi]$ to the nodes w_j and $u_{\langle 1,j \rangle}$. We call such a jump a *node-jump*. Note that in the standard NNJAG model a pebble can jump only to (the node occupied already by) another pebble. Also, we modify a step to be

taken from an id (Q, Π) by the NNJAG to consist of the following substeps.
Substep 1. Based on (Q, Π), either it walks a pebble P along the edge with a specified label ℓ, or it node-jumps a pebble P. It can also choose not to move any pebble. Let Π_1 specify the new pebble locations.
Substep 2. Based on (Q, Π) and Π_1, it performs a (possibly empty) sequence of pebble-to-pebble jumps and then assumes some state Q'.
The intuition supporting these modifications is that a sequence of moves of a standard NNJAG can be viewed as a sequence of "macro steps," each of which starts with a walk, followed by a (possibly empty) sequence of jumps. Each such jump causes the standard NNJAG to enter a unique next id. Intuitively, the NNJAG "learns" about the input only by taking walking steps. Each macro step can be performed in one step in the enhanced model. It follows that a time lower bound on this new model implies the same lower bound on the number of walking steps on the original model. For any NNJAG J (modified as above) with p pebbles, q states, and T time, we can construct a pebble location redundant NNJAG J' so that for any possible id (Q, Π) of J, J' will have a state $\langle Q, \Pi \rangle$. In this state, J' will perform the same action as J does on id (Q, Π).[1] Thus, the pebble location redundant NNJAG J' will have p pebbles and $q \times n^p$ states; hence using space $\log(q \times n^p) + p \log n = \log q + 2p \log n$, which is at most twice the space of J. Moreover, it uses no more time than J.

To be able to discuss subcomputations of the NNJAG better, it is convenient to recast the NNJAG as an r-way branching program [8] (defined below). Although an r-way branching program is a general model of computation, the branching program we will examine has "structure" imposed by Lemmas 6.1 and 6.2 regarding NNJAG computations.

A branching program is a directed acyclic graph with a designated source node and a number of sink nodes. Each sink node in the graph is labeled with either *accept* or *reject* and each nonsink node is labeled with an input variable. Furthermore, for each possible value of the input variable that labels a nonsink node, there is a unique out-edge from this nonsink node, labeled with that value. Hence, the out-degree of the graph is at most r, where r is the maximum number of different values possible for an input variable. A *subbranching program* is simply a subgraph rooted at some node.

The nodes in this graph represent the possible states of the machine's memory. In particular, the source node represents the initial memory state. In each step the machine queries an input variable, depending on the current state of its memory, and then changes its memory to another state based on the value returned. Which variable to query and which state to go to, on each possible outcome, are specified by the graph. It is easy to see that for every input, there will be a unique path in the graph from the source node to a sink node. We call such a path the *computation path* followed by the input. We say that a branching program accepts an input if and only if the computation path followed by the input leads to a sink node labeled with accept.

Consider an arbitrary (pebble location redundant) NNJAG J that uses space S, takes time T, and takes inputs from a distribution of n-node graphs with out-degree

[1] Note that in general, a standard NNJAG cannot be made pebble location redundant because, if the move taken from an id (Q_1, Π_1) is a walk, the new pebble location, Π_2, will depend on the input graph. Hence, the NNJAG cannot know which new state Q_2 to move to so that $\widehat{\Pi}(Q_2) = \Pi_2$. In contrast, in the modified NNJAG the pebble location, Π_2, after Substep 2, is uniquely determined by (Q_1, Π_1) and Π'_1. Hence, it is possible for the NNJAG to choose a state Q_2 so that $\widehat{\Pi}(Q_2) = \Pi_2$.

f. The corresponding branching program A has a row of configuration nodes for each of the time steps $t \in [1 \dots T]$. Each row has 2^S configuration nodes (Q, Π, t), one for each NNJAG id (Q, Π). For every id (Q, Π) of J and time step t, there will be a configuration vertex (Q, Π, t) in A. The configuration vertex $(Q_0, \Pi_0, 1)$, where (Q_0, Π_0) is the *start* id of J, is taken as the *start* vertex of A. For each *accept* id (Q_a, Π_a) of J, $(Q_a, \Pi_a, 1), (Q_a, \Pi_a, 1), \dots, (Q_a, \Pi_a, T)$ are *accept* configuration vertices in A, and likewise for the *reject* ids. The input variables labeling the configuration vertices of A are the variables $X_{\langle u,\ell\rangle}$, where $u \in [0..n-1]$ is a node name and $\ell \in [0..f-1]$ is an edge label. The variable $X_{\langle u,\ell\rangle}$ will have value v if there is an edge (u, v) labeled with ℓ in the input graph and the value "undefined" if there is no such edge. Thus, if in id (Q, Π) the first substep of J walks a pebble from node u along the edge with label ℓ, the configuration vertex (Q, Π, t) in A will be labeled with the variable $X_{\langle u,\ell\rangle}$. Furthermore, the vertex will have a directed edge labeled with v to configuration vertex $(Q', \Pi', t+1)$ if for some input graph, $X_{\langle u,\ell\rangle} = v$ (i.e., the queried edge has destination v) and the subsequent jumps taken in Substep 2 by J bring the machine to the id (Q', Π'). If J does not walk any pebble in Substep 1, the configuration vertex will not get any label and will have only one unlabeled out-edge pointing to some configuration vertex $(Q', \Pi', t+1)$, depending on Substep 2 of J.

Note that the branching program A so constructed is *leveled* in the sense that each configuration vertex can be assigned a level number so that edges from level i only go to level $i+1$. Moreover, all the rows in A are identical, because the transition function of the (deterministic) NNJAG does not depend on time. Therefore, the number of distinct subbranching programs of a fixed height is at most 2^S.

Finally, we introduce a variant of branching programs called *sectioned* branching programs. A branching program is said to be sectioned if its vertices are partitioned into sections so that the out-edges of a vertex in section i can only go to vertices in section i or $i+1$. Thus, each computation path will go through each section at most once.

DEFINITION 2. *A branching program A is* properly sectioned *for an input G if it queries at most $\frac{3fS}{\log n}$ different edges of G in each section. If A is properly sectioned for G, then its* s-height *on G, denoted $h_A(G)$, is $\frac{3fS}{\log n}$ times the number of sections A contains; otherwise, $h_A(G)$ is infinite.*

Note that a set of queries to the same edge of the input graph within a section is charged as only one query in the s-height measure. The branching program defined earlier can be viewed as a sectioned branching program with $T/\frac{3fS}{\log n}$ sections, each of which queries at most $\frac{3fS}{\log n}$ different input edges. Moreover, on every input G, A will have s-height $T = (T/\frac{3fS}{\log n}) \times \frac{3fS}{\log n}$.

8. Proof outline. In the rest of this paper, a directed edge from u to v with label ℓ will be denoted by the triple $\langle u, \ell, v\rangle$. Also, by $\mathcal{G}(\mathcal{O})$ we denote the distribution obtained by selecting those graphs in $\mathcal{G}$ that satisfy a condition $\mathcal{O}$. We will derive Lemma 8.3 by induction. Before doing so, we present two lemmas that are central to the proof of that inductive statement. The first mainly concerns traversing base edges of graphs in $\mathcal{G}_k$. It bounds the probability of a machine making a lot of progress within a short period of time.

LEMMA 8.1 (main lemma). *Let A be any sectioned subbranching program derived from some pebble location redundant NNJAG with at most $S/\log n$ pebbles. Then for any $k \in [1..K]$,*

$$\Pr_{G \in \mathcal{G}_k}[w_A(G) \geq 3S/\log n \text{ and } h_A(G) \leq \chi/8] < 2^{-2S}.$$

The intuition behind Lemma 8.1 is as follows. Recall that $w_A(G)$ is the number of goals that get pebbled. We "give away" one such node for each of the (at most) $S/\log n$ pebbles. When $h_A(G) \leq \chi/8$, A queries at most $\chi/8$ different edges in G. Consider the probability of pebbling the goal corresponding to an arbitrary root r, assuming that the hard path rooted at r does not contain any pebble initially. There are f^{D-1} possibilities for the vector, $\vec{\ell}_r$, of edge labels on this hard path. To remind us of its dependency on G, let us use the symbol $\vec{\ell}_r(G)$ instead of $\vec{\ell}_r$ in the following. An NNJAG can move a pebble down from r following some vector $\vec{\ell} \in [0..f-1]^{D-1}$ of edge labels, hoping that $\vec{\ell} = \vec{\ell}_r(G)$. For G drawn from $\mathcal{G}_k$, this probability is $f^{-(D-1)}$. The NNJAG can dynamically choose $\vec{\ell}$ based on the names of the nodes on the path it has traced so far. However, this will not be a lot of help, since the name of the nodes on the hard path are chosen randomly. Recall that $f^D \geq n^{80}$. Since $\chi \in O(n)$, it follows that $f^{D-1} \gg \chi/8$. Clearly, by querying at most $\chi/8$ different edges in the input graph, on the one hand, the NNJAG cannot try many different $\vec{\ell}$s. Hence, the probability of having at least one of them being successful is small.

On the other hand, the NNJAG can eliminate some of the possibilities it needs to consider by detecting "collisions of edges" and hence increase the probability that it succeeds. For example, when it learns that two different edges have the same destination node v, it learns that this node v is not on the hard path since its in-degree is bigger than 1. Hence, any path continuing from node v need not be traversed. However, within $\chi/8$ steps, the probability that an edge traversed by the NNJAG collides with some other traversed edge can be shown to be at most 1/4. (Intuitively, the probability is $\frac{\chi/8}{2^k\chi/2} \leq 1/8$. For the simplicity of the proof, we argue in section 11 that this probability is at most 1/4.) By analyzing a variant of branching processes, we can show that the probability of eliminating a large number of vectors $\vec{\ell} \in [0..f-1]^{D-1}$ in this way, is small. In other words, with high probability, the NNJAG still has a lot of possible $\vec{\ell}$s to try out. This discussion considers only a single root. When there are many roots, we need to take care of the dependencies among them before we can apply some Chernoff-type bounds. The detailed analysis and proof of Lemma 8.1 comprise sections 10, 11, and 12.

The second lemma concerns the traversal of collapsed edges of graphs in $\mathcal{G}_k$. Let E be a fixed set of D^{k-1} base graphs $G_1, \ldots, G_{D^{k-1}} \in \mathcal{B}(2^k\chi)$ (we call such a set of graphs a *complete* set) and $\mathcal{G}_k(E)$ be the distribution of $\mathcal{G}_k$ conditioned on these fixed graphs. The lemma relates the computation of a pebble location redundant NNJAG J on inputs in $\mathcal{G}_k(E)$ to that of a faster (in terms of s-height) pebble location redundant NNJAG J' on inputs in $\mathcal{G}_{k-1}$. For any complete set E of base graphs, define a function C_E from nodes in $G \in \mathcal{G}_k(E)$ to nodes in $C(G) \in \mathcal{G}_{k-1}$ as follows:

$$C_E(v) = \begin{cases} w_i & \text{if } v = w_i \text{ for some } i \in [1..2^{k-1}\chi], \\ u_{\langle i,j \rangle} & \text{if } v \text{ is on the } j\text{th hard path in } G_i, \\ \text{lost} & \text{otherwise.} \end{cases}$$

Note that the function is well defined because for an input $G \in \mathcal{G}_k(E)$, whether a node v is a hard node, an easy node, or an auxiliary node is fixed. For any pebble mapping Π for graphs in $\mathcal{G}_k(E)$, denote $C_E(\Pi)$ as the pebble mapping Π' for graphs in $\mathcal{G}_{k-1}$ such that for any pebble P, $\Pi'(P) = C_E(\Pi(P))$.

LEMMA 8.2 (collapsing lemma). *Let k be any integer in $[1..K]$, J be any pebble location redundant NNJAG with p pebbles and q states, and E be any complete set of base graphs. There exists a corresponding pebble location redundant NNJAG J' with the same number of pebbles and states such that, for any $G \in \mathcal{G}_k(E)$, J is in id (Q, Π)*

in some step on input G if and only if J' is in id $(Q, C_E(\Pi))$ in the same step on input $C(G) \in \mathcal{G}_{k-1}$.

Note that J and J' use the same space. Moreover, J traverses a collapsed edge $\langle u, \ell, v\rangle$ in G if and only if J' traverses the corresponding edge $\langle C_E(u), \ell, C_E(v)\rangle$ in $C(G)$, and J accepts G if and only if J' accepts $C(G)$. The proof of Lemma 8.2 is given in section 9. Having stated Lemmas 8.1 and 8.2, we are ready to state and prove the following inductive statement.

LEMMA 8.3. *For any $\epsilon > 0$ and any $k \in [0..K]$, if $T_k = \epsilon\chi\big(\frac{\chi \log n}{24fS}\big)^k$ and A is a sectioned branching program with no more than $T_k/(\frac{3fS}{\log n})$ sections, derived from a pebble location redundant NNJAG J which uses at most space S, then*

$$\Pr_{G\in\mathcal{G}_k}[w^*_A(G) > \epsilon\chi \text{ and } h_A(G) \le T_k] \le k2^{-S} < \epsilon.$$

Proof of Lemma 8.3.

Base case. When $k = 0$, the branching program A can query at most $T_0 = \epsilon\chi$ different edges. Hence, it cannot discover more than $\epsilon\chi$ super goals.

Inductive step. Assume that the lemma is true for $k-1$. Consider a sectioned branching program A having at most $T_k/\frac{3fS}{\log n}$ sections derived from some pebble location redundant NNJAG J with at most S space. Suppose, for the sake of contradiction, that $\Pr_{G\in\mathcal{G}_k}[\, w^*_A(G) > \epsilon\chi$ and $h_A(G) \le T_k\,] > k2^{-S}$. We will show that in this case there exists some sectioned branching program A' with at most $T_{k-1}/\frac{3fS}{\log n}$ sections corresponding to some pebble location redundant NNJAG J' using at most S space such that $\Pr_{G'\in\mathcal{G}_{k-1}}[\, w^*_{A'}(G') > \epsilon\chi$ and $h_{A'}(G') \le T_{k-1}\,] > (k-1)2^{-S}$. This contradicts the inductive hypothesis.

We break A into at most $T_{k-1}/\frac{3fS}{\log n}$ slices so that slice i consists of section $i(T_k/T_{k-1})$ to section $(i+1)(T_k/T_{k-1}) - 1$, inclusive. (Thus each slice contains $T_k/T_{k-1} = \frac{\chi \log n}{24fS}$ sections.) Let $\mathcal{F}$ be the set of $G \in \mathcal{G}_k$ such that $h_A(G) \le T_k$ and at least one subbranching program $\widehat{A}$, which lies completely within a slice, has $w_{\widehat{A}}(G) \ge 3S/\log n$. Since $h_A(G)$ being finite implies that $\widehat{A}$ is properly sectioned for G, $h_{\widehat{A}}(G) \le (T_k/T_{k-1})(\frac{3fS}{\log n}) = \chi/8$.

Consider the maximal subbranching program $\widehat{A}$ which lies completely within slice i and is rooted at the node through which G first enters slice i. There are at most 2^S such subbranching programs in A. Combining this fact with Lemma 8.1, we have $\Pr_{G\in\mathcal{G}_k}[\, G \in \mathcal{F}\,] < 2^{-S}$. Therefore, $\Pr_{G\in\mathcal{G}_k}[\, w^*_A(G) > \epsilon\chi$ and $h_A(G) \le T_k$ and $G \notin \mathcal{F}\,] > (k-1)2^{-S}$. Let us choose a complete set E of base graphs so that $\Pr_{G\in\mathcal{G}_k(E)}[\, w^*_A(G) > \epsilon\chi$ and $h_A(G) \le T_k$ and $G \notin \mathcal{F}\,] > (k-1)2^{-S}$. By Lemma 8.2, we can construct from the pebble location redundant NNJAG J, another pebble location redundant NNJAG J' that runs on $\mathcal{G}_{k-1}$ with the same number of pebbles and states as J. From J', we can construct a sectioned branching program A' with at most $T_{k-1}/\frac{3fS}{\log n}$ sections, one section for each of the slices of A. This is done by putting a configuration vertex of A' in section i if and only if the corresponding[2] configuration vertex of A is in slice i. In A, edges go only from vertices in slice i to vertices in slice i or $i+1$. Therefore, in A', edges go only from vertices in section i to vertices in section i or $i+1$. Hence, this is a legal way of partitioning the vertices of A' into sections. Now, consider an arbitrary graph $G \in \mathcal{G}_k(E) - \mathcal{F}$. At most $\frac{3S}{\log n}$ progress

[2]There is a one-to-one correspondence between states of J and J'. It is not hard to see that there is also a one-to-one correspondence between configuration vertices of A and A'.

is made in the unique maximal subbranching program that G passes through in each slice of A. By Lemma 6.2, each such subbranching program can query at most $\frac{3fS}{\log n}$ different collapsed edges in G. Hence, each corresponding subbranching program in A' queries at most $\frac{3fS}{\log n}$ different edges in $C(G)$. Therefore, A' is properly sectioned for $C(G)$ for all $G \in \mathcal{G}_k(E) - \mathcal{F}$. Since A has $T_{k-1}/\frac{3fS}{\log n}$ slices, A' has the same number of sections. It follows that $h_{A'}(C(G)) \leq T_{k-1}$ for all $G \in \mathcal{G}_k(E) - \mathcal{F}$. Since $C(G)$ is chosen independent of the G_is, the distribution $\mathcal{G}_k(E)$ is isomorphic to the distribution $\mathcal{G}_{k-1}$. Therefore,

$$\begin{aligned}
&\Pr_{C(G)\in\mathcal{G}_{k-1}}[\, w^*_{A'}(C(G)) > \epsilon\chi \text{ and } h_{A'}(C(G)) \leq T_{k-1}\,] \\
&\geq \Pr_{G\in\mathcal{G}_k(E)}[\, w^*_{A'}(C(G)) > \epsilon\chi \text{ and } G \notin \mathcal{F}\,] \\
&= \Pr_{G\in\mathcal{G}_k(E)}[\, w^*_{A}(G) > \epsilon\chi \text{ and } G \notin \mathcal{F}\,] \\
&\geq \Pr_{G\in\mathcal{G}_k(E)}[\, w^*_{A}(G) > \epsilon\chi \text{ and } h_A(G) \leq T_k \text{ and } G \notin \mathcal{F}\,] \\
&> (k-1)2^{-S}. \qquad \square
\end{aligned}$$

For $k = K$ the above inductive statement implies that any deterministic pebble location redundant NNJAG which uses at most S space and takes $O(T_K)$ time, will pebble more than $\epsilon\chi$ super goals with probability less than $K2^{-S}$. Recall that $T_K = \epsilon\chi\big(\frac{\chi \log n}{24fS}\big)^K$. For $S \in O(n^{1-\delta})$, we set $f = \Theta((n \log n/S)^{1/2})$, $\chi = \Theta((n^3 S/\log n)^{1/4})$, and $K = \Theta(\log(\frac{n \log n}{S}))$. Hence $T_K = 2^{\Omega(\log^2(\frac{n \log n}{S}))} \times (n^3 S/\log n)^{1/4} = 2^{\Omega(\log^2(n/S))}$. For $S \in \omega(n^{1-\delta})$, we set $f = 2$, $\chi = \Theta((nS/\log n)^{1/2})$, and $K = \Theta(\log(\frac{n \log n}{S})/\log\log n)$. Thus, we get $T_K = 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$. For big enough n, $K2^{-S} < \epsilon$, since $K \leq \log n$ and $S \geq \log n$. Thus, if J is an NNJAG that uses space S and time $T \notin \Omega(T_K)$, then for any $\epsilon > 0$, $\Pr_{G\in\mathcal{D}}[\, w^*_J(G) > \epsilon\chi\,] < \epsilon$.

Note that for $S \in O(n^{1-\delta})$, the input graph has out-degree $f = \Theta((\frac{n \log n}{S})^{1/2})$ which is nonconstant. We can convert the graphs of out-degree f into graphs of out-degree 2 by replacing each node with a binary tree of size $O(f)$. This blows up the number of nodes by a factor of f. Hence, our lower bound becomes $T \in 2^{\Omega(\log^2(n/fS))} = 2^{\Omega(\log^2(n/S))}$, where n is the number of nodes in the out-degree 2 graph.

9. Collapsing an NNJAG.

LEMMA 8.2 (repeated). *Let k be any integer in $[1..K]$, J be any pebble location redundant NNJAG with p pebbles and q states, and E be any complete set of base graphs. There exists a corresponding pebble location redundant NNJAG J' with the same number of pebbles and states such that, for any $G \in \mathcal{G}_k(E)$, J is in id (Q, Π) in some step on input G if and only if J' is in id $(Q, C_E(\Pi))$ in the same step on input $C(G) \in \mathcal{G}_{k-1}$.*

Proof. J' will have the same set of states as J. Let $\widehat{\Pi}$ be the function that maps the states of J to its pebble locations. We shall prove, by induction on the number of steps taken, that if J is in state Q in step t on input $G \in \mathcal{G}_k(E)$, then J' is in the same state in step t on input $C(G) \in \mathcal{G}_{k-1}$ and $C_E(\widehat{\Pi}(Q))$ specifies the locations of its pebbles in that step. This proves the claim. Initially, J and J' are at state Q_0. Both $\widehat{\Pi}(Q_0)$ and $C_E(\widehat{\Pi}(Q_0))$ specify that all pebbles are on node s.

Assume that at step t, J is in state Q on input G and that, at the same step, J' is in state Q on $C(G)$ and its pebble locations are specified by $C_E(\widehat{\Pi}(Q))$. The move of J' will be determined by the move of J. For the first substep there are three cases.

Case 1. If J *does nothing*, then J' also does nothing.

Case 2. If J *walks pebble* P *along the edge with label* ℓ, then there are three subcases, depending on the node u that P was on.

(2a) If u is the lost node or the node w_j for some $j \in [1..2^k\chi]$, then the destination, v, of P is fixed and $C_E(v)$ is either the lost node, the node $u_{\langle 1,j\rangle}$, or w_{j+1}. Hence, J' node-jumps pebble P to $C_E(v)$.

(2b) If u is a hard node in layer D of some G_i (i.e., u is a goal node), then the out-edges of u are collapsed edges. In this case, J' walks pebble P along edge ℓ.

(2c) If u is not a goal node, and not an auxiliary node, then the destination, v, is fixed for all $G \in \mathcal{G}_k(E)$. If $C_E(u) \neq C_E(v)$, then u must be a hard node and v must be an easy node. Hence, J' node-jumps pebble P to $C_E(v)$, which is the lost node. If $C_E(u) = C_E(v)$, then J' does nothing.

Case 3. *If* J *node-jumps pebble* P *to node* v, then v must be either the lost node or w_j or $u_{\langle 1,j\rangle}$ for some $j \in [1..2^k\chi]$. Hence $C_E(v)$ is either the lost node, w_j, or $u_{\langle 1,j\rangle}$ for some $j \in [1..2^{k-1}\chi]$. J' just node-jumps pebble P to $C_E(v)$.

Let Π_1 be the pebble locations of J after the first substep and let J assume state Q' in the second substep. In its second substep, J' performs the same sequence of pebble-to-pebble jumps as in the second substep of J and then assumes state Q' if and only if its pebble locations after the first substep is $C_E(\Pi_1)$.

Let us check that in all the above cases the pebble location of J' after the first substep is indeed $C_E(\Pi_1)$. By the inductive hypothesis pebble P of J' was on node $C_E(u)$ before the first substep while pebble P of J was on node u. Since J only moves pebble P from node u to v in the first substep, we just need to show that J' moves P from node $C_E(u)$ to $C_E(v)$ in the first substep. This is obviously true in all the above cases except (2b). In Case (2b), $\langle u, \ell, v\rangle$ is a collapsed edge in G. By the definitions of $\mathcal{G}_k$ and $\mathcal{G}_{k-1}$, $\langle C_E(u), \ell, C_E(v)\rangle$ is an edge in $C(G)$. Hence, pebble P of J' will be on node $C_E(v)$ after the first substep. It follows that J' will also assume state Q' in the second substep. Moreover, in the second substep, J changes the pebble locations from Π_1 to $\widehat{\Pi}(Q')$ by pebble-to-pebble jumps. By construction, J' will also change the pebble locations from $C_E(\Pi_1)$ to $C_E(\widehat{\Pi}(Q'))$. □

10. Proof of the main lemma.

LEMMA 8.1 (repeated). *Let* A *be any sectioned subbranching program derived from some pebble location redundant NNJAG with at most* $S/\log n$ *pebbles. Then, for any* $k \in [1..K]$,

$$Pr_{G\in\mathcal{G}_k}\,[\,w_A(G) \geq 3S/\log n \text{ and } h_A(G) \leq \chi/8\,] < 2^{-2S}.$$

Proof. Recall that every $G \in \mathcal{G}_k$ consists of D^{k-1} graphs, G_1, G_2,..., $G_{D^{k-1}}$, chosen independently from $\mathcal{B}(2^k\chi)$, a graph G' chosen from $\mathcal{H}_{k-1}$ and some fixed auxiliary nodes and edges. Each G_i has $2^{k-1}\chi$ roots. Therefore, there are $(2D)^{k-1}\chi$ roots. Recall as well that $w_A(G)$ denotes how many of the $(2D)^{k-1}\chi$ goals have been discovered and that $h_A(G)$ is a measure of the number of edges queried.

Our proof will concentrate on the traversing of the base edges in G. We assume that G' is fixed and known to A. Hence, the probability is only over the graphs $G_1, \ldots, G_{D^{k-1}} \in \mathcal{B}(2^k\chi)$. Let $\mathcal{B}_k$ be the distribution

$$\{\,\langle G_1, \ldots, G_{D^{k-1}}\rangle \mid G_1, \ldots, G_{D^{k-1}} \in \mathcal{B}(2^k\chi)\,\}.$$

We allow the machine to query any variable $X_{\langle u,\ell\rangle}$ if u is a node in the top layer of a G_i. Moreover, each time a variable $X_{\langle u,\ell\rangle}$ is queried, the following are returned: (1)

the value, v, of $X_{\langle u,\ell\rangle}$; (2) whether v is a goal node; and, if so, (3) its corresponding root node. With these changes, we can assume that the machine does not query any collapsed edge, as there is no need.

Also, we modify A so that it has the following properties: (1) A is a decision tree (i.e., it will not forget the answer to any previous query); (2) A will not repeat any previous query; and (3) each computation path γ in A queries at most $\chi/8$ different (base) edges and discovers at most $3S/\log n$ different goal nodes. (If γ queries more than $\chi/8$ different edges, we will cut it right after it queries the $(\chi/8)$th edge. Similarly, if γ discovers more than $3S/\log n$ different goal nodes, we will cut it right after it discovers the $(3S/\log n)$th node.) It is clear that the modifications will not decrease the probability stated in the lemma. With all the above assumptions and modifications, we just need to show that $\Pr_{G\in\mathcal{B}_k}[\,w_A(G) \geq 3S/\log n\,] \leq 2^{-2S}$.

Just before A starts, each of the $S/\log n$ pebbles may already be partially way down a hard path or even on a goal node. To simplify the analysis, we assume that the goals of those hard paths that initially contain pebbles will be discovered by A. There are at most $S/\log n$ such goals. To pebble the remaining goals, we know, by Lemma 6.1, that the entire hard path must be traversed by the NNJAG. In other words, every edge in the hard path has to be queried by A. Let $w'_A(G)$ be the number of roots such that every edge on its hard path in G has been queried by A. To prove the lemma, it suffices to show that $\Pr_{G\in\mathcal{B}_k}[\,w'_A(G) \geq 2S/\log n\,] < 2^{-2S}$.

Consider an arbitrary computation path γ in A. It can be specified by the sequence of base edges, E_γ, it has queried and the sequence of node names, R_γ, specifying whether a goal node is discovered in each step (and if applicable, its corresponding root). For example, suppose γ queries the variable $X_{\langle u,\ell\rangle}$ which has the value v and then the variable $X_{\langle u',\ell'\rangle}$ which has the value v'. Suppose v is not a goal but v' is the goal node of root r; then $E_\gamma = (\langle u,\ell,v\rangle, \langle u',\ell',v'\rangle)$ and $R_\gamma = (0, r)$ (assuming no root has name 0).

When γ is the computation path followed on input G we will say that "G follows γ." (It might be useful to think of G as being "processed" by A along γ.) First, let us understand what we can deduce about $\vec{\ell}_r(G)$, the sequence of edge labels on the hard path in G rooted at r, given that $G \in \mathcal{B}_k(E_\gamma)$. (Note that G may not actually follow γ, because it might not agree with R_γ.) We say that a node v is a *collision node* with respect to E_γ if E_γ contains two distinct edges $\langle u,\ell,v\rangle$ and $\langle u',\ell',v\rangle$ with the same destination v. Since v has in-degree at least 2, it is known to be an easy node.

In general, we can classify $\vec{\ell} \in [0..f-1]^{D-1}$ according to γ and r as follows. Suppose we trace out a path through the edges in E_γ, starting at the root r and following the sequence of edge labels $\vec{\ell}$ until the next edge to be taken is not contained in E_γ. Then one of the following three possibilities will occur:

1. The path passes through some collision node with respect to E_γ.
2. The path reaches layer D without passing through any collision node with respect to E_γ.
3. The path stops before reaching layer D and does not pass through any collision node with respect to E_γ.

We define $Y_{\langle\gamma,r\rangle}$ and $Z_{\langle\gamma,r\rangle}$ to contain the vectors $\vec{\ell} \in [0..f-1]^{D-1}$ such that when the above procedure is applied, the second and third outcomes occur, respectively.

Claim 1. *For any computation path γ in A, any input graph $G \in \mathcal{B}_k(E_\gamma)$, and any root r, $\vec{\ell}_r(G) \in Y_{\langle\gamma,r\rangle} \cup Z_{\langle\gamma,r\rangle}$.*

Proof. For any $\vec{\ell} \notin Y_{\langle\gamma,r\rangle} \cup Z_{\langle\gamma,r\rangle}$ and any input $G \in \mathcal{B}_k(E_\gamma)$, the path from

the root r labeled with $\vec{\ell}$ in G contains a collision node. Since collision nodes have in-degree at least two in E_γ, they do not lie on the hard path. Thus, $\vec{\ell}_r(G) \neq \vec{\ell}$. ∎

DEFINITION 3. *For any computation path γ and any $G \in \mathcal{B}_k(E_\gamma)$, $Prog_{\langle\gamma,r\rangle}(G)$ is defined as the random variable indicating that all the edges in the hard path in G rooted at r are mentioned in E_γ.*

Obviously, $Prog_{\langle\gamma,r\rangle}(G)$ is true if $\vec{\ell}_r(G) \in Y_{\langle\gamma,r\rangle}$ and false if $\vec{\ell}_r(G) \in Z_{\langle\gamma,r\rangle}$. If G actually follows γ and $Prog_{\langle\gamma,r\rangle}(G)$ is true, then the goal of root r is discovered. Let $y_{\langle\gamma,r\rangle} = |Y_{\langle\gamma,r\rangle}|$ and $z_{\langle\gamma,r\rangle} = |Z_{\langle\gamma,r\rangle}|$. Briefly, the probability that $Prog_{\langle\gamma,r\rangle}(G)$ is true, given that $G \in \mathcal{B}_k(E_\gamma)$ is approximately $\frac{y_{\langle\gamma,r\rangle}}{y_{\langle\gamma,r\rangle}+z_{\langle\gamma,r\rangle}}$, because all $\vec{\ell} \in Y_{\langle\gamma,r\rangle} \cup Z_{\langle\gamma,r\rangle}$ have about the same probability to be chosen as $\vec{\ell}_r(G)$.

Let $D' = D/8$. We say that root r is a *high-collision root* with respect to the computation path γ if $y_{\langle\gamma,r\rangle} + z_{\langle\gamma,r\rangle} \leq f^{D'}$. Otherwise, we say that it is a *low-collision root* with respect to γ. We say that γ is a *high-collision computation* if there are at least $S/\log n$ high-collision roots with respect to γ. Otherwise, we say that it is a *low-collision computation*. Let $\mathcal{C}$ be the set of all high-collision computation paths. Then

$$\begin{aligned}
&\Pr_{G\in\mathcal{B}_k}\left[\, w'_A(G) \geq \frac{2S}{\log n}\,\right] \\
&\leq \sum_{\gamma\in\mathcal{C}} \Pr_{G\in\mathcal{B}_k}\left[\, G \text{ follows } \gamma\,\right] + \sum_{\gamma\notin\mathcal{C}} \Pr_{G\in\mathcal{B}_k}\left[\, w'_A(G) \geq \frac{2S}{\log n} \text{ and } G \text{ follows } \gamma\,\right] \\
&\equiv SUM_1 + SUM_2.
\end{aligned}$$

By Claim 2 in section 11, SUM_1 is at most 2^{-3S}. Consider SUM_2. If both events "$w'_A(G) \geq 2S/\log n$" and "G follows γ" occur, there exist at least $2S/\log n$ roots r such that $Prog_{\langle\gamma,r\rangle}(G)$ is true; i.e., all the edges on the hard path rooted at r in G are in E_γ. For $\gamma \notin \mathcal{C}$, at least $S/\log n$ of these are low-collision roots with respect to E_γ.

DEFINITION 4. *For any computation path γ and any $G \in \mathcal{B}_k(E_\gamma)$, $w''_\gamma(G)$ is defined as the number of roots r such that r is a low-collision root with respect to γ and $Prog_{\langle\gamma,r\rangle}(G)$ is true.*

Then

$$SUM_2 \leq \sum_{\gamma\notin\mathcal{C}} \Pr_{G\in\mathcal{B}_k}\left[\, w''_\gamma(G) \geq \frac{S}{\log n} \text{ and } G \text{ follows } \gamma\,\right].$$

Since "G follows γ" implies "$G \in \mathcal{B}_k(E_\gamma)$,"

$$\begin{aligned}
SUM_2 &\leq \sum_{\gamma\notin\mathcal{C}} \Pr_{G\in\mathcal{B}_k}\left[\, w''_\gamma(G) \geq \frac{S}{\log n} \text{ and } G \in \mathcal{B}_k(E_\gamma)\,\right] \\
&\leq \max_{\gamma\notin\mathcal{C}} \Pr_{G\in\mathcal{B}_k}\left[\, w''_\gamma(G) \geq \frac{S}{\log n} \mid G \in \mathcal{B}_k(E_\gamma)\,\right] \times \sum_{\gamma\notin\mathcal{C}} \Pr_{G\in\mathcal{B}_k}\left[\, G \in \mathcal{B}_k(E_\gamma)\,\right].
\end{aligned}$$

We claim that, for each graph $G \in \mathcal{B}_k$, there are at most 2^{6S} different computation paths γ for which G satisfies E_γ. To see this, observe that every computation path γ in A queries at most $\chi/8$ different base edges and discovers at most $3S/\log n$ different goal nodes, each having at most n name choices for its corresponding root. Hence, there are at most $\binom{\chi/8}{3S/\log n} \times n^{3S/\log n} \leq 2^{6S}$ different sequences R_γ. If there were more than 2^{6S} different computation paths γ's such that G satisfies E_γ, then there

exist two different computation paths γ and γ' such that $R_\gamma = R_{\gamma'}$ and G satisfies both E_γ and $E_{\gamma'}$. For γ and γ' to be different, there must be an edge $\langle u, \ell, v\rangle$ in E_γ and an edge in $\langle u, \ell, v'\rangle$ in $E_{\gamma'}$ such that $v \neq v'$. Then G cannot satisfy both E_γ and $E_{\gamma'}$, a contradiction. Hence, our claim follows. From this claim, we have $\sum_{\gamma \notin \mathcal{C}} \Pr_{G\in\mathcal{B}_k}[\, G \in \mathcal{B}_k(E_\gamma)\,] \leq 2^{6S}$ and thus,

$$SUM_2 \leq \max_{\gamma\notin\mathcal{C}} \Pr_{G\in\mathcal{B}_k}\left[w''_\gamma(G) \geq \frac{S}{\log n} \mid G \in \mathcal{B}_k(E_\gamma)\right] \times 2^{6S}.$$

Claim 4 of section 12 shows that $\Pr_{G\in\mathcal{B}_k}\left[\, w''_\gamma(G) \geq S/\log n \mid G \in \mathcal{B}_k(E_\gamma)\,\right]$ is at most 2^{-9S} for any γ in A. In conclusion,

$$\begin{aligned} SUM_1 + SUM_2 &\leq 2^{-3S} + 2^{-9S+6S} \\ &\leq 2^{-3S+1} \\ &\leq 2^{-2S}, \end{aligned}$$

where all inequalities hold for big enough n. Therefore, Lemma 8.1 (main lemma) follows. ∎

11. Bounding SUM_1. This section bounds the first sum, SUM_1, at the end of the proof for Lemma 8.1 (main lemma).

CLAIM 2. $\sum_{\gamma\in\mathcal{C}} \Pr_{G\in\mathcal{B}_k}[G \textit{ follows } \gamma] \leq 2^{-3S}$.

Proof. We first define two games called the *edge-collision game* and the *branching-process game*. Let S_{ed} and S_{br} be the random variables indicating the success of each game, respectively. We shall show that $\sum_{\gamma\in\mathcal{C}} \Pr_{G\in\mathcal{B}_k}[\, G \text{ follows } \gamma\,] \leq \Pr[\, S_{ed}\,] \leq \Pr[\, S_{br}\,] \leq 2^{-3S}$. ∎

11.1. The edge-collision game. The *edge-collision game* is defined as follows. D^{k-1} graphs $G_1, G_2, \ldots, G_{D^{k-1}}$ are chosen randomly and independently from $\mathcal{B}(2^k\chi)$. The player is informed of the hard path of each root in each G_i. He then queries edges of the G_is one at a time. When the player queries an edge, he specifies $\langle u, \ell\rangle$, where u is a node and $\ell \in [0..f-1]$ is an edge label. The destination node v of the edge $\langle u, \ell, v\rangle$ is then revealed to the player. Based on the result of the previous queries, he chooses the next edge to query. He is allowed to query at most $\chi/8$ edges total.

The aim of the player is to minimize the number of leaves of certain trees associated with the queried edges. To be precise, let E be the sequence of base edges of the input graph G that the player has queried during the game. Recall that in section 10 a node v is called a *collision node* with respect to E if it is the destination of more than one edge in E. In this game, each edge in E will be in one of two conditions: *alive* or *dead*. An edge is said to be dead if its destination node is (1) a collision node or (2) the source of a previously queried edge. Otherwise, it is alive. We shall construct from E a collection of f-ary trees by taking the following steps.

Step 1. If E does not contain a path from any root r to a node v, then delete v from G (along with all its in-edges and out-edges).

Step 2. Delete all the nodes (along with their in-edges and out-edges) that are proper descendents in E of the destinations of the dead edges. The dead edges and their destinations are kept. The remaining edges in E that are alive are called the *y-edges* and their destination nodes are called the *y-nodes*. Each such node has a unique path from some root to it and that path contains no dead edges. Hence, the y-edges form a collection of disjoint f-ary trees.

*Step*3. "Fill up" the above trees so that each node has exactly f outgoing edges. More precisely, for each y-node that does not have exactly f outgoing edges (counting the dead edges), add the missing edges and attach to each such edge a complete f-ary tree, of appropriate depth, such that its leaves are at layer D. The nodes and edges that are added in this way are referred to as the *z-nodes* and the *z-edges*. They do not correspond to actual nodes and edges in the input graph G. Note that each z-node also has a unique path from some root to it and that path contains no dead edges.
We shall measure the performance of the game player by two sets of parameters. They are somewhat similar to $y_{\langle\gamma,r\rangle}$, $z_{\langle\gamma,r\rangle}$ defined in section 10. Define $\tilde{y}_r$ and $\tilde{z}_r$ (the performance parameters) as the number of y- and z-nodes at layer D that are descendents of the root node r. Again, a root r is said to be a *high-collision* root if $\tilde{y}_r + \tilde{z}_r \leq f^{D'}$, where $D' = D/8$ as defined in section 10. The goal of the player is to create as many high-collision roots as possible. More precisely, the player wins if there are more than $S/\log n$ high-collision roots. Let S_{ed} be the indicator variable of this event.

Note that the edge-collision game player can query whatever edges queried by A and hence ensure that $E_\gamma \subseteq E$, where E_γ and E are the set of edges queried by A and the game player, respectively. Consider the conditions for a root to be a high-collision root. In A, a root r is a high-collision root with respect to E_γ if $y_{\langle\gamma,r\rangle} + z_{\langle\gamma,r\rangle} \leq f^{D'}$, i.e., the number of vectors $\vec{\ell} \in Y_{\langle\gamma,r\rangle} \cup Z_{\langle\gamma,r\rangle}$ is small. If a vector $\vec{\ell} \in [0..f-1]^{D-1}$ is not in $Y_{\langle\gamma,r\rangle} \cup Z_{\langle\gamma,r\rangle}$, then the path obtained by following $\vec{\ell}$ from r must contain a collision node in E_γ. The same node will also be a collision node in the edge-collision game, provided $E_\gamma \subseteq E$. It follows that a high-collision root in A will also be a high-collision root in the game. Therefore, $\sum_{\gamma\in\mathcal{C}} \Pr_{G\in\mathcal{B}_k}[\,G \text{ follows } \gamma\,] \leq \Pr[\,S_{ed}\,]$.

11.2. The branching-process game. Before we introduce the branching-process game, we introduce some machinery that will be useful in bounding its probability of success.

Consider a rooted, complete, f-ary tree of depth d. We allow every edge of such a tree to *die* independently of all other edges with a fixed probability α. A node u is said to be *alive* if and only if no edge along the unique path from the root to u is dead. If Z_i denotes the number of alive vertices at level i, then the sequence $Z_0 = 1, Z_1, \ldots, Z_i, \ldots$ forms a branching process [2]. We will be interested in the distribution of the number of live vertices with depth d, i.e., the random variable Z_d. The expected number of live children for an alive node is $(1-\alpha)f$, and the expected value of Z_d is $((1-\alpha)f)^d$. More precisely, the generating function for the offspring distribution in this branching process is $g(x) = (\alpha+(1-\alpha)x)^f$ (i.e., the probability that a node has i out-edges that do not die is the coefficient of x^i in $g(x)$). A well-known fact is that if $(1-\alpha)f > 1$, then $\Pr[\,Z_d = 0\,] = \xi$, where ξ is the unique $x \in (0,1)$ such that $g(x) = x$. Moreover $\eta = g'(\xi) < 1$. The following lemma states that the probability that Z_d is much smaller than its expected value is not much greater than the probability that it is 0.

LEMMA 11.1 (see [25]). *If $(1-\alpha)f > 1$, then for $\tilde{d} \in [1,\ldots,d]$ such that $d-\tilde{d} \to \infty$ and $d \to \infty$,*

$$\Pr\left[Z_d \leq ((1-\alpha)f)^{\tilde{d}}\right] \leq \xi + O(\eta^{d-\tilde{d}}).$$

In the branching-process game we will consider a variant of the above trees defined in section 11.1, with depth D and out-degree f. For these trees, the variation lies in the existence of a *fixed* path from the root to some leaf whose edges are guaranteed

to survive; all other edges die independently with probability 1/4. Such a tree is said to *wither* if it has at most $f^{D'}$ live leaves. We want to bound the probability, ρ, that this happens.

To allow for a uniform treatment we first convert the f-ary trees to binary trees when $f > 2$. In particular, if $f > 2$, then $f = O(n^\epsilon)$ for some $\epsilon > 0$, and hence we can assume that f is a power of 2. Thus, we replace each node v and its f edges/children with a complete binary tree T_v of depth $\log f$, i.e., with f leaves. If v is not on the path that is guaranteed to survive, then all the edges in T_v die independently, with probability 1/4. Otherwise, the edges along a unique path of T_v (the path corresponding to the edge guaranteed to live) are guaranteed to survive while the rest die independently with probability 1/4. It is easy to see, inductively, that for any set of nodes at depth $i \log f$, $i = 0 \ldots D$, in the resulting binary tree, the probability of being alive is no more than that of the corresponding nodes at depth i in the f-ary tree. Moreover, $D = \lceil 80 \log n \rceil / \log f$, for all f, and hence it will suffice to prove the bound for $f = 2$ ($D = \lceil 80 \log n \rceil$).

Let v_i be the node at depth i which is on the path whose edges are guaranteed to survive and let v_i' be the sibling of v_i. Let ρ' be the probability that for all v_i, $i \in [1..2D/3]$, either v_i' is dead or v_i' is alive, but the subtree rooted at v_i', $T_{v_i'}$ has at most $f^{D'}$ live nodes, at depth D. Clearly, $\rho \le \rho'$. Each $T_{v_i'}$, is a complete binary tree of depth $D - i$, where every edge dies with probability 1/4. For $\alpha = 1/4$ and $f = 2$, we have $\xi = 1/9$ and $\eta = 1/2$. As D tends to infinity with n and $T_{v_i'}$ has depth at least $D/3$, we can apply Lemma 11.1 to bound the probability that $T_{v_i'}$ has fewer than $f^{D'}$ live nodes at depth D (given that v_i' is alive) by $1/9 + O((1/2)^{D-i-D'})$. Thus, the probability that $T_{v_i'}$ has at most $f^{D'}$ live leaves at depth D is no more than $1/4 + 1/9 + O((1/2)^{D-i-D'}) \le 1/4 + 1/9 + O((1/2)^{D/3-D'}) = \beta < 1/2$, since $D' = D/8$, $D = \lceil 80 \log n \rceil$ and n can be arbitrarily large. Since the $2D/3$ subtrees grow independently, we get $\rho \le \rho' \le \beta^{2D/3} \in O(n^{-5})$.

In the branching-process game in our graph we say that the root r of the same type of tree above is a *high-collision* root if the tree rooted at r withers. Since there are at most n roots, the expected number of high collision roots is $\mu \le n \times \rho \in O(n^{-4})$. Let S_{br} be the random variable indicating the event that there are more than $S/\log n$ high-collision roots. As each tree grows independently of the others, we can apply the Chernoff bound and prove that

$$\begin{aligned}
&\Pr[\, S_{br} \,] \\
&= \Pr\left[\text{ number of high collision roots } \ge \left(\frac{S}{\mu \log n}\right) \mu \right] \\
&\le 2^{-(\frac{S}{\log n})(\log(\frac{S}{\mu \log n}) - \log e)}.
\end{aligned}$$

Since $\mu \in O(n^{-4})$, $\Pr[\, S_{br} \,] \le 2^{-3S}$.

11.3. Branching-process game versus edge-collision game. This subsection proves the second inequality mentioned in the proof of Claim 2.

LEMMA 11.2. *The success of the edge-collision game is probabilistically dominated by the success of the branching processes game, i.e.,* $\Pr[S_{ed}] \le \Pr[S_{br}]$.

Proof. The edge-collision game starts by random choice of the graphs $G_1, G_2, \ldots, G_{D^{k-1}}$ in $\mathcal{B}(2^k\chi)$. For each $i \in [1..D^{k-1}]$, the first step in choosing G_i according to the distribution $\mathcal{B}(2^k\chi)$ is to randomly partition the $2^k\chi$ nodes at each layer into $2^k\chi/2$ easy nodes and $2^k\chi/2$ hard nodes and to choose the hard path rooted at each root

r. This information is revealed to the player. The edges in these paths correspond to the edges in the branching-process game that are guaranteed to live.

The next step in choosing G_i according to the distribution $\mathcal{B}(2^k\chi)$ is to choose for every remaining edge its destination among the $2^k\chi/2$ easy nodes at the next layer. This needs to be done only for those edges queried by the player. For each $i \in [1..D^{k-1}]$, for each layer $d \in [1..D]$, and for each $\tau \in [1..\chi/8]$, define the variable $v_{\langle i,d,\tau\rangle}$ to uniformly and independently take on a value from $[1..2^k\chi/2]$. Suppose that the player is querying $\langle u, \ell\rangle$, where u is a node at layer d in the graph G_i, and this is the τth query to easy edges at this layer in this graph. Then the variable $v_{\langle i,d,\tau\rangle} \in [1..2^k\chi/2]$ specifies the other endpoint of edge $\langle u, \ell\rangle$ among the $2^k\chi/2$ easy nodes at the next layer.

Consider the edge $\langle u, \ell\rangle$ queried at time t for some $t \in [1..\chi/8]$. Before this query, we do not know in advance which edges will be queried after time t because the player is able to choose them dynamically based on the result of the current query. However, the random variables $v_{\langle i,d,\tau\rangle}$ do tell us the resulting destinations of all the edges queried or to be queried. Together with the knowledge of the source of edges queried before time t, we can tell whether the current edge will die. Specifically, suppose $\langle u, \ell\rangle$ is the τth edge queried at layer d in G_i. Then it will die if either (1) there is another query at this layer $\tau' \in [1..\chi/8]$ with the same destination, i.e., $v_{\langle i,d,\tau\rangle} = v_{\langle i,d,\tau'\rangle}$ for some $\tau' \neq \tau$ or (2) the destination is the source of some edge queried before time t.

In order to compare the success probability of the edge-collision game and the branching-process game, let us first define random variables that will indicate which edges die in the branching-process game. For each root r in each of the graphs G_i, there is a corresponding root in the branching-process game. Consider the complete f-ary tree of height D rooted at such a root r. A specific edge in this tree can be specified by a string $\vec{\ell} \in [0..f-1]^*$. For each such edge, let $x_{\langle r,\vec{\ell}\rangle} \in \{0,1\}$ be the random variable indicating whether this edge dies. If the edge is one of the edges that are guaranteed to live in the branching-process game, i.e., the fixed hard path in the input graph, then $\Pr[x_{\langle r,\vec{\ell}\rangle}] = 0$. Otherwise, $\Pr[x_{\langle r,\vec{\ell}\rangle}] = 1/4$ independent of the other x variables.

Now consider a fixed algorithm for the edge-collision game. For each intermediate time step $t \in [0..\chi/8]$, we define the tth game as follows. The game starts with t time steps of the fixed algorithm for the edge-collision game. Let E_t be the resulting base edges queried. We want these edges to die in the tth game if and only if they die in the edge-collision game. As previously mentioned, the edge associated with $v_{\langle i,d,\tau\rangle}$ is dead if and only if (1) there is another query at this layer $\tau' \in [1..\chi/8]$ with the same destination, i.e., $v_{\langle i,d,\tau\rangle} = v_{\langle i,d,\tau'\rangle}$, or (2) the destination, i.e., $v_{\langle i,d,\tau\rangle}$, is the source of an edge queried before time t. In (1), the query associated with $v_{\langle i,d,\tau'\rangle}$ can occur either before or after time step t. Either way, we consider the edge associated with $v_{\langle i,d,\tau\rangle}$ dead. Given which edges in E_t have died, the set E_t can be transformed, as described in Steps (1) through (3) in section 11.1, into a collection of f-ary trees made up of y-nodes and y-edges (the living ones), z-nodes and z-edges, and some collision nodes and dead edges. The tth game is completed by finishing the branching process on the z-edges. Namely, each such edge will live or die according to the corresponding random variable $x_{\langle r,\vec{\ell}\rangle} \in \{0,1\}$. A node u in the resulting collection of trees is said to be *alive* if all the edges on the path from the root of the tree to u are alive. A root is said to be a *high-collision* root if it has at most $f^{D'}$ living nodes in layer D. The tth game succeeds if there are more than $S/\log n$ high-collision roots. Let S_t be the random variable indicating the success of the game.

Observe that the 0th game is simply the branching-process game and hence $S_{br} = S_0$. The $(\chi/8)$th game differs from the edge-collision game only in that in the edge-collision game all the z-nodes and z-edges added in Step 3 are treated as live while in the $(\chi/8)$th game some of the z-edges may die according to the $x_{\langle r,\vec{\ell}\rangle}$ variables. The additional children at layer D hurt only the edge-collision game player. Therefore, $\Pr\left[\, S_{(\chi/8)} \,\right] \geq \Pr\left[\, S_{ed} \,\right]$. What remains to be proved is that, for every $t \in [1..\chi/8]$, $\Pr\left[\, S_{t-1} \,\right] \geq \Pr\left[\, S_t \,\right]$.

Let $\vec{V}_{(<t)}$ specify a possible computation up to and including the $(t-1)$st query. It will specify the values of $t-1$ of the $v_{\langle i,d,\tau\rangle}$ variables. Which of them are specified will depend dynamically on the computation. The computation $\vec{V}_{(<t)}$ will also specify the set of queried edges in the graph E_{t-1} and the next query $\langle u,\ell\rangle$ made by the player. Let the node u be on layer d of G_i and the query be the τth one at this layer in this graph.

Let us consider the following cases. In the first case, E_{t-1} does not contain a unique path with no dead edges from a root to u. In this case, the descendant nodes and edges of node u will be deleted from the y-node tree, both in the $(t-1)$st game and in the tth game. Hence, whether this edge dies has no effect on either game. In the second case, $\langle u,\ell\rangle$ is on a hard path. For both games, the edge is guaranteed to live.

In the third case, E_{t-1} contains a unique path with no dead edges from a root to u and $\langle u,\ell\rangle$ is not on a hard path. Let r and $\vec{\ell}$ specify the root and the labels in this path. In the $(t$–$1)$st game, whether the edge from $\langle u,\ell\rangle$ dies is specified by the variable $x_{\langle r,\vec{\ell}\rangle} \in \{0,1\}$. In the tth game, the destination of the edge from $\langle u,\ell\rangle$ is specified by the variable $v_{\langle i,d,\tau\rangle}$. Consider one setting $\vec{V}_{(>t)}$ of all the $v_{\langle i',d',\tau'\rangle}$ variables other than those set by $\vec{V}_{(<t)}$ and other than the variable $v_{\langle i,d,\tau\rangle}$. Consider as well one setting $\vec{X}_{(\neq t)}$ of all the x variables other than $x_{\langle r,\vec{\ell}\rangle}$.

Compare $\Pr[S_{t-1} \mid \vec{V}_{(<t)}, \vec{V}_{(>t)}, \vec{X}_{(\neq t)}]$ and $\Pr[S_t \mid \vec{V}_{(<t)}, \vec{V}_{(>t)}, \vec{X}_{(\neq t)}]$. In both cases, the probability is only over the values of $v_{\langle i,d,\tau\rangle}$ and $x_{\langle r,\vec{\ell}\rangle}$. Everything else is fixed by $\vec{V}_{(<t)}$, $\vec{V}_{(>t)}$, and $\vec{X}_{(\neq t)}$. For every value of $v_{\langle i,d,\tau\rangle}$ and $x_{\langle r,\vec{\ell}\rangle}$, which edges die before time step t and which die after time step t is the same for both the $(t-1)$st and the tth game. The only change in the game is whether or not the edge from $\langle u,\ell\rangle$ dies. In the $(t-1)$st game, this edge dies with probability $\Pr[x_{\langle r,\vec{\ell}\rangle} \mid \vec{V}_{(<t)}, \vec{V}_{(>t)}, \vec{X}_{(\neq t)}] = 1/4$. In the tth game, this edge dies if there exists a $\tau' \in [1..\chi/8]$ $(\tau' \neq \tau)$ for which $v_{\langle i,d,\tau\rangle} = v_{\langle i,d,\tau'\rangle}$ or $v_{\langle i,d,\tau\rangle}$ is equal to the source of an edge queried before time t. $\vec{V}_{(<t)}$ and $\vec{V}_{(>t)}$ fix at most $\chi/8-1$ different values of the variables $v_{\langle i,d,\tau'\rangle}$ and at most $t-1 \leq \chi/8-1$ different values as the sources. The value for $v_{\langle i,d,\tau\rangle}$ is chosen uniformly from $[1..2^k\chi/2]$. Therefore, the probability that $v_{\langle i,d,\tau\rangle}$ collides with one of these $\leq 2(\chi/8-1)$ values given $\vec{V}_{(<t)}$, $\vec{V}_{(>t)}$, $\vec{X}_{(\neq t)}$ is at most $1/4$. Having a smaller probability of this edge dying can hurt only the tth game player. We can conclude that

$$\Pr\left[\, S_{t-1} \;\middle|\; \vec{V}_{(<t)}, \vec{V}_{(>t)}, \vec{X}_{(\neq t)} \,\right] \geq \Pr\left[\, S_t \;\middle|\; \vec{V}_{(<t)}, \vec{V}_{(>t)}, \vec{X}_{(\neq t)} \,\right]$$

and hence

$$\Pr\left[\, S_{t-1} \,\right]$$

$$= \sum_{\vec{V}_{(<t)},\vec{V}_{(>t)},\vec{X}_{(\neq t)}} \Pr\left[S_{t-1} \;\middle|\; \vec{V}_{(<t)},\vec{V}_{(>t)},\vec{X}_{(\neq t)} \right] \times \Pr\left[\vec{V}_{(<t)},\vec{V}_{(>t)},\vec{X}_{(\neq t)} \right]$$
$$\geq \sum_{\vec{V}_{(<t)},\vec{V}_{(>t)},\vec{X}_{(\neq t)}} \Pr\left[S_{t} \;\middle|\; \vec{V}_{(<t)},\vec{V}_{(>t)},\vec{X}_{(\neq t)} \right] \times \Pr\left[\vec{V}_{(<t)},\vec{V}_{(>t)},\vec{X}_{(\neq t)} \right]$$
$$= \Pr[\, S_t \,]. \quad \square$$

12. Bounding SUM_2. This section bounds the second sum at the end of the proof of Lemma 8.1. It suffices to show that $\Pr_{G\in\mathcal{B}_k}\left[\, w''_\gamma(G) \geq S/\log n \mid G \in \mathcal{B}_k(E_\gamma) \,\right] \leq 2^{-9S}$. The event $w''_\gamma(G) \geq S/\log n$ happens when at least $S/\log n$ of the low collision roots r have $Prog_{\langle\gamma,r\rangle}(G)$ true. If, for every root r, $Prog_{\langle\gamma,r\rangle}(G)$ were true with a fixed probability independent of the other roots, then we could apply the Chernoff bound directly. However, there are indeed dependencies among different roots. Fortunately, if each event has a low probability of success no matter what outcomes of the other events have, then by the following lemma from Edmonds [14] the Chernoff bound still holds.

LEMMA 12.1 (Lemma 14 of [14]). *Let $\mathcal{R}$ be the set of roots. For each $r \in \mathcal{R}$, let $\hat{x}_r \in \{\,0,1\,\}$ be the random variable indicating the success of the rth trial. For each $r \in \mathcal{R}$ and $\mathcal{O} \in \{\,0,1\,\}^{\mathcal{R}-\{\,r\,\}}$, let $Z_{\langle r,\mathcal{O}\rangle} = \Pr[\hat{x}_r = 1 \mid \mathcal{O}]$, where $\mathcal{O}$ indicates that the other trials have the stated outcomes. If for every r and every possible outcome of the other trials $\mathcal{O}$, $Z_{\langle r,\mathcal{O}\rangle} \leq \rho$, then for every $\delta > 1$, $\Pr[\sum_{r\in\mathcal{R}} \hat{x}_r \geq 2\delta\rho|\mathcal{R}|] \leq 2^{-0.38\delta\rho|\mathcal{R}|}$.*

Proof. Let $\hat{X} = \sum_{r\in\mathcal{R}} \hat{x}_r$. To bound $\Pr[\hat{X} \geq 2\delta\rho|\mathcal{R}|]$, we will consider a sequence of random variables, x_r, $r \in \mathcal{R}$, defined as follows: for x_1, we choose uniformly at random $\lambda_1 \in [0,1]$ and set $x_1 = 1$ if and only if $\lambda_1 \leq \Pr[\,\hat{x}_1 = 1\,]$. In general, if we have set $x_1 = a_1, \ldots, x_i = a_i$, we choose uniformly at random $\lambda_{i+1} \in [0,1]$ and set $x_{i+1} = 1$ if and only if $\lambda_{i+1} \leq \Pr[\,\hat{x}_{i+1} = 1 \mid \hat{x}_1 = a_1 \wedge \cdots \wedge \hat{x}_i = a_i\,]$. Clearly, the sequences $\hat{x}_r$ and x_r are identically distributed and $\Pr[\,x_r = 1\,] \leq \rho$ for all $r \in \mathcal{R}$.

Consider now a sequence of random variables y_r defined by $y_r = 1$ if and only if $\lambda_r \leq \rho$, $r \in \mathcal{R}$, where λ_r is as above. By construction, $x_r \leq y_r$ for all $r \in \mathcal{R}$. Hence, if $X = \sum_{r\in\mathcal{R}} x_r$ and $Y = \sum_{r\in\mathcal{R}} y_r$, then $X \leq Y$. Moreover Y is the sum of $|R|$ independent Boolean random variables. Applying the Chernoff bound we get, for $\delta > 1$,

$$\square\ \Pr\left[\, \hat{X} \geq 2\delta\rho|\mathcal{R}| \,\right] = \Pr[\, X \geq 2\delta\rho|\mathcal{R}|\,] \leq \Pr[\, Y \geq 2\delta\rho|\mathcal{R}|\,] \leq 2^{-0.38\delta\rho|\mathcal{R}|}.$$

In Claim 3, we first show that the probability that $Prog_{\langle\gamma,r\rangle}(G)$ is true is small for low-collision roots r. Then we will apply Lemma 12.1 in Claim 4 to get the desired bound for the second sum.

CLAIM 3. *For any computation path γ in A, any root r, and any subset $\mathcal{O}$ of roots indicating for which roots r' other than r, $Prog_{\langle\gamma,r'\rangle}(G)$ is true,*

$$\Pr_{G\in\mathcal{B}_k}\left[Prog_{\langle\gamma,r\rangle}(G) \mid G \in \mathcal{B}_k(E_\gamma) \text{ and } \mathcal{O} \right] \leq \frac{4}{3}\frac{y_{\langle\gamma,r\rangle}}{y_{\langle\gamma,r\rangle} + z_{\langle\gamma,r\rangle}}.$$

Proof. Let us consider a fixed γ and r. Recall that $\vec{\ell}_r(G)$ is the random variable indicating the vector of edge labels on the hard path rooted at r in graph G drawn from $\mathcal{B}_k$ and that $\vec{\ell}_r(G) \in Y_{\langle\gamma,r\rangle} \cup Z_{\langle\gamma,r\rangle}$. Recall as well that $Prog_{\langle\gamma,r\rangle}(G)$ is true if and only if $\vec{\ell}_r(G) \in Y_{\langle\gamma,r\rangle}$. We shall drop the subscripts in $Y_{\langle\gamma,r\rangle}$, $Z_{\langle\gamma,r\rangle}$, $y_{\langle\gamma,r\rangle}$, $z_{\langle\gamma,r\rangle}$, $\vec{\ell}_r(G)$, $Prog_{\langle\gamma,r\rangle}(G)$, and E_γ, when there is no chance of confusion. We shall also write

$\Pr_{G\in\mathcal{B}_k}[\,\cdot \mid G \in \mathcal{B}_k(E_\gamma)\,]$ as $\Pr[\,\cdot \mid E\,]$. Note that

$$\begin{aligned}
&\Pr[\,\mathit{Prog}(G) \mid E \text{ and } \mathcal{O}\,] \\
&= \frac{\Pr[\,\mathit{Prog}(G) \mid E \text{ and } \mathcal{O}\,]}{\Pr[\,\mathit{Prog}(G) \mid E \text{ and } \mathcal{O}\,] + \Pr[\,\neg\mathit{Prog}(G) \mid E \text{ and } \mathcal{O}\,]} \\
&= \frac{\sum_{\vec{\ell}\in Y} \Pr\left[\,\vec{\ell}(G) = \vec{\ell}\mid E \text{ and } \mathcal{O}\,\right]}{\sum_{\vec{\ell}\in Y} \Pr\left[\,\vec{\ell}(G) = \vec{\ell}\mid E \text{ and } \mathcal{O}\,\right] + \sum_{\vec{\ell}\in Z} \Pr\left[\,\vec{\ell}(G) = \vec{\ell}\mid E \text{ and } \mathcal{O}\,\right]}.
\end{aligned}$$

Let $\vec{\ell}_y$ be the vector in Y that maximizes $\Pr[\vec{\ell}(G) = \vec{\ell}\mid E \text{ and } \mathcal{O}]$ over $\vec{\ell}\in Y$ and let $\vec{\ell}_z$ be the vector in Z that minimizes $\Pr[\vec{\ell}(G) = \vec{\ell}\mid E \text{ and } \mathcal{O}]$ over $\vec{\ell}\in Z$. The above probability is at most

$$\begin{aligned}
&\frac{y \times \Pr\left[\,\vec{\ell}(G) = \vec{\ell}_y \mid E \text{ and } \mathcal{O}\,\right]}{y \times \Pr\left[\,\vec{\ell}(G) = \vec{\ell}_y \mid E \text{ and } \mathcal{O}\,\right] + z \times \Pr\left[\,\vec{\ell}(G) = \vec{\ell}_z \mid E \text{ and } \mathcal{O}\,\right]} \\
&\qquad = \frac{y}{y + \frac{\Pr\left[\,\vec{\ell}(G)=\vec{\ell}_z \mid E \text{ and } \mathcal{O}\,\right]}{\Pr\left[\,\vec{\ell}(G)=\vec{\ell}_y \mid E \text{ and } \mathcal{O}\,\right]} \times z}.
\end{aligned}$$

What remains to be proven is that

$$\frac{\Pr\left[\,\vec{\ell}(G) = \vec{\ell}_z \mid E \text{ and } \mathcal{O}\,\right]}{\Pr\left[\,\vec{\ell}(G) = \vec{\ell}_y \mid E \text{ and } \mathcal{O}\,\right]} \geq \frac{3}{4}.$$

Let $N_y(G)$ and $N_z(G)$, respectively, be the set of edges on the path with label $\vec{\ell}_y$ and $\vec{\ell}_z$ from root r in G. Let $H(G)$ be the random variable specifying the hard path rooted at r in G, i.e., both the nodes and the labels $\vec{\ell}(G)$. The fact that $\vec{\ell}_y \in Y$ means that the path following the edge labels in $\vec{\ell}_y$ is totally contained in E. Therefore, $N_y(G)$ is equal to some fixed value N_y determined by E. Then the statements $\vec{\ell}(G) = \vec{\ell}_y$ and $\vec{\ell}(G) = \vec{\ell}_z$ are equivalent to $H(G) = N_y$ and $H(G) = N_z(G)$, respectively. The possible values N_z for the random variable $N_z(G)$ (i.e., the path in G rooted at r with edge labels $\vec{\ell}_z$) can be divided into two sets. Let $N_z \in \mathcal{A}_z$ if and only if some edge $\langle u, \ell, v\rangle$ in N_z has the same destination with a different edge $\langle u', \ell', v'\rangle$ in E, i.e., $v = v'$ but $\langle u, \ell\rangle \neq \langle u', \ell'\rangle$. In this case, N_z cannot be the hard path. That is, for $N_z \in \mathcal{A}_z$, $\Pr[\,H(G) = N_z \mid E \text{ and } \mathcal{O}\,] = 0$. Now consider an $N_z \notin \mathcal{A}_z$. Given that G contains $E \cup N_z$ and satisfies $\mathcal{O}$, we argue that it is equally likely for $H(G)$ to be N_y or N_z. To see this, first observe that $\mathcal{O}$ does not affect how $H(G)$ can be chosen because $\mathcal{O}$ is a condition on the hard paths of roots other than r. Second, both fixed paths N_y and N_z, started from root r, are contained in $E \cup N_z$. Furthermore, neither N_y nor N_z contains any collision node with respect to $E \cup N_z$. By symmetry, it is equally likely for N_y and N_z to be chosen as $H(G)$. Hence,

$$\begin{aligned}
&\Pr[\,H(G) = N_y \mid N_z(G) = N_z \text{ and } E \text{ and } \mathcal{O}\,] \\
&= \Pr[\,H(G) = N_z \mid N_z(G) = N_z \text{ and } E \text{ and } \mathcal{O}\,].
\end{aligned}$$

Note as well that "$H(G) = N_z$" implies "$N_z(G) = N_z$." Therefore,

$$\Pr[\,H(G) = N_y \text{ and } N_z(G) = N_z \mid E \text{ and } \mathcal{O}\,]$$

$$= \Pr\left[\, H(G) = N_z \text{ and } N_z(G) = N_z \mid E \text{ and } \mathcal{O} \,\right]$$
$$= \Pr\left[\, H(G) = N_z \mid E \text{ and } \mathcal{O} \,\right].$$

The above ratio then becomes

$$\frac{\Pr\left[\, \vec{\ell}(G) = \vec{\ell}_z \mid E \text{ and } \mathcal{O} \,\right]}{\Pr\left[\, \vec{\ell}(G) = \vec{\ell}_y \mid E \text{ and } \mathcal{O} \,\right]}$$
$$= \frac{\sum_{N_z \notin \mathcal{A}_z} \Pr\left[\, H(G) = N_z \mid E \text{ and } \mathcal{O} \,\right]}{\Pr\left[\, H(G) = N_y \mid E \text{ and } \mathcal{O} \,\right]}$$
$$= \frac{\sum_{N_z \notin \mathcal{A}_z} \Pr\left[\, H(G) = N_y \text{ and } N_z(G) = N_z \mid E \text{ and } \mathcal{O} \,\right]}{\Pr\left[\, H(G) = N_y \mid E \text{ and } \mathcal{O} \,\right]}$$
$$= \Pr\left[\, N_z(G) \notin \mathcal{A}_z \mid H(G) = N_y \text{ and } E \text{ and } \mathcal{O} \,\right].$$

The input distribution $\mathcal{B}_k$ first chooses the hard paths. Then every other edge is added independently at random. If $N_z(G)$ is not a hard path, at each level $i \in [2..D]$, its node is chosen from the $2^k\chi/2$ easy nodes at this level. A sufficient condition for $N_z(G)$ not to be in $\mathcal{A}_z$ is that for all its edges not fixed by E, their destinations do not collide with any node mentioned in E. Let h_i be the number of nodes mentioned in E at level i that $N_z(G)$ must avoid. It follows that

$$\Pr\left[\, N_z(G) \notin \mathcal{A}_z \mid H(G) = N_y \text{ and } E \text{ and } \mathcal{O} \,\right]$$
$$\geq \Pi_{i \in [2..D]} \left(1 - \frac{h_i}{2^k\chi/2}\right)$$
$$\geq 1 - \frac{\sum_{i \in [2..D]} h_i}{2^k\chi/2} \;\geq\; 1 - \frac{2 \cdot \chi/8}{2^k\chi/2} \;\geq\; \frac{3}{4}$$

because $\sum_{i \in [2..D]} h_i \leq \chi/8$ and E contains at most $\chi/8$, different edges and each edge involves two nodes. □

CLAIM 4. *For any computation path γ in A,*

$$\Pr_{G \in \mathcal{B}_k}[w''_\gamma(G) \geq S/\log n \mid G \in \mathcal{B}_k(E_\gamma)] \leq 2^{-9S}.$$

Proof. Recall that $w''_\gamma(G)$ is the number of roots r in G such that r is a low-collision root with respect to γ and $Prog_{\langle\gamma,r\rangle}(G)$ is true. Hence, the expected value μ of $w''_\gamma(G)$ is

$$\sum_{\text{low-collision roots } r} \Pr_{G \in \mathcal{B}_k}\left[\, Prog_{\langle\gamma,r\rangle}(G) \mid G \in \mathcal{B}_k(E_\gamma) \,\right],$$

and by Claim 3,

$$\mu \leq \sum_{\text{low-collision root } r} \frac{4}{3} \times \frac{y_{\langle\gamma,r\rangle}}{y_{\langle\gamma,r\rangle} + z_{\langle\gamma,r\rangle}}$$
$$\leq \frac{4}{3} \frac{\sum_{\text{low-collision root } r} y_{\langle\gamma,r\rangle}}{f^{D'}}$$
$$\leq \frac{\chi}{6 \cdot f^{D'}},$$

as $\sum_r y_{\langle\gamma,r\rangle} \leq \chi/8$ (at most $\chi/8$ different edges are queried by γ). Since $\chi \in O(n)$ and $f^{D'} \geq n^{10}$, we have $\mu \in O(n^{-9})$. By Lemma 12.1,

$$\begin{aligned}&\Pr_{G\in\mathcal{B}_k}\left[\, w''_\gamma(G) \geq \frac{S}{\log n} \mid G \in \mathcal{B}_k(E_\gamma)\right]\\ &\leq 2^{-(\frac{S}{\log n})(\log(\frac{S}{\mu \log n})-\log e)}\\ &\leq 2^{-9S}. \qquad \square\end{aligned}$$

13. Conclusion. We have proven that any 2-sided probabilistic NNJAG solving the *st*-connectivity problem for n-node graphs in (expected) time T using space S must have $T \in 2^{\Omega(\log^2(n/S))}$ when $S \in O(n^{1-\delta})$ for some $\delta > 0$, and $T \in 2^{\Omega(\log^2(\frac{n\log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$ for general $S \in O(n \log n)$. This greatly improves the previous bounds of $ST \in \Omega(n^2/\log n)$ by Barnes and Edmonds [4] and $S^{1/3}T \in \Omega(n^{4/3})$ by Edmonds [14]. Moreover, the bound is tight for $S \in n^{1-\Omega(1)}$. As a corollary, we also obtained a space lower bound of $\Omega(\log^2 n)$ on a probabilistic NNJAG. No such tight lower bound was known before, even in the more restricted JAG model.

An obvious open problem is to close the gap between the upper and lower bounds when $S \notin n^{1-\Omega(1)}$. However, the major open problem is to prove similar lower bounds on a general model of computation. To achieve that, one possible approach is to start with a JAG/NNJAG-like model and add more and more power, pushing our way towards the ultimate model of the branching program. A major complaint regarding a JAG or NNJAG is its restricted access to the inputs. As pointed out in Etessami and Immerman [17], the space lower bounds of [13, 7, 26] are proven on a tree. However, it is easy for a RAM to solve STCON on trees in $O(\log n)$ space. All it needs to do is to walk a "pebble" from node t backward and see if it hits node s.

In response to this, we define a model called the *Stack NNJAG* that can solve STCON for trees in $O(\log n)$ space, and yet on this model we can still prove the same time–space lower bound. In this model, there is a constant number of *stack* pebbles in addition to those regular pebbles. Each stack pebble has a stack which can remember the path that it has traversed since its last jump. More precisely, all the pebbles, whether regular or stack pebbles, are initially on node s. The stack of each stack pebble is empty initially. Whenever a stack pebble walks along an edge (u, v), the node u is pushed onto the stack. Whenever a stack pebble jumps to another pebble P', it empties its stack. If P' is also a stack pebble, then P copies the stack of P' to its own stack. A stack pebble can also backtrack along the path, i.e., to move to the node v if v is the top of the stack and then pop the stack. Note that the pebble is not allowed to visit any arbitrary node. Any node reachable by a stack pebble must be reachable from s by a directed path. The space for storing the stacks is given for free.

To prove the time–space lower bound, observe that the height of the graph used in our paper is $O(\sqrt{(n\log n)/S})$. If $\sqrt{(n\log n)/S} \leq S/\log n$, each stack can store only at most $O(S/\log n)$ nodes. Since a stack NNJAG has a constant number of stack pebbles, it can be simulated by a normal NNJAG with at most $\Theta(S/\log n)$ extra pebbles. The extra pebbles simply jump to and remain on each node that a stack pebble reaches. This increases the space used by the algorithm by at most $\Theta(S)$. If $\sqrt{(n\log n)/S} \geq S/\log n$, then $S \leq n^{1/3}\log n$. In this case, the bound we have for a normal NNJAG is $T = 2^{\Omega(\log^2 n)}$. Now observe that the height of each stack is at most

the height of the graph, i.e., at most $O(\sqrt{n \log n})$. Hence, any stack NNJAG with space S can be simulated by a normal NNJAG with space $O(S + \sqrt{n \log n}) \in O(\sqrt{n \log n})$, and the same lower bound applies.

Note that the stack NNJAG model seems to be incomparable with a branching program because of the way we charge the space. Also, defining an intermediate model between the NNJAG model and branching program seems hard. For example, allowing the model to move a pebble to an arbitrary node or to the next node in some fixed ordering would give the power of branching programs. Within a polynomial factor of time and constant factor of space, so does allowing it to move a pebble backward along any directed edge [6]. The idea is that one can treat the graph as undirected and, using a universal traversal sequence [1], visit any vertex in polynomial time. Hence, whenever the branching program queries the out-edges of a node v, the enhanced NNJAG can place a pebble on node v (by the universal traversal sequence) and perform the same query on v.

Acknowledgments. We are especially grateful to Greg Barnes for his invaluable insights. We thank Allan Borodin, Faith Fich, Charles Rackoff, and Hisao Tamaki for their helpful discussions and support. We also thank the annonymous referees for their careful reading and helpful comments. Last but not the least, we thank Johan Håstad for pointing out a bug in the original proof of Lemma 12.1.

REFERENCES

[1] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. Rackoff, *Random walks, universal traversal sequences, and the complexity of maze problems*, in Proc. 20th Annual Symposium on Foundations of Computer Science, IEEE, San Juan, PR, 1979, pp. 218–223.

[2] K. B. Athreya and P. E. Ney, eds., *Branching Processes*, Springer-Verlag, Berlin, 1972.

[3] G. Barnes, J. F. Buss, W. L. Ruzzo, and B. Schieber, *A sublinear space, polynomial time algorithm for directed s-t connectivity*, in Proc. 7th Annual IEEE Conference on Structure in Complexity Theory, Boston, MA, 1992, pp. 27–33.

[4] G. Barnes and J. Edmonds *Time-space lower bounds for directed s-t connectivity on JAG models*, in Proc. 34th Annual Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 228–237.

[5] P. Beame, *A general sequential time-space tradeoff for finding unique elements*, SIAM J. Comput. 20 (1991), pp. 270–277.

[6] P. Beame, A. Borodin, P. Raghavan, W. L. Ruzzo, and M. Tompa, *Time-space tradeoffs for undirected graph connectivity*, SIAM J. Comput., 28 (1998), pp. 1051–1072.

[7] P. Berman and J. Simon, *Lower bounds on graph threading by probabilistic machines*, in Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, Tucson, AZ, November 1983, pp. 304–311.

[8] A. Borodin and S. Cook, *A time-space tradeoff for sorting on a general sequential model of computation*, SIAM J. Comput., 11 (1982), pp. 287–297.

[9] A. Borodin, F. Fich, F. Meyer auf der Heide, E. Upfal, and A. Wigderson, *A time-space tradeoff for element distinctness*, SIAM J. Comput., 16 (1987), pp. 97–99.

[10] A. Borodin, M. J. Fischer, D. G. Kirkpatrick, N. A. Lynch, and M. Tompa, *A time-space tradeoff for sorting on non-oblivious machines*, J. Comput. System Sci., 22 (1981), pp. 351–364.

[11] A. Borodin, W. L. Ruzzo, and M. Tompa, *Lower bounds on the length of universal traversal sequences*, J. Comput. System Sci., 45 (1992), pp. 180–203.

[12] A. Borodin, *Structured vs. general models in computational complexity*, Enseign. Math., XXVIII 1982, pp. 171–190. Also in [21, pp. 47–65].

[13] S. A. Cook and C. W. Rackoff, *Space lower bounds for maze threadability on restricted machines*, SIAM J. Comput., 9 (1980), pp. 636–652.

[14] J. Edmonds, *Time-Space Lower Bounds for Undirected and Directed ST-Connectivity on JAG Models*, Ph.D. thesis, University of Toronto, Toronto, ON, Canada, 1993.

[15] J. Edmonds, *Time-space trade-offs for undirected st-connectivity on a JAG*, in Proc. 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 718–727.

[16] J. Edmonds and C. K. Poon, *A nearly optimal time-space lower bound for directed st-connectivity on the NNJAG model*, in Proc. 27th Annual ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 147–156.

[17] K. Etessami and N. Immerman, *Reachability and the power of local ordering*, in Proc. 11th Annual Symposium on Theoretical Aspects of Computer Science, February 1994, Lecture Notes in Comp. Sci. 775, Springer-Verlag, New York, 1994, pp. 123–135.

[18] J. Gill, *Computational complexity of probabilistic Turing machines*, SIAM J. Comput. 6 (1977), pp. 675–695.

[19] N. Immerman, *Nondeterministic space is closed under complementation*, SIAM J. Comput. 17 (1988), pp. 935–938.

[20] D. S. Johnson, *A catalog of complexity classes*, in Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity, Jan van Leeuwan, ed., Elsevier, Amsterdam, 1990, Chap. 2, pp. 67–161.

[21] *Logic and Algorithmic*, An International Symposium Held in Honor of Ernst Specker, Zürich, February 5–11, 1980, Enseign. Math. 30, Université de Genève, Geneva, Switzerland, 1982.

[22] H. R. Lewis and C. H. Papadimitriou, *Symmetric space-bounded computation*, Theoret. Comput. Sci., 19 (1982), pp. 161–187.

[23] N. Nisan, $RL \subseteq SC$. in Proc. 24th Annual ACM Symposium on Theory of Computing, Victoria, BC. Canada, 1992, pp. 619–623.

[24] N. Nisan, E. Szemerédi, and A. Wigderson, *Undirected connectivity in* $O(\log^{1.5} n)$ *space*, in Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science, Pittsburgh, PA, 1992.

[25] N. Pippenger, *The asymptotic optimality of spider-web networks*, Discrete Appl. Math., 37/38 (1992), pp. 437–450.

[26] C. K. Poon, *Space bounds for graph connectivity problems on node-named JAGs and node-ordered JAGs*, in Proc. 34th Annual Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 218–227.

[27] C. K. Poon, *On the Complexity of the ST-Connectivity Problem*, Ph.D. thesis, University of Toronto, Toronto, ON, Canada, 1996.

[28] W. J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.

[29] R. Szelepcsényi, *The method of forcing for nondeterministic automata*, Acta Inform., 26 (1988), pp. 279–284.

[30] M. Tompa, *Two familiar transitive closure algorithms which admit no polynomial time, sublinear space implementations*, SIAM J. Comput., 11 (1982), pp. 130–137.

[31] A. Wigderson, *The complexity of graph connectivity*, in Proc. 17th Symp. Mathematical Foundations of Computer Science, August 1992, I. M. Havel and V. Koubek, eds., Lecture Notes in Comput. Sci. 629, Springer-Verlag, New York, 1992, pp. 112–132.

[32] A. C. Yao, *Probabilistic computations: Toward a unified measure of complexity*, in Proc. 18th Annual IEEE Symposium on Foundations of Computer Science, Providence, RI, 1977, pp. 222–227.

[33] A. C. Yao, *Near-optimal time-space tradeoff for element distinctness*, in Proc. 29th Annual IEEE Symposium on Foundations of Computer Science, White Plains, NY, 1988, pp. 91–97.

SIAM J. COMPUT.
Vol. 28, No. 6, pp. 2285–2299

COMPUTING TWO-DIMENSIONAL INTEGER HULLS*

WARWICK HARVEY†

Abstract. An optimal algorithm is presented for computing the smallest set of linear inequalities that define the integer hull of a possibly unbounded two-dimensional convex polygon R. Input to the algorithm is a set of linear inequalities defining R, and the integer hull computed is the convex hull of the integer points of R. It is proven that the integer hull has at most $O(n \log A_{max})$ inequalities, where n is the number of input inequalities and A_{max} is the magnitude of the largest input coefficient. It is shown that the algorithm presented has complexity $O(n \log A_{max})$ and that this is optimal by proving that the integer hull may have $\Omega(n \log A_{max})$ inequalities in the worst case.

Key words. integer convex hull, linear inequalities, continued fractions

AMS subject classifications. 52C05, 11A55, 68Q25, 90C10

PII. S009753979528977X

1. Introduction and motivation.

In this paper we present an algorithm for finding the integer hull of a possibly unbounded two-dimensional (planar) convex region in polynomial time, given the set of linear inequalities defining the region. By the integer hull of a region we mean the convex hull of the integer points contained in that region. This algorithm grew out of work in the field of constraint logic programming (CLP) [11], specifically integer solvers for CLP. As a result, the algorithm is presented in an incremental formulation (that is, the input is processed one inequality at a time with the integer hull being updated fully after each) because that is what is most suitable for use in a CLP language. While a nonincremental formulation of the algorithm can likely be made to run faster in practice, it will not improve the (worst case) time complexity, since in section 7 we prove our algorithm is optimal.

We see this two-dimensional algorithm as a first step and hope to generalize it to an algorithm for efficiently finding the integer hulls of systems in an arbitrary number of dimensions. There are several reasons why being able to compute the integer hull of a system is useful. The integer hull provides a convenient and concise description of the set of all integer solutions of the input set, which is particularly useful if the number of such solutions is large or even infinite. It also implicitly answers the satisfiability question, which is the basis of constraint solvers for CLP. Thus if incrementally computing the integer hull is efficient enough, it may provide an interesting alternative to the partial solvers for integer CLP used to date (e.g., [7, 4, 10]). Finally, the integer hull allows integer linear optimization to be performed in polynomial time by using real optimization algorithms, though this is likely to be of limited utility unless a number of such optimizations are to be performed on the same feasible set.

In section 2 we discuss existing and related work. In section 3 we discuss some assumptions and notation. Section 4 shows how to find the integer hull of a single pair of inequalities, while section 5 uses this technique to compute the integer hull of

*Received by the editors August 1, 1995; accepted for publication (in revised form) September 18, 1997; published electronically August 16, 1999.
http://www.siam.org/journals/sicomp/28-6/28977.html

†Department of Computer Science, University of Melbourne, Parkville 3052, Australia. Present address: School of Computer Science and Software Engineering, Monash University, Clayton 3168, Australia (wharvey@cs.monash.edu.au).

a full set of inequalities. In section 6 we prove a time bound on the algorithm, and in section 7 we prove optimality. Finally, in section 8 we describe areas for future work.

2. Existing and related work. The only other existing algorithm for computing the integer hull of a polyhedron of which we are aware is presented in Schrijver [16, Chap. 23]. The algorithm works by successive approximation of the integer hull and is guaranteed to terminate after a finite number of such approximation steps. Each step involves finding the minimal total dual integral system describing the current approximation. Since such a system can be exponentially large, clearly each step can take exponential time. Moreover, Schrijver presents an example which shows that, even when restricted to the two-dimensional case, the number of steps taken may be exponential in the size of the problem. Our algorithm solves the given example in linear time.

Kannan [12] gave an algorithm for two-dimensional linear integer optimization in polynomial time (assuming the variables are nonnegative). Lenstra [13] showed that linear integer optimization in any fixed number of dimensions could be done in polynomial time. An interesting question is whether a similar result can be found for the problem of computing the integer hull, i.e., whether it can be done in polynomial time for any fixed number of dimensions.

There exist a number of algorithms for computing the convex hull of a finite set of points. Several algorithms and a review of others can be found in [9] and [15]. They include algorithms specifically for two dimensions, as well as algorithms for an arbitrary number of dimensions. Attempting to use these algorithms to help find the integer hull still leaves the problem of constructing the point set to give them and does not allow the handling of infinite solution sets.

Quite a number of algorithms have been developed for finding a (precise) finite description of all solutions of linear Diophantine equations and inequalities. For example, there are algorithms which focus on finding minimal solutions (e.g., [8]), algorithms which focus on finding nonambiguous solutions (e.g., [1]), and algorithms which focus on avoiding introducing slack variables (e.g., [2]). All of these algorithms resort to complete enumeration when the solution space is finite, which is inefficient if it is particularly large. Whether this matters, and whether the integer hull would be a better representation, depends on the end use for the algorithm.

3. Preliminaries. In the following, we assume that the coefficients of the input inequalities are integers; rational coefficients can be handled by appropriate multiplication. Because we are working in two dimensions, any given inequality C_i (for some i) can be written in the form

$$a_i x + b_i y \leq c_i.$$

For each such inequality, we assume that $\gcd(|a_i|, |b_i|) = 1$. If this is not the case (say, $\gcd(|a_i|, |b_i|) = k_i$), then it can be made so by replacing it with

$$\frac{a_i}{k_i} x + \frac{b_i}{k_i} y \leq \left\lfloor \frac{c_i}{k_i} \right\rfloor.$$

Note that the set of integer points satisfying this replacement inequality is exactly the set of integer points satisfying the original inequality.

We also use $C_i^{=}$ to denote the supporting line of C_i, namely,

$$a_i x + b_i y = c_i.$$

We make use of a number of results from the theory of continued fractions. For an introduction to continued fractions, see, for instance, [6] or [3, Chap. 32]. Briefly, any quantity X can be written as

$$a_1+\cfrac{1}{a_2+\cfrac{1}{a_3+\cfrac{1}{a_4+\cdots}}},$$

where the a_k's (called *partial quotients*) are integers and all denominators are positive. The expansion terminates exactly when X is rational. The *principal convergents* of X are rational approximations p_k/q_k to X, obtained by truncating the continued fraction expansion of X after the kth term. p_k and q_k can be computed using the recurrence relations

$$\begin{array}{lll} p_0 = 1; & p_1 = a_1; & p_k = a_k p_{k-1} + p_{k-2},\ k \geq 2; \\ q_0 = 0; & q_1 = 1; & q_k = a_k q_{k-1} + q_{k-2},\ k \geq 2. \end{array}$$

The principal convergents are "good" approximations in that no simpler fraction is as close to X in value as any particular principal convergent. If any $a_k > 1$, then between p_{k-2}/q_{k-2} and p_k/q_k we define *intermediate convergents*

$$\frac{p_{k-2} + jp_{k-1}}{q_{k-2} + jq_{k-1}},\ j = 1 \cdots a_k - 1.$$

4. Finding the integer hull of a pair of inequalities. Before we look at the full integer hull computation, we first show how to compute the integer hull of a pair of inequalities that are adjacent on the (real) hull. This is a key step in computing the integer hull of an arbitrary set of inequalities, which is described in section 5.

Assume the two inequalities are

$$\begin{aligned} a_1 x + b_1 y \leq c_1 \quad (C_1), \\ a_2 x + b_2 y \leq c_2 \quad (C_2). \end{aligned}$$

Let Δ be the determinant of the coefficient matrix (i.e., $\Delta = a_1 b_2 - b_1 a_2$). Without loss of generality, assume that the (counterclockwise) angle between C_1 and C_2 is less than 180 degrees so that $\Delta > 0$ (otherwise, swap C_1 and C_2).

We now compute the intersection point of $C_1^=$ and $C_2^=$, which will be an extreme point of the (real) feasible region. If this point is integral, then clearly we already have the integer hull and we are done. Otherwise, we need to perform cuts to obtain the integer hull, and we now describe how to compute exactly which cuts are needed to completely define the integer hull.

First, we perform a unimodular[1] transformation from x and y to X and Y such that one of the inequalities is transformed into an inequality in only one variable (say, X), specifically, an inequality of the form $X \leq c$. We would also like the other transformed inequality to have a nonpositive X coefficient and a positive Y coefficient to ensure a standard orientation for later parts of the algorithm. The unimodularity of the transformation ensures that computing the integer hull in XY space is equivalent

[1] A unimodular matrix (transformation) is an integral matrix with determinant ± 1. The main property of interest to us here is that both the transformation and its inverse preserve integrality: they both map integer points to integer points.

to doing it in xy space. To determine the transformation matrix, we thus need to solve

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} = \begin{bmatrix} t & u \\ 1 & 0 \end{bmatrix} \tag{4.1a}$$

such that

$$\text{(unimodularity)} \quad \alpha\delta - \beta\gamma = \pm 1, \tag{4.1b}$$

$$t \le 0, \tag{4.1c}$$

$$u > 0. \tag{4.1d}$$

The transformed inequalities will then be

$$tX + uY \le c_1, \tag{4.2a}$$

$$X \le c_2. \tag{4.2b}$$

LEMMA 4.1. *The transformation matrix*

$$\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix} = \begin{bmatrix} \alpha_0 + kb_2 & b_2 \\ \gamma_0 - ka_2 & -a_2 \end{bmatrix}$$

satisfies the conditions (4.1), *where* α_0 *and* γ_0 *are any integral solution of* $a_2\alpha_0 + b_2\gamma_0 = 1$ *and* $k = \left\lfloor \frac{-a_1\alpha_0 - b_1\gamma_0}{\Delta} \right\rfloor$.

Proof. It is straightforward that (4.1a), (4.1b), and (4.1d) hold. For (4.1c) we have

$$\begin{aligned} t &= a_1\alpha + b_1\gamma \\ &= a_1\alpha_0 + ka_1b_2 + b_1\gamma_0 - kb_1a_2 \\ &= \Delta \left\{ \frac{a_1\alpha_0 + b_1\gamma_0}{\Delta} + \left\lfloor \frac{-a_1\alpha_0 - b_1\gamma_0}{\Delta} \right\rfloor \right\} \\ &\le 0, \text{ since } \Delta > 0. \end{aligned}$$

∎

We now find the integer point on the supporting line of (4.2a) that is immediately on the feasible side of the supporting line of (4.2b), i.e., the point on the line with the largest X coordinate while still being feasible (call it (x_1, y_1)). Given an arbitrary integral point (x_0, y_0) on the supporting line of (4.2a) (which we can find by using, for example, a modified Euclid's algorithm), the point we are after is given by

$$(x_1, y_1) = \left(x_0 + \left\lfloor \frac{c_2 - x_0}{u} \right\rfloor u, y_0 - \left\lfloor \frac{c_2 - x_0}{u} \right\rfloor t \right).$$

We now translate the coordinate system so that this point becomes the new origin:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix} - \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}.$$

This means the inequalities (4.2a) and (4.2b) now become

$$tX' + uY' \le 0, \tag{4.3a}$$

$$X' \le c_2 - x_1. \tag{4.3b}$$

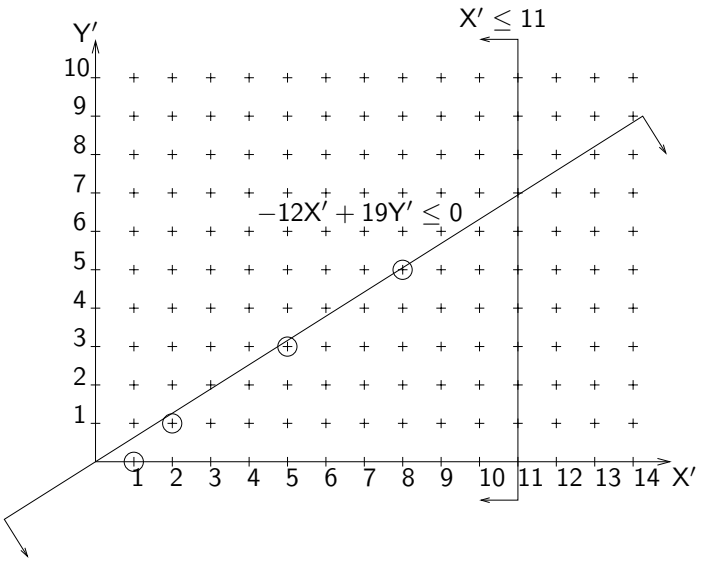


FIG. 4.1. *Two inequality integer hull example: the transformed initial inequalities.*

For example, if the initial inequalities are $5x + 2y \leq 8$ and $-2x + 3y \leq 4$, the transformed inequalities are $-12X' + 19Y' \leq 0$ and $X' \leq 11$, as shown in Figure 4.1.

We are now up to the "interesting" part, where we actually start computing the integer hull. If we think of the boundary of the real region as being a piece of string, and assume each integer point on the plane has a peg in it, finding the integer hull can be thought of as pulling the string tight. To compute this, we essentially need to find the points where the string bends. Due to our construction, there are no integer points on (4.3a) between the origin and where it meets (4.3b), so the first of these articulation points is the origin itself. The main task in finding the next one is determining the gradient of the next segment of "string." This involves rotating this "slack" section of (4.3a) (between the origin and where it meets (4.3b)) clockwise about the origin until it hits the first integer point(s) which satisfy (4.3b). The requirement that it satisfies (4.3b) means we only consider points with X'-coordinate no greater than $c_2 - x_1$. If we look at the problem in terms of the gradients of lines passing through the origin, the feasible integer points correspond to gradients p/q such that $q \leq c_2 - x_1$. Thus we are looking for the fraction p/q which is as close to $-t/u$ as possible (on the "less than" side) with a denominator no greater than $c_2 - x_1$. To find it, we make use of the following theorem.

THEOREM 4.2. *Suppose we are required to find the fraction, whose denominator does not exceed D, which most closely approximates, but is no greater than, the quantity X. If we construct a sequence of fractions containing all the odd principal convergents of X with their corresponding intermediate convergents (if such convergents exist), then the fraction we desire is the element of this sequence with the largest denominator no greater than D.*

Proof. See, for example, [3, Chap. 32, sects. 12–16]. □

Thus the fraction we are looking for can be found by searching the sequence of odd principal convergents for $-t/u$ and the corresponding intermediate convergents (these convergents correspond to the circled points in Figure 4.1). Since the inter-

mediate convergents, if any, have denominators in arithmetic progression between the denominators of the principal convergents on either side, we actually only (construct and) search the sequence of odd principal convergents and then compute the appropriate intermediate convergent directly.

Let p/q be the fraction found by the search. Then this is the gradient of the line from the origin to the first integer point encountered when sweeping the hull segment around. Thus the next segment of the integer hull is given by

$$-pX' + qY' \leq 0. \tag{4.4}$$

Transformed back to the original coordinate system, it is

$$(p\delta + q\gamma)x - (p\beta + q\alpha)y \leq -px_1 + qy_1. \tag{4.5}$$

As the next step in our algorithm, we determine whether this new inequality (4.4) intersects (4.3b) (the vertical inequality) at an integer extreme point. If it does, then we have finished computing the integer hull of the original pair of inequalities. If it does not, then we need to proceed with computing the next cut. If we stay in $X'Y'$ space, this is just the first cut of the integer hull of (4.4) and (4.3b). The inequalities are already oriented appropriately, so we just need to translate them to bring them into our standard form. This involves finding the integer point on the supporting line of (4.4) that has greatest X' coordinate while still being feasible with respect to (4.3b) and moving it to the origin. Note that the "hard" part of finding this integer point, namely, finding an arbitrary point on the line, can be skipped because we already have one: the origin. Thus computing the translation is trivial. We also note that we need not compute the list of convergents for the gradient of (4.4) because it is a prefix of the list of convergents we already computed for $-t/u$ (p/q was basically formed by truncating the continued fraction expansion of $-t/u$).

5. Constructing the full two-dimensional integer hull. We now describe how to find the full two-dimensional integer hull of a set of inequalities in an incremental fashion. Adding the first inequality (to an empty existing set) is straightforward and obvious, so we concentrate on what happens when we add an inequality to an existing integer hull. We keep the inequalities sorted based on orientation (e.g., using the counterclockwise angle between the x-axis and the normal vector of the inequality). The question of what data structure to use to store the inequalities is deferred to section 6, where we analyze the computational complexity of the algorithm.

Since the existing system is an integer hull, we may assume that it is satisfiable and contains no redundant inequalities. Let the inequality being added be $C \equiv ax+by \leq c$. The first thing we do is check whether the augmented system is (real) unsatisfiable. This is equivalent to checking whether $ax + by > c$ (or equivalently $ax + by \geq c + 1$) is (real) redundant with respect to the existing system.

An inequality is (real) redundant if every point that is feasible with respect to every other inequality is also feasible with respect to this inequality. This means that if it is redundant, its supporting line lies on or outside the convex hull of the other inequalities and it is either parallel to the closest edge of the feasible region or there is a closest vertex. For the first case (parallel), we can just check whether there is another inequality with the same orientation and compare right-hand side constants. Otherwise, for the second case, we can find the inequalities that belong just before (C_{i-1}) and just after (C_{i+1}), the inequality being checked (C_i) in the sorted sequence, and see whether these form a vertex which is both feasible with respect to C_i and

also the closest feasible point to $C_i^=$. This is the case if the (counterclockwise) angle between C_{i-1} and C_{i+1} is less than 180 degrees and the intersection of $C_{i-1}^=$ and $C_{i+1}^=$ is feasible with respect to C_i.

If we have discovered that the augmented system is unsatisfiable, then there is no integer hull and we stop. Otherwise, we proceed by checking whether the inequality we are adding is redundant. If it is, then we have nothing to do and the integer hull is unchanged. Otherwise, we proceed to check the inequalities immediately before and after it for redundancy. If the inequality immediately before (after) it is redundant, the redundant inequality is discarded and the next inequality before (resp., after) it is checked. This process continues until an irredundant inequality is found.

Note that at this point the set of inequalities define a convex hull (though most likely not an integer hull), and since all but one inequality (C) was part of the existing integer hull, all the vertices that are not on $C^=$ are guaranteed to be at integral locations.

We consider two cases, depending on whether or not the new inequality has a feasible integer point on its supporting line. The first case is where it does. We can just compute the integer hull of C pairwise with each of the adjacent inequalities, as per section 4 (as long as the relevant angle is less than 180 degrees; if it isn't, the region is unbounded between the inequalities and there are no cuts to compute), and add the resulting cuts to our description of the integer hull. Note that since the new inequality contains a feasible integer point, the pairwise integer hulls are guaranteed not to overlap. This is because at worst there is only one feasible integer point on the supporting line of the new inequality and the two pairwise integer hulls meet at this point. Note that any inequality involved in a nonintegral intersection before the cuts were made may have become redundant (if it only had one feasible integer point on its supporting line to begin with), so this must be checked for and the inequalities must be discarded if they are indeed redundant.

The second case is where there is no feasible integer point on the supporting line of the new inequality. In this case, computing the pairwise integer hulls of the new inequality with the inequalities on either side of it will generate cuts which overlap, and the new inequality is guaranteed to be redundant with respect to the final integer hull. The method described here shows how to deal with these overlapping cuts in a way which ensures that no more than one redundant cut is ever generated.

Let C_i be the new inequality with C_{i-1} and C_{i+1} the inequalities immediately to the clockwise and counterclockwise sides of C_i, respectively. We commence computing the cuts that form the integer hull of C_{i-1} and C_i, as per section 4, and continue until we make a cut (call it C_i') that causes C_i to become (real) redundant.[2] At this point, we now have at most one vertex remaining which is nonintegral, namely, the one at the intersection of C_i' and C_{i+1}. Thus to complete the integer hull, we simply compute the pairwise integer hull of C_i' and C_{i+1}. As before, some of the inequalities need to be checked for (real) redundancy. These are C_{i-1}, C_{i+1}, and C_i' (C_i is guaranteed to be redundant and thus need not be checked, just discarded).

This completes the integer hull algorithm.

6. Computational complexity. We now demonstrate that the algorithm presented has complexity $O(n \log A_{max})$. The time analysis is presented in terms of basic arithmetic operations $(+, -, \times, /)$.

[2] We start with C_{i-1} and C_i rather than C_i and C_{i+1} so that the pairwise hull algorithm generates cuts in an appropriate order ("outside in"), beginning with those adjacent to C_{i-1} and working toward C_i.

We start with the following definitions. Let A_{max} be the maximum absolute value of any coefficient of x or y in any original inequality added to the system.

We start by obtaining bounds for the coefficients of the inequalities that define the integer hull.

In the following, we assume all fractions p/q are in lowest terms (i.e., $\gcd(p,q) = 1$). Let p^+/q^+ (p^-/q^-) be the smallest (resp., largest) rational that is greater than (resp., less than) p/q with a denominator no larger than q.

LEMMA 6.1. *If p_{j-1}/q_{j-1} and p_j/q_j are consecutive principal convergents of p/q, then $p_jq_{j-1} - p_{j-1}q_j = (-1)^j$.*

Proof. See, for example, [3, Chap. 32, sect. 8]. □

LEMMA 6.2. *If p/q and p'/q' are two fractions such that $pq' - p'q = 1$ and $q > 0$, then no fraction can lie between them unless its denominator is greater than the denominator of both of them.*

Proof. See, for example, [3, Chap. 32, Sect. 12]. □

LEMMA 6.3. *Consider the two fractions*

$$\frac{p_{n-1}}{q_{n-1}}, \qquad \frac{p_n - p_{n-1}}{q_n - q_{n-1}}.$$

If $0 < p/q < 1$, then one of these is the closest above approximation (p^+/q^+) of $p/q = p_n/q_n$ and the other is the closest below approximation (p^-/q^-), depending on whether n is odd or even.

Proof. $0 < p/q < 1$ implies that $q > 0$, and also that $n \geq 1$, so that p_{n-1} and q_{n-1} are defined. Thus, using Lemmas 6.2 and 6.1, it is straightforward that these are the closest above and below approximations with denominators no greater than q. If n is odd, then p_{n-1}/q_{n-1} is the above approximation; if n is even, it is the below approximation. □

COROLLARY 6.4. *$pq^+ - p^+q = -1$ and $pq^- - p^-q = 1$ if $0 < p/q < 1$.*

Proof. This is obvious from Lemmas 6.3 and 6.1. □

LEMMA 6.5. *Consider two inequalities C_0 and C_n with $C_1 \cdots C_{n-1}$ being the cuts required to form the pairwise integer hull. Consider any cut C_i ($1 \leq i \leq n-1$). If the magnitude of the larger of the coefficients of C_i is greater than 1, then it is strictly smaller than the largest magnitude coefficient appearing in C_{i-1} and C_{i+1}.*

Proof. If $C_i \equiv a_ix + b_iy \leq c_i$, then we can assume, without loss of generality, that $0 \leq a_i \leq b_i$. However, $b_i > 1$ implies $b_i \neq a_i$ and $a_i \neq 0$ (since $\gcd(a_i, b_i) = 1$), so we have $0 < a_i < b_i$. Let a_i^+/b_i^+ and a_i^-/b_i^- be closest above and below approximations to a_i/b_i, respectively. Let $P_0 = (x_0, y_0)$ and $P_1 = (x_1, y_1)$ be the (integral) intersection points of $C_i^=$ with $C_{i-1}^=$ and $C_{i+1}^=$, respectively, so that $x_0 > x_1$ and $y_0 < y_1$ (see Figure 6.1).Note that $x_0 - x_1 = kb_i$ and $y_1 - y_0 = ka_i$ for some integer $k > 0$. Also note that C_i is the cut required to form the integer hull of C_{i-1} and C_{i+1} and that there are no integer points which are infeasible with respect to C_i that are feasible with respect to C_{i-1} and C_{i+1} (i.e., the introduced cut excludes no feasible integer points). Consider the integer point $P_2 = (x_2, y_2) = (x_0 - b_i^+, y_0 + a_i^+)$. P_2 is infeasible with respect to C_i, so it must be infeasible with respect to at least one of C_{i-1} and C_{i+1}. We consider two possibilities:

1. Suppose P_2 is infeasible with respect to C_{i-1}. This means that the gradient of C_{i-1} must be between $a_i/-b_i$ and $a_i^+/-b_i^+$. By Corollary 6.4 and Lemma 6.2, any gradient that lies between these two has a denominator of magnitude larger than b_i. Since $b_i > a_i$, this means C_{i-1} must have a coefficient (namely, b_{i-1}) larger in magnitude than both a_i and b_i.

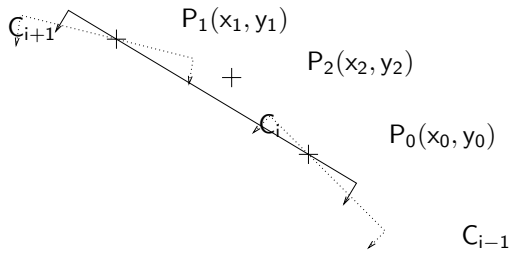


FIG. 6.1. *Determining bounds on the coefficients of adjacent cuts.*

2. Suppose P_2 is infeasible with respect to C_{i+1}. This means that the gradient of C_{i+1} must be between $a_i/-b_i$ and

$$\frac{y_1 - y_2}{-(x_1 - x_2)} = \frac{ka_i - a_i^+}{-(kb_i - b_i^+)}.$$

Again by Corollary 6.4 and Lemma 6.2, any gradient that lies between these two has a denominator of magnitude larger than b_i, which in turn means C_{i+1} must have a coefficient (namely, b_{i+1}) larger in magnitude than both a_i and b_i.

Since at least one of these two possibilities must be true, we have that at least one of C_{i-1} and C_{i+1} must have a coefficient larger in magnitude than those of C_i, as long as at least one of the coefficients of C_i is greater than 1. ∎

THEOREM 6.6. *For the integer hull of a pair of inequalities, the coefficients of the generated cuts are no larger in magnitude than the largest of those of the original inequalities.*

Proof. Let the two inequalities be C_0 and C_n with the generated cuts being $C_1 \cdots C_{n-1}$. Let A_i be the magnitude of the largest coefficient of C_i, $i = 0 \cdots n$. We note that no $A_i = 0$, since that would imply that $C_i \equiv 0x + 0y \leq c_i$. Let j be the smallest i such that A_i is minimal, so that $\forall i, A_i \geq A_j$. If $j > 0$, we have $A_{j-1} > A_j$, so $A_{j-1} > 1$. If $j > 1$, Lemma 6.5 implies $A_{j-2} > A_{j-1}$ since $A_j < A_{j-1}$. Further applications of Lemma 6.5 yield, in turn,

$$A_{j-2} < A_{j-3}, A_{j-3} < A_{j-4}, \ldots, A_1 < A_0.$$

Thus $\forall i \leq j, A_i \leq A_0$.

Let $k = \min\{i : i > j \wedge A_i > A_j\}$. If k is not defined (e.g., $j = n$), then $A_j = A_{j+1} = \cdots = A_n$. Otherwise, we have $A_j = A_{j+1} = \cdots = A_{k-1}$ and $A_k > A_{k-1}$. As before, repeated applications of Lemma 6.5 yield, in turn,

$$A_k < A_{k+1}, A_{k+1} < A_{k+2}, \ldots, A_{n-1} < A_n.$$

Thus $\forall i \geq j, A_i \leq A_n$.

This means that all A_i are no larger than the larger of A_0 and A_n, and the theorem is proved. ∎

THEOREM 6.7. *The coefficients of the inequalities defining an integer hull are no larger in magnitude than the largest coefficient of the input inequalities.*

Proof. Since all the inequalities in the system at any given time are either original input inequalities or are part of a pairwise integer hull of some combination of original

inequalities and inequalities already in the system, the result follows by a simple induction argument on Theorem 6.6. ∎

Now that we have an upper bound on the size of the coefficients in the system, we can obtain an upper bound on the number of inequalities in the system.

LEMMA 6.8. *The integer hull of a pair of inequalities with largest magnitude coefficient A_{max} requires at most $O(\log A_{max})$ inequalities to define it.*

Proof. Each cut generated in the transformed space corresponds to either an odd principal convergent of the gradient $(-t/u)$ of the main transformed inequality or to an intermediate convergent between two such odd principal convergents. To achieve the bound, we start by showing that even if there are many intermediate convergents between two odd principal convergents, only one can correspond to a cut of the integer hull. Consider an intermediate convergent that is used for a cut. If p_{2k-1}/q_{2k-1} and p_{2k+1}/q_{2k+1} are the odd principal convergents it lies between, then it is of the form $\frac{p_{2k-1}+jp_{2k}}{q_{2k-1}+jq_{2k}}$ for some j. Note that j is chosen as large as possible subject to the bound on the denominator, so the "slack" between the denominator and the bound must be less than q_{2k}. Since the translation in preparation for the next cut reduces the bound by (at least) the size of the denominator selected, this means the new bound must be less than q_{2k}. This, in turn, precludes any other intermediate convergents between p_{2k-1}/q_{2k-1} and p_{2k+1}/q_{2k+1} from being used to generate cuts, since such a convergent must have a denominator of at least $q_{2k-1} + q_{2k}$.

Thus we can generate no more than two cuts for each odd principal convergent: one for the odd principal convergent itself and one for a corresponding intermediate convergent.

The number of principal convergents is the same as the number of steps in the Euclidean algorithm for finding g.c.ds, applied to $|t|$ and u. This is $O(\log(\min(|t|, u)))$ [5, p. 811], but

$$\min(|t|, u) \le u = a_1\beta + b_1\delta = a_1b_2 - b_1a_2 = O(A_{max}^2)$$

(this last step by Theorem 6.7). So we have $O(\log A_{max})$ principal convergents and thus $O(\log A_{max})$ cuts generated. ∎

LEMMA 6.9. *When computing the integer hull of n (initial) inequalities with largest magnitude coefficient A_{max}, at most $O(n \log A_{max})$ cuts are added.*

Proof. Each time we add a new inequality to the integer hull, we perform at most two pairwise integer hull computations. By Lemma 6.8, each of these generates $O(\log A_{max})$ inequalities, and so after adding n (initial) inequalities, we can have added no more than $O(n \log A_{max})$ inequalities. ∎

LEMMA 6.10. *The integer hull of a set of any number of inequalities with coefficients of magnitude no larger than A_{max} requires at most $O(A_{max}^2)$ inequalities to define it.*

Proof. If the largest magnitude of an input coefficient is A_{max}, then by Theorem 6.7, the magnitudes of all coefficients in the integer hull are no greater than A_{max}. This means there are at most $O(A_{max}^2)$ different possible combinations of coefficients. Since there are no redundant inequalities in the integer hull, there can be no more than one inequality with a given combination of coefficients, and the result follows. ∎

We now proceed to analyze the time complexity of the algorithm. We start by analyzing the pairwise integer hull, and then we use that analysis to derive a time bound for the full algorithm.

THEOREM 6.11. *Computing the pairwise hull of two inequalities with largest magnitude coefficient A_{max} is $O(\log A_{max})$.*

Proof. To compute the integer hull of a pair of inequalities, the following steps are performed:

1. *If the intersection point is integral, we stop.* Finding the intersection point and determining whether it is integral is $O(1)$.

2. *Determine the transformation matrix.* The main step in determining the transformation matrix is finding a solution to $a_2\alpha + b_2\gamma = 1$; everything else is $O(1)$. Thus this step is $O(\log A_{max})$, using, for instance, a modified Euclid's algorithm to solve the equation.

3. *Compute the odd principal convergents.* We just compute all the principal convergents of $|t/u|$; this is $O(\log(\min(|t|, |u|))) = O(\log A_{max})$.

4. *Repeat the following steps until the intersection point is integral.* Each pass through this loop generates one cut, and so by Lemma 6.8, we loop no more than $O(\log A_{max})$ times.

(i) *Determine the appropriate translation.* The first time we perform this translation we must find an initial integer point on the supporting line of the "main" inequality, which is $O(\log A_{max})$, and then find the correct integer point, which is $O(1)$. All subsequent translation computations start with an integer point given, so we just require the $O(1)$ adjustment.

(ii) *Search for the appropriate principal convergent.* If we do a simple linear search backwards in the list of principal convergents, this is $O(\log A_{max})$. However, no principal convergent checked in one pass needs be checked again in a subsequent pass, so we can amortize the search cost for a total of $O(\log A_{max})$ over all passes through the loop.

(iii) *Compute the appropriate intermediate convergent.* This is $O(1)$.

(iv) *Transform the new cut back to the original coordinate system.* This is $O(1)$.

The result follows directly from the above analysis. □

We now turn to the full integer hull algorithm.

THEOREM 6.12. *Incrementally computing the integer hull of n inequalities with largest magnitude coefficient A_{max} is $O(n \log A_{max})$.*

Proof. For this analysis, we assume the inequalities in the system are stored in a level-linked (a,b)-tree, sorted based on their orientation. Level-linked (a,b)-trees are described in [14], along with proofs of the complexity results used here. Some modification of the standard operations on level-linked (a,b)-trees is required for our purposes, since our data is inherently circular in nature and wraps around from largest to smallest, but these do not affect the results.

Let N_j be the number of inequalities defining the existing integer hull just before we add the jth inequality ($C_j \equiv a_j x + b_j y \le c_j$). By Lemma 6.10, N_j is $O(A_{max}^2)$. In particular, this means that $\log N_j$ is $O(\log A_{max})$, and thus we have bounds for various tree operations (search, split, concatenate) which are independent of the number of inequalities added so far.

To add the inequality to the system and recompute the integer hull, we perform the following steps:

1. *Tighten the inequality being added.* This step consists mainly of computing the g.c.d. of $|a_j|$ and $|b_j|$, which is $O(\log(\min(|a_j|, |b_j|))) = O(\log A_{max})$.

2. *Check satisfiability of augmented system.* This requires searching the data structure to find where the complement of C_j would go and accessing at most two adjacent items. This is $O(\log A_{max})$, and the actual check is $O(1)$.

3. *Eliminate initial redundancy.* This requires searching the data structure to find where C_j would go ($O(\log A_{max})$) and then performing a series of redundancy checks. Since no inequality can be found to be redundant more than once, and at most $O(n \log A_{max})$ inequalities are added to the system for n input inequalities (Lemma 6.9), we have a total of $O(n \log A_{max})$ eliminations performed, summed over all n incremental steps. Since all but a constant number of redundancy checks at each incremental step result in an elimination, this means $O(n \log A_{max})$ redundancy checks are performed in total. Accessing the adjacent inequalities to perform one of these checks is $O(1)$, as is the actual check, which means the total cost of redundancy checks over all n additions is $O(n \log A_{max})$. Since all the inequalities to be eliminated due to redundancy are adjacent to each other, we can delete them all at the same time by performing two splits of the tree, which is $O(\log A_{max})$ (we leave it split until we are ready to insert the new inequalities). Thus this step as a whole, summed over all n additions, is $O(n \log A_{max})$.

4. *Compute the cuts.* We treat this as two instances of finding the integer hull of a pair of inequalities. This is not strictly the case when there are no integer points on the feasible segment of the supporting line of the inequality being added, but the extra redundancy checks required have no bearing on the computational complexity. Thus this step is $O(\log A_{max})$, by Theorem 6.11.

5. *Add the cuts to the system.* Since all the inequalities added will be adjacent to each other, and will be inserted at the point the existing tree was split, we can insert them all at the same time. Constructing a new tree from a sorted set is linear in the number of items in the tree, so this is $O(\log A_{max})$ by Lemma 6.8. Then we just perform two concatenation operations to add the new tree between the two parts created in step 3. This is also $O(\log A_{max})$, so the step as a whole is $O(\log A_{max})$.

6. *Eliminate final redundancy.* This requires at most three redundancy checks and three deletions and hence is $O(\log A_{max})$.

Each step is either $O(\log A_{max})$ for each inequality added or is $O(n \log A_{max})$ summed over all n inequalities. Outputting the inequalities is linear in the number output, which is $O(n \log A_{max})$ by Lemma 6.9. The result follows. ∎

7. Proof of optimality. We now show that our algorithm is optimal by demonstrating a worst case lower bound complexity of $\Omega(n \log A_{max})$. We do this by constructing a family of examples which generate $\Omega(n \log A_{max})$ output constraints.

LEMMA 7.1. *The integer hull of*

$$-2x + y \le -1, \tag{7.1a}$$

$$-F_{2k+5}x + F_{2k+4}y \le -1 \tag{7.1b}$$

has k cuts, where F_n is the nth Fibonacci number (with $F_0 = 0, F_1 = 1$).

Proof. Let $\phi = \frac{1+\sqrt{5}}{2}$ be the golden ratio. The continued fraction representation of ϕ is infinite with all partial quotients 1. Thus the sequence of closest below approximations to ϕ are given by p_{2j+1}/q_{2j+1} for $j \ge 0$, where p_i, q_i satisfy the recurrence relations

$$\begin{array}{lll} p_0 = 1; & p_1 = 1; & p_i = p_{i-1} + p_{i-2}, i \ge 2; \\ q_0 = 0; & q_1 = 1; & q_i = q_{i-1} + q_{i-2}, i \ge 2. \end{array}$$

Clearly, the p_i's and q_i's both form the Fibonacci sequence with $p_i = F_{i+1}$ and $q_i = F_i$. The below approximations are thus given by $F_{2j+2}/F_{2j+1}, j \ge 0$.

If we were to compute the (infinite) integer hull of $-\phi x + y \le 0$ and $-x \le -1$ (see Figure 7.1), then these below approximations clearly define the vertices of the integer

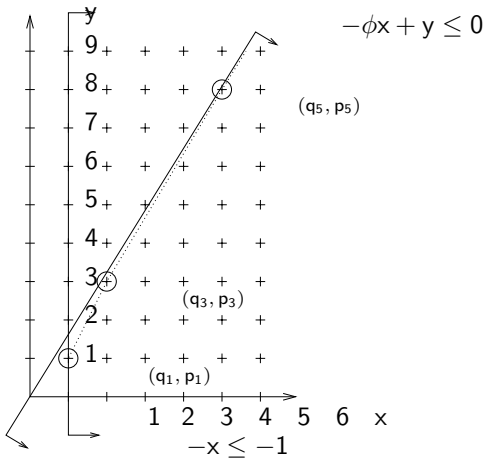


FIG. 7.1. *The infinite integer hull of* $-\phi x + y \leq 0$ *and* $-x \leq -1$.

hull. The inequality defining the jth edge (the edge between the jth and $(j+1)$th vertices) is then

$$-(p_{2j+3} - p_{2j+1})x + (q_{2j+3} - q_{2j+1})y \leq -(p_{2j+3} - p_{2j+1})q_{2j+1} + (q_{2j+3} - q_{2j+1})p_{2j+1},$$

i.e.,
$$-p_{2j+2}x + q_{2j+2}y \leq -p_{2j+2}q_{2j+1} + q_{2j+2}p_{2j+1},$$

i.e.,
$$-F_{2j+3}x + F_{2j+2}y \leq -1$$

(the last step by Lemma 6.1).

Note that we can select any two of these inequalities (say, $j = j_0$ and $j = j_1$), and the cuts needed to form the integer hull of the selected pair consist simply of all the inequalities between them in the above integer hull (namely, $j = j_0 + 1 \ldots j_1 - 1$). In particular, we can choose $j_0 = 0$ and $j_1 = k + 1$ and have the integer hull contain k cuts. The result follows. □

LEMMA 7.2. *It is possible for a system with* $2m$ *inequalities with largest magnitude coefficient* $O(m\phi^{2k})$ *to require at least* mk *cuts to complete the integer hull.*

Proof. We note that we can perform both translations and unimodular transformations on the integer hull used in Lemma 7.1 without altering its basic structure. Applying the unimodular transformation $\left[\begin{smallmatrix} 1 & 0 \\ -i & 1 \end{smallmatrix}\right]$, we obtain

$$-(2+i)x + y \leq -1, \tag{7.2a}$$

$$-(F_{2k+5} + iF_{2k+4})x + F_{2k+4}y \leq -1. \tag{7.2b}$$

Note that, since $F_{2k+5}/F_{2k+4} > 1$, both the inequalities (and all the cuts) have gradients larger than $1 + i$, and no greater than $2 + i$, which means that the range of gradients for one value of i does not overlap with that of another. This means that, with appropriate translation, we can construct a system of $2m$ inequalities, by including a "copy" of (7.2) for all values of i from 0 to $m - 1$, such that the integer hulls of each component do not interfere with each other. By Lemma 7.1, each component has k cuts, so with m noninterfering components, we have (at least) mk cuts (depending on the exact translation used, there may be cuts between each "block").

The magnitude of the largest coefficient of the system is $F_{2k+5} + (m-1)F_{2k+4}$. Using the closed form expression for F_j in terms of ϕ ($F_j = \frac{1}{\sqrt{5}}(\phi^j - (-\phi)^{-j})$), we

have that this is $O(m\phi^{2k+4}) = O(m\phi^{2k})$, and the result follows. ∎

THEOREM 7.3. *It is possible for a system of n inequalities with coefficients of magnitude no larger than A_{max} to have an integer hull consisting of $\Omega(n \log A_{max})$ inequalities.*

Proof. Consider Lemma 7.2 with $k = \Omega(\log m)$. Then $n = 2m$, $A_{max} = O(m\phi^{2k})$, and we have mk cuts. Thus $n \log A_{max} = O(m(k + \log m))$. But $k = \Omega(\log m)$, so $n \log A_{max} = O(mk)$. This means $mk = \Omega(n \log A_{max})$ and we have the result. ∎

THEOREM 7.4. *Our algorithm for computing two-dimensional integer hulls is optimal.*

Proof. By Theorem 7.3, any algorithm must be $\Omega(n \log A_{max})$ on some problem instances. By Theorem 6.12, our algorithm is $O(n \log A_{max})$ on all problem instances, and the result follows. ∎

8. Future work. We have done some work toward developing an integer hull algorithm that works in three dimensions, but we have yet to establish whether this algorithm can be made to be polynomial time. If it can, then we hope to generalize to an arbitrary number of dimensions and tackle the very interesting question of whether such an algorithm is polynomial time for any fixed number of dimensions.

At some stage we also hope to determine whether such algorithms can be usefully employed in a constraint solver for a CLP language by buying enough of a reduction in domain sizes to offset the extra overhead incurred on more than just a few select problem classes.

Acknowledgments. The geometrical interpretation of continued fractions and their convergents used in this paper is from [6, Chap. IV, sect. 12] and appears to be due to Smith [17, Art. 20].

The author would like to thank Peter Stuckey for his comments on many drafts of this paper. The author would also like to thank the anonymous referees for their feedback, advice, and suggestions. In particular, credit must be given to one anonymous referee for a suggestion on how to obtain improved bounds on the coefficients of generated cuts (the bounds presented in an earlier version of this work were messy and not particularly tight, with the proofs hard to follow). While the ideas proposed by the referee are not used in the paper, the bounds yielded were sufficiently good for the whole question of optimality to be addressed and provided the motivation to find the proof of even tighter bounds that is presented in this version of the paper.

Finally, the author would like to thank Gary Eddy for his help in selecting an appropriate data structure for storing the inequalities.

REFERENCES

[1] H. ABDULRAB AND M. MAKSIMENKO, *General solution of systems of linear diophantine equations and inequations*, in Rewriting Techniques and Applications—6th International Conference, RTA-95, J. Hsiang, ed., Lecture Notes in Comput. Sci. 914, Springer-Verlag, Berlin, 1995, pp. 339–351.

[2] F. AJILI AND E. CONTEJEAN, *Complete solving of linear diophantine equations and inequations without adding variables*, in Principles and Practice of Constraint Programming—CP '95, U. Montanari and F. Rossi, eds., Lecture Notes in Comput. Sci. 976, Springer-Verlag, Berlin, 1995, pp. 1–17.

[3] G. CHRYSTAL, *Algebra–An Elementary Text-Book–Part* II, Adam and Charles Black, Edinburgh, 1889.

[4] P. CODOGNET AND D. DIAZ, *Compiling constraints in clp(FD)*, J. Logic Programming, 27 (1996), pp. 185–226.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Electrical Engineering and Computer Science Series, MIT Press, Cambridge, MA, 1990.

[6] H. Davenport, *The Higher Arithmetic*, 6th ed., Cambridge University Press, Cambridge, UK, 1992.

[7] M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier, *The constraint logic programming language CHIP*, in Proceedings of the International Conference on Fifth Generation Computer Systems FGCS-88, Tokyo, Japan, Springer-Verlag, New York, 1988, pp. 693–702.

[8] E. Domenjod and A. P. Tomàs, *From Elliott-MacMahon to an algorithm for general linear constraints on naturals*, in Principles and Practice of Constraint Programming—CP '95, U. Montanari and F. Rossi, eds., Lecture Notes in Comput. Sci. 976, Springer-Verlag, Berlin, 1995, pp. 18–35.

[9] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.

[10] W. Harvey and P. J. Stuckey, *A unit two variable per inequality integer constraint solver for constraint logic programming*, in Proceedings of the Twentieth Australasian Computer Science Conference (ACSC'97), Sydney, Australia, Macquarie University, Sydney, 1997, pp. 102–111.

[11] J. Jaffar and M. J. Maher, *Constraint logic programming: A survey*, J. Logic Programming, 19/20 (1994), pp. 503–581.

[12] R. Kannan, *A polynomial algorithm for the two variable integer programming problem*, J. ACM, 27 (1980), pp. 118–122.

[13] H. W. Lenstra, Jr., *Integer programming with a fixed number of variables*, Math. Oper. Res., 8 (1983), pp. 538–547.

[14] K. Mehlhorn, *Data Structures and Algorithms* 1*: Sorting and Searching*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin, 1984.

[15] F. P. Preparata and M. I. Shamos, *Computational Geometry—An Introduction*, Texts Monogr. Comput. Sci., Springer-Verlag, New York, 1985.

[16] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley-Interscience Series in Discrete Mathematics, Wiley-Interscience, New York, 1986.

[17] H. J. S. Smith, *A note on continued fractions*, in The Collected Mathematical Papers of Henry John Stephen Smith, vol. 2, J. W. L. Glaisher, ed., Clarendon Press, Oxford, UK, 1894, pp. 135–147.